opentext™ | Cybersecurity

Position Paper

Developer Guide to the 2023 OWASP Top 10 for API Security

Table of Contents

Developer Guide to the 2023 OWASP Top 10 for API Security

As companies have adopted cloud-native infrastructure and DevOp-style methodologies, Web application programming interfaces, or APIs, have proliferated. Some of the most popular public APIs include those that allow developers to access Google Search, scrape data from TikTok, track vehicles, gather sports scores, and collect data on image downloads from popular sites.[1] In 2023, API-related traffic accounts for 58% of all dynamic—defined as non-cacheable—traffic, up from 54% at the end of 2021.[2]

APIs have become the way for enterprise applications to communicate and integrate with each other as well. Companies use about two-thirds of their APIs (64%) to connect their applications to partners, while about half (51%) are access points to microservices. Overall, more than three-quarters of firms use an average of at least 25 APIs per application.[3]

The adoption of API-based application infrastructure should come as no surprise: Companies that adopt APIs to attract third-party developers and create ecosystems see increased growth. These "inverted firms"—so called because they flip the traditional concepts of creating barriers around technologies and allow open access to some capabilities and data—grew by nearly 13% over two years, and 39% over 16 years, compared to firms who did not adopt APIs, according to a 2022 paper by researchers at Chapman University and Boston University.[4]

With the adoption of microservices, containerization, and APIs, however, comes a variety of risks, such as insecure software components, poor business logic, and flawed data security. Nine-in-ten organizations (92%) have suffered at least one security incident related to insecure APIs.[5] Large companies typically have thousands of APIs and attacks on those systems account for about 20% of security incidents, while smaller companies have hundreds of APIs whose smaller attack surface accounts for 5% of security incidents.[6] Annual losses due to breaches caused by API vulnerabilities exceed $40 billion globally, according to an estimate by Marsh McLennan.[7]

The problem is so serious that the US National Security Agency teamed up with the Australian Cyber Security Centre (ACSC) and the U.S. Cybersecurity and Infrastructure Security Agency (CISA) to offer guidance on API security issues, especially the most common, known as insecure direct object reference (IDOR) vulnerabilities.[8]

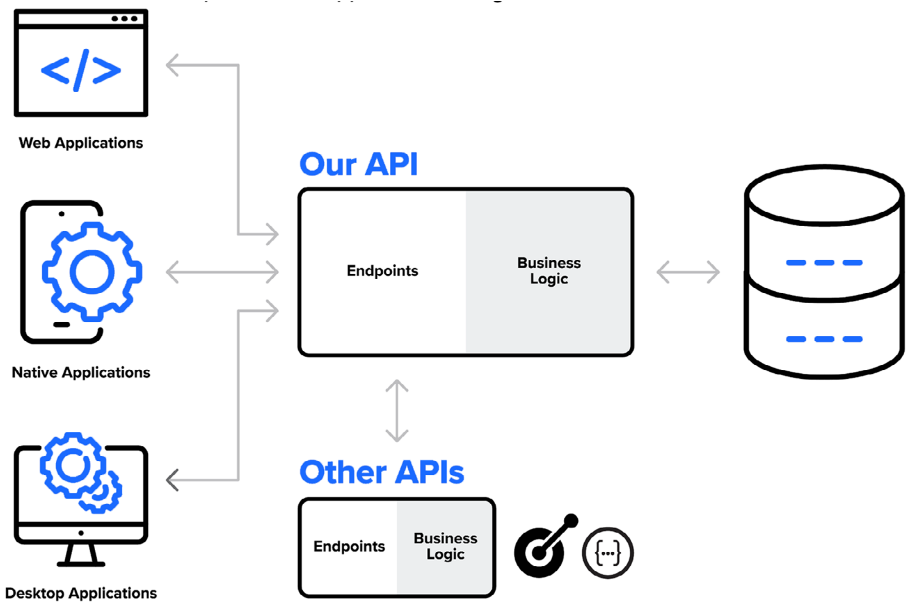
Unsurprisingly, against this backdrop of burgeoning security concerns, the Open Worldwide Application Security Project (OWASP) released an update to its API Security Top-10 list.

1. Arellano, Kelly. "The Top 50 Most Popular APIs." RapidAPI Blog. RapidAPI. Web Page. 16 March 2023. **https://rapidapi.com/blog/most-popular-api/**.
2. Tremante, Michael, et al. "Application Security Report: Q2 2023." Cloudflare Blog. Cloudflare. Blog post. 21 Aug 2023. **https://blog.cloudflare.com/application-security-report-q2-2023**/.
3. Marks, Melinda. "Securing the API Attack Surface." Enterprise Strategy Group. Sponsored by Palo Alto Networks. PDF Report, p. 10. 23 May 2023. **www.paloaltonetworks.com/resources/research/api-security-statistics-report**.
4. Benzell, Seth G., et al. "How APIs Create Growth by Inverting the Firm." Social Science Research Network. Research Paper. Revised: 30 Dec 2022. **https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3432591**.
5. "Securing the API Attack Surface." Enterprise Strategy Group, p. 14.
6. Lemos, Robert. "API Security Losses Total Billions, But It's Complicated." Dark Reading. News Article. 30 June 2022. **www.darkreading.com/application-security/api-security-losses-billions-complicated**.
7. Marsh McLennan. "Quantifying the Cost of API Insecurity." Sponsored by Imperva. PDF Report. 22 June 2022. **www.imperva.com/resources/reports/Imperva-Marsh-McLennan-Report-2022.pdf**.
8. "New Cybersecurity Advisory Warns About Web Application Vulnerabilities." National Security Agency. Press Release. 27 July 2023. **www.nsa.gov/Press-Room/Press-Releases-Statements/Press-Release-View/Article/3473830/new-cybersecurity-advisory-warns-about-web-application-vulnerabilities/**.

Refreshing its inaugural 2019 list, the 2023 API Security Top-10 list highlights the ten most common and serious security risks created when developing applications that expose or use APIs. Issues such as Broken Object-Level Authorization, a superset that includes IDOR vulnerabilities, remains the same from the prior list. Yet, new categories—or reorganized categories—now highlight issues overlooked in the past, such as Server-Side Request Forgery (API7:2023) and Unrestricted Access to Sensitive Business Flows (API6:2023).

"By nature, APIs expose application logic and sensitive data such as Personally Identifiable Information (PII) and because of this, APIs have increasingly become a target for attackers," the OWASP group stated in its announcement.[9] "Without secure APIs, rapid innovation would be impossible."

The 2023 API Security Top-10 list highlights the ten most common and serious security risks created when developing applications that expose or use APIs.



API Security Cheat Sheet

OWASP Top 10 Category	Fortify/Cybersecurity Solution
1. Broken Object Level Authorization	SAST
2. Broken Authentication	SAST, DAST
3. Broken Object Property Level Authorization	SAST, DAST
4. Unrestricted Resource Consumption	SAST, DAST, Secure API Manager
5. Broken Function Level Authorization	SAST
6. Unrestricted Access to Sensitive Business Flows	DAST
7. Server Side Request Forgery	DAST
8. Security Misconfiguration	SAST, DAST
9. Improper Inventory Management	Secure API Manager
10. Unsafe Consumption of APIs	SCA, SAST

9. Open Worldwide Application Security Project. "OWASP API Security Top 10: Forward." OWASP.org. Web Page. 3 July 2023. **https://owasp.org/API-Security/editions/2023/en/0x02-foreword/**.

Definitions

API Endpoint—The point of communication between two systems, typically a URL of a container or server running a microservice. Using an URL, an application or developer can request information from the server or execute an action on the API server or microservice.

API-Related Traffic—Internet traffic that consists of an HTTP or HTTPS request and has a response content of XML or JSON, indicating that data is being passed to an application, usually through SOAP, WSDL, a REST API, or gRPC *(see below)*.

Dynamic Application Security Testing (DAST)—The process of analyzing an application or API server by using the interface, whether the user interface for an application, a web front end for a web application, or URLs for API endpoints. At type of black-box testing, this approach evaluates an application from the "outside in" by attacking an application in the same way as an attacker, usually without knowledge of internal processes.

Static Application Security Testing (SAST)—An approach to application security that scans the source, binary or byte code for recognized patterns of errors or vulnerabilities. Sometimes referred to as white-box testing, SAST uses an "inside-out" approach that identifies potential vulnerabilities and errors that may, or may not, be exploitable by an external attacker. Lightweight static tools can provide real-time feedback to developers in their IDE.

SOAP/WSDL—An XML-based protocol for creating Web APIs. SOAP is the protocol itself and WSDL (Web Service Definition Language) is the format used to formally describe services. Due to the heavy overhead, this API style has become unpopular for new developments.

REST—A Web API style that involves exchanging messages directly over HTTP, using the semantics of HTTP URLs and verbs, without using an additional "envelope". The content is usually encoded as JSON, although in some cases it is XML.

GraphQL—A query language designed to be used in APIs (with requests and responses in JSON), together with server-side runtimes to execute these queries. It allows clients to define the structure of data they need and then receive this from the server in that format.

gRPC—An API protocol that is more high performant than REST. It uses HTTP/2 and the performance advantages that offers over HTTP/1.1. The format of the individual messages is usually binary and based on ProtoBuf, again creating performance advantages over REST and SOAP.

2023 API Security Top 10	Analogous 2019 API Security Entry
API1:2023—Broken Object Level Authorization	API1:2019—Broken Object Level Authorization
API2:2023—Broken Authentication	API2:2019—Broken User Authentication
API3:2023—Broken Object Property Level Authorization	API3:2019—Excessive Data Exposure, API6:2019—Mass Assignment
API4:2023—Unrestricted Resource Consumption	API4:2019—Lack of Resources & Rate Limiting
API5:2023—Broken Function Level Authorization	API5:2019—Broken Function Level Authorization
API6:2023—Unrestricted Access to Sensitive Business Flows	
API7:2023—Server Side Request Forgery	
API8:2023—Security Misconfiguration	API7:2019—Security Misconfiguration
API9:2023—Improper Inventory Management	API9:2019—Improper Assets Management
API10:2023—Unsafe Consumption of APIs	API8:2019—Injection, API10:2019—Insufficient Logging & Monitoring

Source: **https://owasp.org/API-Security/editions/2023/en/0x11-t10/**
Source: **https://owasp.org/API-Security/editions/2019/en/0x11-t10/**

API1:2023—Broken Object Level Authorization

What Is It?

APIs allow access to services and data using standardized web requests. Companies expose their infrastructure and data to insecure direct access when those assets are not well protected or when the authorization controls are poorly implemented or absent. **Broken Object Level Authorization**—also referred to as Insecure Direct Object Reference (IDOR)—can lead to a variety of risks, from data disclosure to full account takeover.

What Makes an Application Vulnerable?

This is a **widespread** and **easy-to-exploit** issue in web applications. Applications are vulnerable if they allow a user to take actions by specifying an identifier in an API without checking whether they have authorization to take those actions.

In an example detailed by OWASP, a platform for online stores could allow access to shop data using a simple call:

```
/shops/{shopName}/revenue_data.json
```

This is insecure because any user can replace the shopName with the name of another user's store, gaining access to data they should not have.

Attack Examples

In 2021, a security researcher found that the web-application and back-end servers that provided data to Peloton exercise bikes had several API endpoints—such as **https://api.onepeloton.co.uk/stats/workouts/details**—that allowed unauthenticated users to access private data. In February 2021, Peloton implemented a partial fix for the issue, limiting API access to authenticated users, but still allowing those users to access any private data for other members. A full fix came in May 2021.[10]

Broken Object Level Authorization is a **widespread** and **easy-to-exploit** issue in web applications because API calls carry state information. Applications are vulnerable if they allow a user to take actions by specifying an identifier in an API without checking whether they have authorization to take those actions.

10. Masters, Jan. "Tour de Peloton: Exposed user data." Pen Test Partners Blog. Pen Test Partners. Web Page. 5 May 2021. **www.pentestpartners.com/security-blog/tour-de-peloton-exposed-user-data**/.

How to Prevent It as a Developer?

Developers prevent insecure access to objects by enforcing strict controls, assigning unpredictable user identifiers to dissuade enumeration of accounts, and checking object-level authorization for every function that accesses a data source. Developers should encapsulate such checks, especially if based on user input, to remove the possibility that inadvertent errors could undermine security. Application-security and operations professionals should require authorization checks for each request to backend data.

How Can Fortify Help?

Fortify SAST and DAST by OpenText can detect a broad range of vulnerabilities in the Insecure Direct Object Reference (IDOR) category. IDOR can include vulnerabilities such as Directory Traversal, File Upload, and File Inclusion. More generally, IDOR also includes classes of vulnerabilities where identifiers can be modified via URL, Body, or Header manipulation. The system will alert developers to cases where the user can directly choose the primary key in the API request for a database or storage container, a problem that often leads to this class of vulnerabilities. The system will also warn when an expected authorization check is missing.

Developers and application-security teams also must properly implement capabilities to check user identity through authentication.

API2:2023—Broken Authentication

What Is It?

Authorization checks limit access to data based on specific roles or users, but those limitations are not sufficient to protect systems, data, and services. Developers and application-security teams also must properly implement capabilities to check user identity through authentication. Despite the critical nature of authentication, the components are often poorly implemented or improperly used—the root causes of Broken User Authentication. **Broken user authentication** allows attackers the ability to assume other user's identities temporarily or permanently by exploiting insecure authentication tokens or compromising implementation flaws.

What Makes an Application Vulnerable?

This **common** and **easy-to-exploit** issue occurs because authentication is a complex process that can be confusing and is, by definition, exposed to the public. Developer mistakes and application misconfigurations can result in a lack of necessary checks allowing attackers to avoid authentication. Developers who fail to implement authentication for a particular endpoint or allow weak authentication mechanism expose applications to a variety of attacks, such as credential stuffing, token replay, or password sniffing.

Attack Examples

Between February and June 2023, credential stuffing attacks targeted clothing retailer Hot Topic, who notified its customers that an unknown number of accounts had been compromised. The attackers—using credentials harvested from unknown sources—were able to access sensitive personal data, such as customers' names, email addresses, order histories, phone numbers, and months and days of birth.[11]

11. Toulas, Bill. "Retail chain Hot Topic discloses wave of credential-stuffing attacks." BleepingComputer. News article. 1 Aug 2023. **www.bleepingcomputer.com/news/security/retail-chain-hot-topic-discloses-wave-of-credential-stuffing-attacks/**.

In February 2022, a misconfigured cloud storage bucket left 1 GB of sensitive data from email marketing service Beetle Eye without password protection or encryption. The data included contact information and tourism-related information collected by various tourist agencies and US states.[12] Misconfigured authentication mechanisms are considered a variant of the Broken User Authentication category.

How to Prevent It as a Developer?

Standardization is your friend for authentication. DevSecOps teams should create one—or a limited number—of authentication methods for applications and ensure that developers uniformly implement the mechanisms across all microservices and APIs. Any authentication implementation should be reviewed within the context of the OWASP Application Security Verification Standard (ASVS), currently at version 4[13], to ensure the correctness of the implementation and associated security controls. Any deviation from the standard—especially any intentional exposure of unauthenticated endpoints—should be evaluated by the security team and only allowed to satisfy a strong business requirement.

Standardization is your friend for authentication. DevSecOps teams should create one—or a limited number—of authentication methods for applications and ensure that developers uniformly implement the mechanisms across all microservices and APIs.

How Can Fortify Help?

OAuth and JWT are two of the most common types of authentication used to implement APIs, and Fortify WebInspect by OpenText has checks for weak implementations of both standards in applications, as well as misconfigurations and vulnerable patterns, such as CSRF and Session Fixation, that come up in custom authentication implementations. Dynamic Application Security Tool (DAST) Scanning by OpenText is a great way to detect authentication vulnerabilities, especially in an API.

Fortify SAST by OpenText allows a wide range of checks relating to poor authentication as well. The static analysis tool includes detection for generic issues—such as credential leakage—as well as highly API-specific problems like missing protection claims in JWT tokens, or claims occurring in JWT headers.

API3:2023—Broken Object Property Level Authorization

What Is It?

Broken Object Property Level Authorization is a new category in the 2023 OWASP list that combines two categories from the previous list: Excessive Data Exposure (API3:2019) and Mass Assignment (API6:2019). The issue is caused by the lack of validation of a user's authorization—or the improper authorization of a user—at the object-property level. API endpoints should validate that each user has authorization for every property that they are trying to access or change. Exploiting the issue can lead to information exposure or manipulation of data by unauthorized parties.

12. Nair, Prajeet. "Data of 7 Million People Exposed Via US Marketing Platform." Data Breach Today. ISMG Network. 11 Feb 2022. **www.databreachtoday.com/data-7-million-people-exposed-via-us-marketing-platform-a-18502**.
13. "OWASP Application Security Verification Standard." OWASP. GitHub page. Last accessed: 17 November 2023. **https://github.com/OWASP/ASVS**.

What Makes an Application Vulnerable?

The **common** and **easy-to-exploit** issue occurs when a user may be authorized to access some properties of a specific object, such as reserving a room in travel application, but not others, such as the price of a room. When the user accesses an object's properties through an API, the application should check that the user:

- Should be able to gain access to the specific property of the object (violations were previously known as **Excessive Data Exposure**), and/or
- Is allowed to change the specific property of the object (some applications fail to check this because they use a framework to automatically map web request parameters to object fields, a problem known as **Mass Assignment**).

In an OWASP example, an online video platform allows a user to change the description of a video, even a blocked video, but should not allow the user to modify the 'blocked' property.

```
PUT /api/video/update_video
{
 "description": "a funny video about cats",
 "blocked": false
}
```

Broken Object Property Level Authorization is a new category in the 2023 OWASP list that combines two categories from the previous list: Excessive Data Exposure (API3:2019) and Mass Assignment (API6:2019).

Attack Examples

In January 2022, a bug bounty program discovered a flaw in Twitter that allowed a user to submit an email address or phone number to Twitter's system, which would then return the account name to which the information belonged.[14] An unknown attacker used the flaw to compile a list of millions of user accounts linked to phone numbers and email addresses. By allowing anyone to link two properties, Twitter inadvertently allowed pseudonymous users to be more specifically identified.

Fortify SAST helps to prevent both excessive data exposure and mass assignment through data flow analysis. The system will highlight many sources of private data, such as those based on variables names or particular API calls, and identify objects that allow mass assignment.

How to Prevent It as a Developer?

Developers should always implement proper controls on the ability to access or change specific object properties. Rather than return a general data structure with every property—which often happens with generic methods, such as to_json() and to_string()—programmers should be very specific in what information they return. As an extra measure of security, applications should implement schema-based response validation that enforces security controls on all data returned by API methods. Access should follow least privilege principles, only allowing access if absolutely necessary.

How Can Fortify Help?

Fortify SAST helps to prevent both excessive data exposure and mass assignment through data flow analysis. The system will highlight many sources of private data, such as those based on variables names or particular API calls, and identify objects that allow mass assignment. Fortify users may define sources of their own as well, tracking data through the program, and if it ends up in an inappropriate place, alerting the developer or operator of the risk.

14. "An incident impacting some accounts and private information on Twitter." Twitter Privacy Center. Twitter. Web Page. 5 Aug 2022. **https://privacy.twitter.com/en/blog/2022/an-issue-affecting-some-anonymous-accounts**.

In addition, Fortify SAST has knowledge of the most important JSON and XML serialization and deserialization mechanisms. Using this, the tool can detect code that does not properly deserialize the domain transfer objects (DTOs), which could allow mass assignment of its attributes. Some cases of information exposure and mass assignment can also be detected using Fortify WebInspect. Finally, some countermeasures can be implemented through adding rules to the web application firewall (WAF).

API4:2023—Unrestricted Resource Consumption

What Is It?

APIs expose many useful business functions. To do so, they use computing resources like database servers or may have access to a physical component through operational technology. Because systems have a finite set of resources to respond to API calls, attackers can specially craft requests to create scenarios that result in resource exhaustion, denial of service, or increased business costs. In many cases, attackers can send API requests that tie up significant resources, overwhelming the machine or bandwidth resources and resulting in a denial-of-service attack. By sending repeated requests from different IP addresses or cloud instances, attackers can bypass defenses designed to detect suspicious spikes in usage.

Applications that do not limit the resources assigned to satisfy a request can be vulnerable, including those that fail to restrict allocable memory, number of files or processes accessed, or the allowed rate of requests, among other attributes.

What Makes an Application Vulnerable?

API requests trigger responses. Whether those responses involve accessing a database, performing I/O, running calculations, or (increasingly) generating the output from a machine-learning model, APIs use computing, network, and memory resources. An attacker can send API requests to an endpoint as part of a denial-of-service (DoS) attack that, rather than overwhelm bandwidth—the goal of a volumetric DoS attack—instead exhaust CPU, memory, and cloud resources. Applications that do not limit the resources assigned to satisfy a request can be vulnerable, including those that fail to restrict allocable memory, number of files or processes accessed, or the allowed rate of requests, among other attributes.

The server processing APIs needs to have limits in place to prevent excessive allocation of memory and workloads, excessive requests for API-triggered operations, or excessive charges for a third-party service without spending limits.

A common attack is to modify the arguments passed to the API endpoint, such as increasing the size of the response and requesting millions of database entries, rather than, say, the first ten:

```
/api/users?page=1&size=1000000
```

In addition, if the attacker can access a backend service that charges for usage, resource consumption attacks can be used to run up charges for the application owner. Another OWASP example points to a reset-password feature that uses an SMS text message to verify identity and which could be called thousands of times to increase expenses for the victim.

```
POST /sms/send_reset_pass_code

Host: willyo.net
{
 "phone_number": "6501113434"
}
```

Attack Examples

Since resource-consumption attacks are often lumped in with performance and availability issues, targeted companies tend to treat them as part of the cost of doing business, rather than incidents that need to be reported, reducing visibility into the threat. In 2022, application-layer distributed-denial-of-service (DDoS) attacks, a superset of API resource consumption attacks, declined as a share of all attacks, but Q4 2022 still logged 79% more attacks than the same quarter the previous year.[15]

In one attack outlined in 2015, a developer detected an Android client that repeated contacted their site's Web API with randomly generated API keys, resulting in a denial-of-service attack. The developer hypothesized that a malicious application installed on Android devices was attempting to guess the 64-bit API key.[16]

Filtering at the edge of the network using content delivery networks (CDNs) paired with web application firewalls (WAFs) can reduce traffic floods while minimizing the impact to individual users.

How to Prevent It as a Developer?

By using rate limits and threshold, most resources consumption attacks can be blunted, although legitimate traffic could also be affected by poorly constructed defenses. Specific limits should be set on:

- Memory allocation
- Processes
- Cloud instances
- Uploaded file descriptors and file size
- Records returned
- Number of paid transactions to third-party services
- All incoming parameters (e.g., string lengths, array lengths, etc.)
- Number of API interactions per client within a specific time window

Filtering at the edge of the network using content delivery networks (CDNs) paired with web application firewalls (WAFs) can reduce traffic floods while minimizing the impact to individual users. Application delivery platforms allow easy filtering, including limits on memory, CPUs, and processes.

How Can Fortify Help?

With Fortify SAST and Fortify WebInspect, DevSecOps teams can test their code and infrastructure for resilience to resource exhaustion attacks. Fortify SAST can spot many areas where an attacker would be able to abuse the application logic to create extreme resource consumption.

15. Yoachimik, Omer. "Cloudflare DDoS threat report for 2022 Q4." Cloudflare Blog. Web Page. 10 Jan 2023. **https://blog.cloudflare.com/ddos-threat-report-2022-q4/**.
16. How to stop hack/DOS attack on web API." StackOverflow. Web Page. 15 Sep 2015. **https://stackoverflow.com/questions/32575924/how-to-stop-hack-dos-attack-on-web-api**.

Code-level security is not sufficient to address this problem in the application. Resource exhaustion and rate limiting are specific sub-segments of denial-of-service attacks that should be mitigated at runtime. Fortify WebInspect can test servers and API functions for vulnerability to denial-of-service attack without impacting the service. In addition, the very act of running a DAST scan can stress test an environment enough to show potential resource-consumption weaknesses.

Fortify WebInspect can test servers and API functions for vulnerability to denial-of-service attack without impacting the service. In addition, the very act of running a DAST scan can stress test an environment enough to show potential resource-consumption weaknesses.

API5:2023—Broken Function Level Authorization

What Is It?

The modern application has many different functions that access, create, manipulate, delete, and manage data. Not every application user needs access to every function or all the data, nor should it be allowed under the principle of least privilege. Every API endpoint has an intended audience which may include anonymous, regular non-privileged, and privileged users. Administrative and management functions should require privileged authorization, but are sometimes accessible through legitimate API calls from non-authorized user—the origin of **Broken Function Level Authorization**. Because of the different hierarchies, groups, and roles create complexity in access controls, applications functions may not have approriate restrictions on who may call them.

What Makes an Application Vulnerable?

Applications that allow specific functions to conduct administrative tasks may not restrict access to those functions in a secure way. APIs that directly map to such functions will expose those weaknesses to exploitation. Functions that do not use the application's authentication and authorization mechanism should be considered potential security weaknesses.

In an example cited by OWASP, an attacker gains access to the API requests for adding an invited user to a new mobile application, noting that the invite includes information on the invitee's role. Exploiting the weakness, the attacker sends a new invite:

```
POST /api/invites/new
{
 "email": "attacker@somehost.com",
 "role":"admin"
}
```

This allows them to gain administrative privileges on the system.

Attack Examples

In 2022, the Texas Department of Insurance notified the public that information of nearly 2 million Texans had been exposed through a part of the workers' compensation application that inadvertently allowed members of the public to access protected data.[17] In a second incident in 2022, Australian telecommunications firm Optus acknowledged that personal and account information on as many as 10 million Australians had been exposed by an API that did not require any authentication or authorization. While Optus called the attack "sophisticated," a security researcher familiar with the details of the attack described it as "trivial."[18]

17. Beeferman, Jason. "Personal information of 1.8 million Texans with Department of Insurance claims was exposed for years, audit says." The Texas Tribune. 17 May 2022. **www.texastribune.org/2022/05/16/texas-insurance-data-breach/**.
18. Taylor, Josh. "Optus data breach: everything we know so far about what happened." The Guardian. 28 Sep 2022. **www.theguardian.com/business/2022/sep/29/optus-data-breach-everything-we-know-so-far-about-what-happened**

How to Prevent It as a Developer?

DevSecOps teams should design a standard approach to authentication and authorization that prevents access to requests by default, enforcing a default of “deny all.” From this default, always apply the principle of least privilege when determining access for roles/groups/users. Developers should ensure that authentication and authorization are in place for all relevant HTTP verbs/methods (e.g., POST, GET, PUT, PATCH, DELETE) related to each API endpoint. Irrelevant verbs should be disallowed. In addition, developers should implement a base class for administrative access and management, using class inheritance to ensure that authorization controls check the user’s role before granting access. All critical administrative functions should use the authorization mechanism to prevent privilege escalation.

DevSecOps teams should design a standard approach to authorization and authentication that prevents access to requests by default, enforcing a default of “deny all.”

How Can Fortify Help?

By combining the static code and API analysis features of Fortify SAST with the runtime checks of the Fortify WebInspect dynamic application security testing (DAST) suite, DevSecOps teams can evaluate their application for broken function-level authorization issues and continuously test production code for security weaknesses before deploying. To detect Broken Object Function Authorization issues, Fortify SAST uses rules specifying when an authorization check would be expected in certain programming languages and frameworks, and the absence of such a check is reported.

API6:2023—Unrestricted Access to Sensitive Business Flows

What Is It?

From sneakerbots to ticket bots, attacks on the inventory of online retailers through their APIs has become a significant problem for e-commerce sites. By understanding the business model and the application logic, an attacker can create a series of API calls that can automatically reserve or purchase inventory, thus preventing other, legitimate consumers from gaining access to the businesses’ products or services. Any API that allows access to a business process can be used by an attacker to impact the business and falls under the definition of **Unrestricted Access to Sensitive Business Flows.**

What Makes an Application Vulnerable?

Application control and logic flows are the heart of any online businesses, and as companies move more of their operations to the cloud, those flows can be exposed and exploited. This excessive access may harm the business, when attackers automate the purchase of products, create bots for leaving comments and reviews, or automate the reservation of goods or services.

Application control and logic flows are the heart of any online businesses, and as companies move more of their operations to the cloud, those flows can be exposed and exploited. This excessive access may harm the business.

If an application offers an endpoint that has access to the company’s business flow without limiting access to the business operations behind the endpoint, then the application will be vulnerable. Protections include limiting the number of access attempts from a single device through fingerprinting, detecting whether the activity originates from a human actor, and detecting whether automation is involved.

Attack Examples

When Taylor Swift tickets went on sale on Ticketmaster in November 2022, 1.5 million customers had pre-registered, but more than 14 million requests—including three times as much bot traffic—swamped the purchasing links and APIs as soon as ticket sales opened. The site crashed, preventing many customers from purchasing tickets.[19]

The onslaught of reseller bots resembled those that ruined the launch of the PlayStation 5 in November 2020. Supply-chain issues had already limited supply prior to the launch of the latest Sony gaming console, but the automated bots made finding available units even harder and led to astronomical resale prices. In one e-commerce site's case, the number of "add to cart" transactions grew from an average of 15,000 requests per hour to more than 27 million, using the store's API to directly request products by SKU number.[20]

Preventing Unrestricted Access to Sensitive Business Flows is more about a holistic approach to application security and less about finding a specific technology.

How to Prevent It as a Developer?

Developers should work with both the business-operation and engineering teams to address issues of potential malicious access to business-flows. Business teams can identify which flows are exposed through APIs and conduct threat analyses to determine how attackers could abuse those endpoints. Meanwhile, developers should work with engineering operations as part of a DevOps team to establish additional technical defensive measures, such as using device fingerprinting to prevent automated browser instances from overwhelming and identifying patterns in behavior that differentiate between human and machine actors.

Operations teams should also review any APIs designed to be used by other machines, such as for B2B use cases, and ensure that some defenses are in place to prevent attackers from exploiting machine-to-machine interactions.

How Can Fortify Help?

Catching vulnerable and sensitive business flows often relies on doing the basics. Companies need to document and track all of their functioning APIs and determine which ones expose sensitive processes and data to potential attackers. Application logic also needs to be analyzed for logic flaws that could be exploited by attackers.

Overall, preventing Unrestricted Access to Sensitive Business Flows is more about a holistic approach to application security and less about finding a specific technology.

API7:2023—Server Side Request Forgery

What Is It?

Backend servers handle requests made through API endpoints. **Server-Side Request Forgery (SSRF)** is a vulnerability that allows an attacker to induce a server to send requests on their behalf and with the server's level of privilege. Often the attack uses the server to bridge the gap between the external attacker and the internal network. Basic SSRF attacks result in a response returned to the attacker, a far easier scenario than Blind SSRF attacks, where no response is returned, leaving the attacker with no confirmation whether the attack was successful.

19. Steele, Billy. "Ticketmaster knows it has a bot problem, but it wants Congress to fix it." Engadget. News Article. 24 Jan 2023. **www.engadget.com/ticketmaster-live-nation-senate-judiciary-hearing-195504179.html**.
20. Muwandi, Tafara and Warburton, David. "How Bots Ruined the PlayStation 5 Launch for Millions of Gamers." F5 Labs Blog. F5. Web Page. 18 March 2023. **www.f5.com/labs/articles/cisotociso/how-bots-ruined-the-playstation-5-launch-for-millions-of-gamers**.

What Makes an Application Vulnerable?

Server-Side Request Forgery (SSRF) flaws essentially are a result of a lack of validation of user-supplied input. Attackers are able to craft requests and include a URI that supplies access to the targeted application.

Modern concepts in application development, such as webhooks and standardized application frameworks, make SSRF more common and more dangerous, according to OWASP.

In an example cited by OWASP, a social network that allows users to upload profile pictures could be vulnerable to SSRF, if the server does not validate arguments sent to the application. Rather than a URL pointing to an image, such as:

```
POST /api/profile/upload_picture
{
 "picture_url": "http://example.com/profile_pic.jpg"
}
```

An attacker could send a URI that could determine whether a specific port is open using the following API call:

```
{
 "picture_url": "localhost:8080"
}
```

Even in a Blind SSRF case, an attacker could figure out whether the port is open by measuring the time it take to get a response.

The most well-known example of an SSRF attack involved a former Amazon Web Services (AWS) engineer who exploited a misconfigured web application firewall (WAF) to then use an SSRF flaw to gather data from a server instance belonging to financial giant Capital One.

Attack Examples

The most well-known example of an SSRF attack involved a former Amazon Web Services (AWS) engineer who exploited a misconfigured web application firewall (WAF) to then use an SSRF flaw to gather data from a server instance belonging to financial giant Capital One. The incident, which occurred in July 2019, resulted in data from approximately 100 million US citizens and 6 million Canadian citizens being stolen.[21] Amazon considers the misconfiguration to be the source of the compromise, rather than the SSRF flaw.[22]

In October 2022, a cloud security firm notified Microsoft of four SSRF vulnerabilities in the company's flagship Azure cloud platform. Each vulnerability affected a different Azure service, including the Azure Machine Learning service and the Azure API Management service.[23]

How to Prevent It as a Developer?

Developers should encapsulate the resource-fetching mechanisms in their code, isolating the feature and layering addition protections to verify any requests. Because such features are typically used to fetch remote resources and not internal ones, developers should configure the encapsulated features to use a list of allowed remote resources and block attempts to access internal resources. HTTP redirection should be disabled for the resource-fetching functions and any requests parsed for malicious code.

21. "Information on the Capital One cyber incident." Capitol One Advisory. Web Page. Updated 22 Apr 2022. **www.capitalone.com/digital/facts2019/**.
22. Ng, Alfred. "Amazon tells senators it isn't to blame for Capital One breach. CNET News.com. News article. 21 Nov 2019. **www.cnet.com/tech/services-and-software/use-cnet-shopping-to-seek-out-the-best-deals/**.
23. Shitrit, Lidor Ben. "How Orca Found Server-Side Request Forgery (SSRF) Vulnerabilities in Four Different Azure Services." Orca Security Blog. Web Page. 17 Jan 2023. **https://orca.security/resources/blog/ssrf-vulnerabilities-in-four-azure-services/**.**www.upguard.com/breaches/power-apps**.

The risk of SSRF weaknesses cannot always be completely eliminated, so companies should closely considered the risk of using calls to external resources.

How Can Fortify Help?
Fortify WebInspect allows DevSecOps teams to regularly test for server-side request forgery. WebInspect's DAST scans an application server in a configured environment so that all components—application, server, and network—can be tested, giving the dynamic analysis platform a comprehensive view of the impact of server requests.

Fortify SAST can detect many cases of SSRF through taint analysis—for example, if the application uses unvalidated user input to construct a URL that will then be fetched. Fortify will flag the use of unrestricted user input.

API8:2023—Security Misconfiguration

What Is It?
Developers often misconfigure their applications, failing to separate development assets from production assets, exporting sensitive files—such configuration files—to their public repositories, and failing to change default configurations. **Security Misconfiguration** includes setting up applications with vulnerable default configurations, allowing overly permissive access to sensitive functions and data, and publicly revealing application information through detailed error messages.

Security Misconfiguration includes setting up applications with vulnerable default configurations, allowing overly permissive access to sensitive functions and data, and publicly revealing application information through detailed error messages.

What Makes an Application Vulnerable?
Default application configurations are often overly permissive, lacking security hardening, and leaving cloud storage instances open to the public. Often, the web frameworks on which applications are based include a host of application features that are not needed and whose inclusion reduces security.

In an example detailed by OWASP, a social network that offers a direct-messaging feature should protect users' privacy, but offers an API request to retrieve a specific conversation using the following example API request:

```
GET /dm/user_updates.json?conversation_id=1234567&cursor=GRlFp7LCUAAAA
```

The API endpoint does not restrict the data stored in the cache, resulting in private conversations being cached by the web browser. Attackers could retrieve the information from the browser, exposing the victim's private messages.

Attack Examples
In May 2021, a cloud security firm notified Microsoft that at least 47 different customers had failed to change the default configuration of their instances of Microsoft Power Apps. The affected organizations included companies, such as American Airlines and Microsoft, and state government, such as those of Indiana and Maryland, and exposed 38 million records to potential compromise across the Power Apps portals.[24]

24. Upguard Research. "By Design: How Default Permissions on Microsoft Power Apps Exposed Millions." Upgard Research Blog. Web Page. 23 Aug 2021. **www.upguard.com/breaches/power-apps**.

In 2022, a vulnerability management firm discovered that 12,000 cloud instances hosted on Amazon Web Services and 10,500 hosted on Azure continued to expose Telnet, a remote access protocol considered "inappropriate for any internet-based usage today," according to a 2022 report.[25] The inclusion of unnecessary and insecure features undermines these security of the APIs and applications.

How to Prevent It as a Developer?

DevSecOps teams need to understand the steps necessary to create secure configurations for their applications and use an automated development pipeline to check configuration files before deployment, including regular unit tests and runtime checks to continuously check the software for configuration errors or security problems. Security-as-code can help, by making configurations repeatable and giving application-security teams the ability to set standard configuration sets for specific application components.

Security-as-code can help, by making configurations repeatable and giving application-security teams the ability to set standard configuration sets for specific application components.

As part of their secure development lifecycle, developers and operations teams should:

- Establish a hardening process that simplifies the repeatable creation and maintainance of a secure application environment,
- Review and update all configurations across the API stack to incorporate the new standard consistently, and
- Automate the assessment of the effectiveness of the configuration settings across all environments.

How Can Fortify Help?

Fortify SAST can check configurations during the development process and spot many types of weaknesses. Because Security Misconfigurations occur at both the application-code level and at the infrastructure level, different Fortify products can be used to catch misconfigurations.

Fortify SAST scans can check application code for misconfiguration issues. During the static analysis check, Fortify SAST can evaluate configuration files for security errors, including those for Docker, Kubernetes, Ansible, Amazon Web Services, CloudFormation, Terraform, and Azure Resource Manager templates.

Configuration errors can also be caught during runtime. Fortify WebInspect allows DevSecOps teams to regularly test for common security misconfigurations. One of the biggest strengths of DAST scanning is that it runs on the application server in a configured environment, which means that the full environment—application, server, and network—are tested all at once, giving the dynamic analysis platform a comprehensive view of the production environment is configured.

25. Beardsley, Todd. "2022 Cloud Misconfigurations Report." Rapid7. PDF Report. p. 12. 20 Apr 2022. Accessed through: **www.rapid7.com/blog/post/2022/04/20/2022-cloud-misconfigurations-report-a-quick-look-at-the-latest-cloud-security-breaches-and-attack-trends/**.

API9:2023—Improper Inventory Management

What Is It?

Like most software assets, APIs have a lifecycle, with older versions replaced by more secure and efficient APIs or, increasingly, using API connected to third-party services. DevSecOps teams who do not maintain their API versions and documentation can introduce vulnerabilities when older, flawed API versions continue to be used—a weakness known as **Improper Inventory Management**. Best practices for inventory management require the tracking of API versions, the regular assessment and inventorying of integrated services, and the regular deprecation of legacy versions to prevent the propagation of security vulnerabilities.

A **documentation blindspot** is when the details of the API's purpose, functioning, and versioning are unclear because of a lack of documentation detailing these important attributes.

What Makes an Application Vulnerable?

Software architectures reliant on APIs—especially those using microservice architectures—tend to expose more endpoints than traditional web applications. The plethora of API endpoints, along with the likelihood of multiple versions of an API existing at the same times, requires additional management resources from the API provider to prevent an expanding attack surface. OWASP identifies two major blindspots that DevSecOps teams may have regarding their API infrastructure.

First, a **documentation blindspot** is when the details of the API's purpose, functioning, and versioning are unclear because of a lack of documentation detailing these important attributes.

Second, a **data-flow blindspot** happens when APIs are used in ways that lack clarity, resulting in capabilities that should not necessarily be allowed without a strong business justification. Sharing sensitive data with a third party without security guarantees, lacking visibility of the end result of a data flow, and failing to map all data flows in chained APIs are all blindspots.

As an example, the OWASP report cites a fictional social network that allows integration with third-party independent applications. While consent is required from the end user, the social network does not maintain enough visibility into the data flow to prevent downstream parties from accessing the data, such as monitoring the activity of not just the user, but their friends.

Attack Examples

In 2013 and 2014, as many as 300,000 people took an online psychological quiz on the Facebook platform. The company behind the quiz, Cambridge Analytica, not only collected information on those users, but their linked friends as well—a population that totaled as many as 87 million people, the vast majority of whom gave no permission to have their information collected. The company then used the information to tailor ads and messaging to those people on behalf of their clients, including sending political ads supporting the Trump campaign in the 2016 election.[26] Facebook's lack of visibility into how third parties used the information harvested from its platform is an example of improper inventory management.

26. Rosenberg, Matthew and Dance, Gabriel. "'You Are the Product': Targeted by Cambridge Analytica on Facebook." The New York Times. News article. 8 April 2018. **www.nytimes.com/2018/04/08/us/facebook-users-data-harvested-cambridge-analytica.html**.

How to Prevent It as a Developer?

DevSecOps teams should document all API hosts and focus on maintaining visibility into the data flows between APIs and third-party services. The primary way to prevent Improper Inventory Management is the detailed documentation of the critical aspects of all API services and hosts, including information on what data they handle, who has access to the hosts and data, and the specific API versions of each host. Technical details that should be documented include the authentication implementation, error handling, rate limiting defenses, the cross-origin resource sharing (CORS) policy, and details of each endpoint.

The significant volume of documentation is difficult to manage manually, so generating documentation through the continuous integration process and using open standards is recommended. Access to API documentation should also be limited to those developers who are authorized to use the API.

During the application building and testing phases, developers should avoid using production data on development or staged versions of the application to prevent data leaks. When new versions of APIs are released, the DevSecOps team should do a risk analysis to determine the best approach to upgrading applications to take advantage of increased security.

How Can Fortify Help?

Organizations can manage, monitor, secure, and document their API usage using the NetIQ Secure API Manager by OpenText, which allows application-security teams to maintain an up-to-date inventory of API assets. The Secure API Manager provides an authoritative repository where your DevSecOps team can store and manage all of the APIs used by the organization, allowing an easy-to-manage life cycle from API development to retirement. The software helps improve compliance with regulations and licensing by allowing detailed analytics.

Organizations can manage, monitor, secure, and document their API usage using the NetIQ Secure API Manager by OpenText, which allows application-security teams to maintain an up-to-date inventory of API assets.

API10:2023—Unsafe Consumption of APIs

What Is It?

With the increasing use of native cloud infrastructure to create applications, APIs have become the point of integration between application components. However, the security posture of third-party services accessed through APIs is rarely clear, allowing attackers to determine on which services an application relies and whether any of those services have security weaknesses. Developers tend to trust the endpoints that their application interacts without verifying the external or third-party APIs. This **Unsafe Consumption of APIs** often leads to the application's reliance on services that have weaker security requirements or lack fundamental security hardening, such as input validation.

What Makes an Application Vulnerable?

Developers tend to trust data received from third-party APIs more than user input, although the two sources are essentially equivalent for a motivated attacker. Because of this misplaced trust, developers essentially end up relying on weaker security standards due to a lack of input validation and sanitization.

Unsafe consumption of APIs may occur if the application:

- Uses or consumes other APIs using unencrypted communications,
- Fails to validate and sanitize data from other APIs or services,
- Allows redirection without any security checks, or
- Fails to limit resource consumption using thresholds and timeouts.

In an example from the OWASP report, an API that integrates with a third-party service provider to store sensitive user medical information might send private data through an API endpoint. Attackers could compromise the third-party API host to respond to future requests with a 308 Permanent Redirect:

HTTP/1.1 308 Permanent Redirect
Location: **https://attacker.com/**

If the developer does not code security checks into their application to verify any data returned by the API endpoint, their application will follow the redirect and send sensitive medical information to the attacker.

If the developer does not code security checks into their application to verify any data returned by the API endpoint, their application will follow the redirect and send sensitive medical information to the attacker.

Attack Examples

In December 2021, a set of vulnerabilities in a commonly used open-source software component, Log4J, allowed an attacker to provide unsanitized input, such as an encoded script, and use vulnerable versions of Log4J to execute the script on the server. The issue behind the Log4J vulnerability originated in a lack of input validation, specifically the failure to conduct security checks on deserialized user-supplied data. By sending serialized malicious code, attackers could exploit the vulnerability and execute an attack on a server with the vulnerability. Developers should check all input provided by third-party APIs and other external sources.[27]

How to Prevent It as a Developer?

Developers should conduct due diligence when evaluating service providers, assessing their API security posture and implementing strict security controls. In addition, developers should confirm that all communications to third-party APIs and from third parties to the organization's APIs use a secure communication channel to prevent snooping and replay attacks.

When receiving data from external users and machines, the inputs should always be sanitized to prevent the inadvertent execution of code. Finally, for cloud services integrated through APIs, allow lists should be used to lock the address of the integrated solution, rather than blindly allowing any IP address to call the application's API.

How Can Fortify Help?

By combining the static code and API analysis features of Fortify SAST with the runtime checks of the Fortify WebInspect dynamic application security testing (DAST) suite, DevSecOps teams can check their application's use of third-party APIs and test common attack types. To find unsafe APIs, Fortify's Secure API Manager can build a repository of all APIs called by the system as well as which external applications can use your application's APIs.

27. Microsoft Threat Intelligence. "Guidance for preventing, detecting, and hunting for exploitation of the Log4j 2 vulnerability." Microsoft. Web page. Updated: 10 January 2022. **www.microsoft.com/en-us/security/blog/2021/12/11/guidance-for-preventing-detecting-and-hunting-for-cve-2021-44228-log4j-2-exploitation/**.

The API Security Top-10 Is Not Sufficient!

For cloud-native developers specifically focused on creating APIs to offer services to other parts of an application, internal users, or for global consumption, the OWASP API Security Top 10 list is an important document to read and understand.

However, the OWASP API Security Top 10 is not a standalone document. Developers also need to make sure that they utilize other sources of best practices, such as the OWASP Top 10, that are relevant to their current application and architecture. Common application vulnerabilities -SQL injection, data exposure, and security misconfiguration- continue to be common ways that cyber threat groups can compromise corporate infrastructure and should be remediated quickly. In addition, some API-based applications, such as mobile apps, require different appsec hardening steps than a stand-alone web-app, and different from what may be required for connect and IoT devices. Overall, the API Security Top 10 list is important, but it remains only a facet of the overall secure software development lifecycle. The list, and the OWASP Top 10 list, should be used in conjunction with any other relevant standards and best practices that are required for the solution under analysis.

The OWASP API Security Top-10 is crucial for cloud-native developers building APIs. Yet, addressing common application vulnerabilities like SQL injection, data exposure, and security misconfiguration should take priority, as they are frequently exploited by cyber threats. The API Security Top-10 is an essential part of secure software development but should be secondary to addressing general application vulnerabilities.

Conclusion

As applications increasingly rely on cloud infrastructure, web application programming interfaces (APIs) have become the foundation of the Internet. Companies typically have hundreds, if not thousands, of API endpoints in their environment, dramatically increasing their attack surface and exposing applications to a variety of weaknesses.

The release of the 2023 OWASP API Security Top 10 list is a good starting point for companies and developers to educate themselves on the risks of API-based infrastructure and to assess their own applications. Along with the more well-known Application Security Top-10 list, the pair of rankings can help DevSecOps teams toward developing a holistic approach to the overall security of their applications.

DevSecOps teams need to be aware of the security implications of APIs, how to reduce an implementation's vulnerabilities and security weaknesses, and how to harden their development pipeline and the resulting API server to make it more difficult for attackers to compromise an application through its APIs.

Where to Go Next

Here are the products mentioned in this document:

- Fortify API Security
- Fortify Static Code Analyzer (SAST)
- Fortify WebInspect (DAST)
- NetIQ Secure API Manager

Additional Resources

- OWASP Top 10 API Security Risks—2023
- Gartner Magic Quadrant fo Application Security Testing
- Fortify Code Security Webinar Series
- Fortify Application Security

Connect with Us

www.opentext.com

opentext™ | Cybersecurity

762-000081-004 | O | 11/23 | © 2023 Open Text