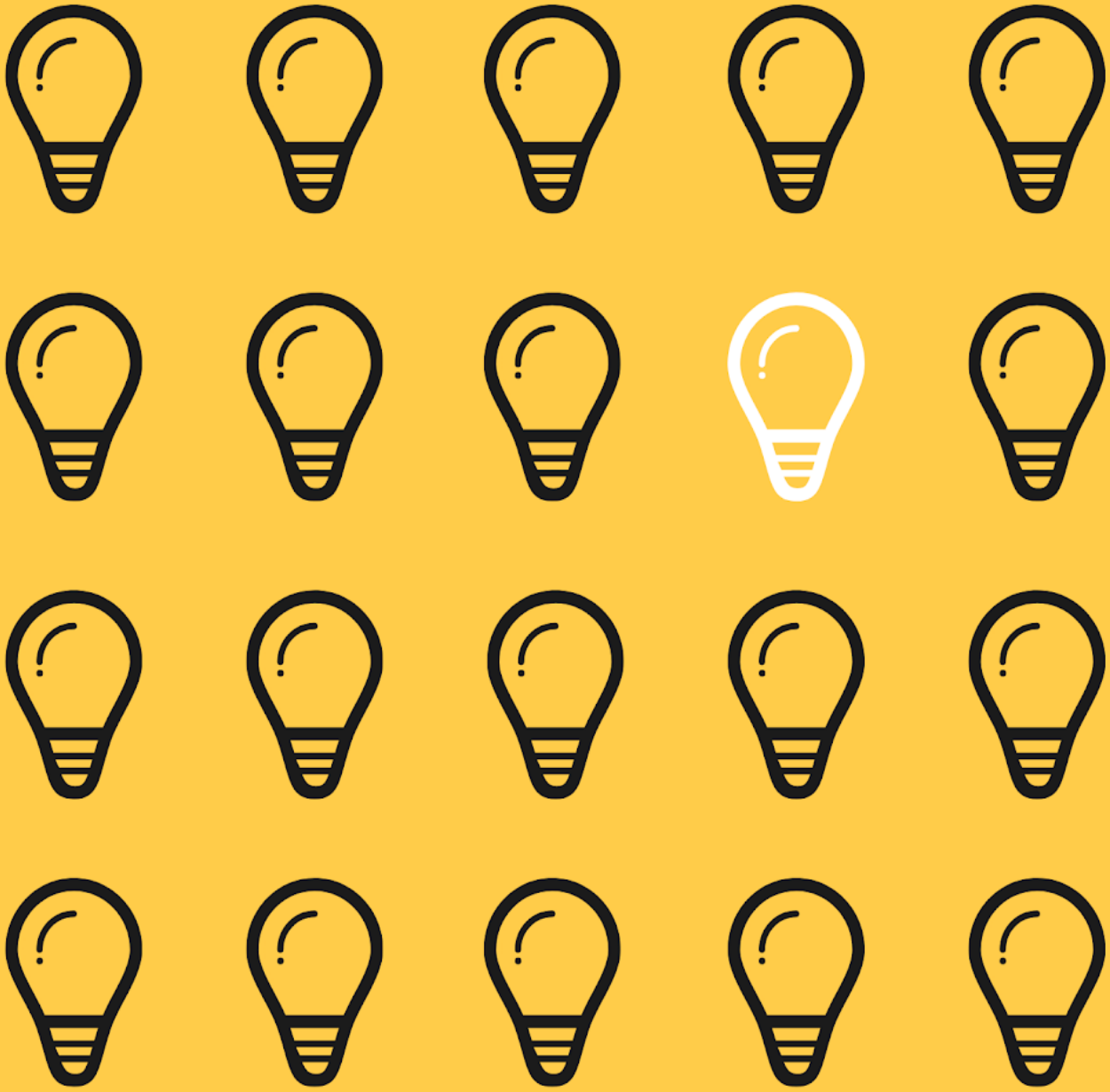

• API SECURITY •

#1 API VULNERABILITY

{ API SECURITY MANAGEEMENT }

APISec

03/31/2020

FX Labs, Inc.
API Security Management
https://cybersecuriti.ai

Important Stats

1. 83% of web traffic stems API calls - Akamai[1]
2. 40% attack surface is exposed by API-enabled Web & Mobile apps - Gartner[2]
3. $3.92m average cost of a breach - SecurityIntelligence[3]

What is OWASP

OWASP[4] (Open Web Application Security Project) is a non-profit and vendor-neutral organization. OWASP publishes AppSec Top 10 critical security risks based on the broad industry consensus.

At the moment, this research is hard to back by data as most organizations still don't reveal the breaches, this may change with new laws like GDPR & CCPA, which requires that organizations report accidental data exposures and breaches.

OWASP API Security Top 10 Official Definition[5]:

Skip this section if you already know it.

Rank	Name	Comments

[1] "State of the Internet Security Reports | 2019 | Akamai." https://www.akamai.com/us/en/resources/our-thinking/state-of-the-internet-report/archives/state-of-the-internet-security-reports-2019.jsp. Accessed 31 Mar. 2020.
[2] "API Security: What You Need to Do to Protect Your APIs." https://www.gartner.com/en/documents/3956746/api-security-what-you-need-to-do-to-protect-your-apis. Accessed 31 Mar. 2020.
[3] "What's New in the 2019 Cost of a Data Breach Report." 23 Jul. 2019, https://securityintelligence.com/posts/whats-new-in-the-2019-cost-of-a-data-breach-report/. Accessed 31 Mar. 2020.
[4] "owasp." https://owasp.org/. Accessed 31 Mar. 2020.
[5] "OWASP API Security Project." https://owasp.org/www-project-api-security/. Accessed 31 Mar. 2020.

API1	Broken Object Level Authorization	APIs tend to expose endpoints that handle object identifiers, creating a wide attack surface Level Access Control issue. Object-level authorization checks should be considered in every function that accesses a data source using input from the user.
API2	Broken Authentication	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.
API3	Excessive Data Exposure	Looking forward to generic implementations, developers tend to expose all object properties without considering their individual sensitivity, relying on clients to perform the data filtering before displaying it to the user.
API4	Lack of Resources & Rate Limiting	Quite often, APIs do not impose any restrictions on the size or number of resources that can be requested by the client/user. Not only can this impact the API server performance, leading to Denial of Service (DoS), but also leaves the door open to authentication flaws such as brute force.
API5	Broken Function Level Authorization	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.
API6	Mass Assignment	Binding client provided data (e.g., JSON) to data models, without proper properties filtering based on a whitelist, usually lead to Mass Assignment. Either guessing objects properties, exploring other API endpoints, reading the documentation, or providing additional object properties in request payloads, allows attackers to modify object properties they are not supposed to.
API7	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.

API8	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.
API9	Improper Assets Management	APIs tend to expose more endpoints than traditional web applications, making proper and updated documentation highly important. Proper hosts and deployed API versions inventory also play an important role to mitigate issues such as deprecated API versions and exposed debug endpoints.
API10	Insufficient Logging & Monitoring	Insufficient 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.

API1 - Broken Object Level Authorization

Also known as IDOR (Insecure Direct Object Reference)

The fundamental concept in API is the resource or object. A resource has a set of attributes and is identified by an Id and may have a set of methods/operations/endpoints that operate on it.

Let's walk through a sample scenario to understand how common this vulnerability is and how it's exploited. A Bitcoin Exchange application may have the following resources Users, Accounts, etc.

A resource is identified by the resource id: /users/{id}, /accounts/{id}.

A typical resource may have several methods to operate on it, for example:

HTTP Method	API Endpoint	Comment
POST	/accounts	Registers an account with the exchange
GET	/accounts/{id}	Returns account information

POST	/accounts/{id}/order	Process order
DELETE	/accounts/{id}	Deregisters account with exchange

All Bitcoin exchange customers go through the same workflow. When a customer registers an account, the exchange stores the information in the system and assigns an identifier to the resource and entitles the user to operate the account. The exchange manages millions of accounts identified by an Id for individual customers.

Broken Object Level Authorization vulnerability means a resource (account) belonging to Customer-A(user) is operable by another customer Customer-B.

For example
Customer-A registers an Account, and the system assigns ‘AC-123’ as the account ID. Customer-A is entitled to the resource ‘AC-123’ and should be able to operate on it.

If a customer Customer-B get’s hold of the ID ‘AC-123’. Customer-B an can invoke all operations on the accounts endpoints by passing the unauthorized ID. The system should have enough access-control policies to defend against every invalid request.

Customer	API Endpoint		Response	Comment
Customer-A	POST /accounts	{}	200 OK {“id”: “AC-123”}	Customer-A registers an account and its id is returned
Customer-A	GET /accounts/AC-123		200 OK Data	Customer-A can fetch resource AC-123
Customer-A	POST /accounts/AC-123/order	{}	200 OK	Customer-A can place an order on resource AC-123
Customer-A	DELETE /accounts/AC-123		200 OK	Customer-A can delete resource

				AC-123
Customer-B	GET /accounts/AC-123		401 Unauthorized	Customer-B cannot fetch resource AC-123
Customer-B	POST /accounts/AC-123/order	{}	200 OK	Customer-B can place an order on resource AC-123
Customer-B	DELETE /accounts/AC-123		200 OK	Customer-B can delete resource AC-123

How to interpret the severity of this vulnerability?

Customer-B is able to successfully invoke order and delete operations on Customer-A's resource/data. This can lead towards a few scenarios. Any other customer in the system or a hacker who registers with your service or steals credentials have access to placing orders and deleting every account in the system i.e. all orders and vulnerables and all accounts can be automatically deregistered.

What can cause this problem ?

1. **Using numbers as resource ids**
    a. A typical design time mistake is to use a number generator for resource Ids. This approach makes it much easier for hackers and other users alike to guess resource Ids. If other systems consume the API data, it becomes harder to go back and refactor the ID to a strong non-guessable GUID, etc.

2. **Adding new resource types**
    a. As new developers add features to existing resources like introducing a new account type called hold-wallet. The new feature requires similar hardening across access-control policies. If not done right, it can introduce a regression in the old feature or ship the new feature with missing controls.

3. **Supporting complex entitlements**
    a. Most resources require a very complex entitlement model. A resource can be entitled to multiple users with varying levels of access-control; this makes the problem even harder to test and validate.

4. **Relying on GUID**
    a. Relying on GUID based IDs to protect authorized access to customer's business critical data. GUID IDs are not a security measure, they make guessing hard. But most GUID IDs can be easily reproduced. Most of the time these GUID IDs are stored as logs in mobile, web, production and third party

integrated systems. Sometimes these IDs are also sent as part of links in emails, reports, etc.. Hence relying on these IDs makes no sense.

Why is this vulnerability rated as #1 in OWASP?

- **Easy to discover**
  - Most likely hackers sign-up or steal two accounts. Using one account they create a resource. The other account is used for operating on the resource created by the first account. Once a vulnerability is discovered they most like automate the entire process.

- **Most targeted verticals**
  - B2C - Easy to register new accounts. No need to steal or phish credentials
    - ECommerce - Access to PII and financial information
    - Fintech -
    - Technology Startups - Most startups cannot effort AppSec teams until they grow significantly.
    - Event Management
  - B2B - Hard to get access requires phishing credentials or brute forcing default credentials.
    - Databases like ElasticSearch, PostgreSQL, MySQL, MongoDB exposed to internet with no-credentials or default credentials
    - On-premises, cloud, managed products. Most organizations stay on older versions for very long periods. Once a vulnerability is leaked the hackers get ample of time to discover these instances and exploit them.

- **Most attacked vector**

  Many large and small organizations alike were breached or have leaked data when it comes to API related incidents. Anytime an organization blames the issue on a code related bug, mostly likely it's because of this problem. A few references are mentioned below.

- **Large attack surface**

  Access-control issues offer a large surface area for hackers. A typical 100 endpoint API may at least expose 500 unique attack points.

- **Hard to detect & fix**

  A small application requires close to 500 hours of manual testing or 2000 hours of building automated testing. Most organizations lack tactical SecOps/DevSecOps teams. And a few who have these teams still cannot effort 500 hours of manual testing or 2000 hours of automated coverage per Application. The result is large numbers of APIs have these critical vulnerabilities.

Is in-house manual testing the right strategy?

It's tough to test APIs manually. Typical manual testing still involves CURL or Postman approach. The building, testing, and maintaining these tests can easily overwhelm a small team.

Testing cost?

2000 hours for building automation for a small API is roughly equivalent to **30%** of development cost. Automated testing requires continuous maintenance to adopt to changing APIs and scale execution to support per commit valiations

Can WAF (Web-Application Firewall) cover these flaws?

No, WAFs are excellent for filtering & blocked IP-addresses, SQL, & XSS attacks. But they cannot protect against data-access vulnerabilities because of the lack of contextual knowledge.

Can WAS (Web Application Scanner) detect these flaws?

No, WAS are good against detecting SQL & XSS vulnerability in web applications. A few WAS now covering the SQL/XSS coverage to APIs. But again, these solutions don't cover IDOR/access-control/business-logic flaws.

Is Penetration Testing the right approach?

Periodic penetration testing doesn't cover all the scenarios and requires intimate product knowledge before it can deliver quality results. Most penetration testing approaches only focus on detection, and they do very little in terms of validation and regression testing.

Can Static Code Analysis cover these flaws?

No, these issues are hard to surface in static code analysis. Most resource entitlements are complex and sprinkled across thousands of source files, and they keep changing as new features are added.

Who should drive this initiative?

AppSec (DevSecOps) should own and drive these initiatives.

When should this be done?

Should be applied early in the development cycle. Detecting late can cost 20x more.

What is the most targeted data?

- PII

- Financial
- Business Critical Data

How does the friendly data leak happen?

The customer created data through broken access in search and data listing features.

Cost of Breach/Incident (~ $4M)

CCPA and GDPR privacy laws require organizations to report data exposure incidents to these governing bodies. Also requires the organizations report details about stolen and accessed data to affected users and customers.

Businesses spend a large amount of money on performing forensics on identifying stolen and authorized accessed data.

Businesses are required to disclose security policies and practices. Non-standard practices are penalized additionally by the governing bodies in the respective regions.

B2B companies are financially responsible for diagnosing and mitigating risk for their business customers.

Most breaches attract a lot of legal disputes jumping the cost even higher.
If the business that has leaked sensitive customer information to third parties via the APIs. They're now required to work with these integrators to identify and scrape data in their systems adding more financial and resource burden to an already overwhelmed team.

Recommended Strategy:

Instant and automatic coverage for all business-logic use-cases at the speed of development. Data-driven validation for visibility and customizability. Continuous execution of tests against every commit and nightly builds to detect and fix issues early in the development cycle. APISec is based on these principles allowing small AppSec teams to support large development teams and a large set of APIs simultaneously

Conclusion

- Broken Object Level Authorization exposes a large attack surface
- Hard to test using manual audits and in-house automation
- Bad design can cause problem even worse
- New features can break and introduce new access-control issues
- Missing controls can leak customer critical information to other tenants

- Automated and continuous approach to coverage creation and execution
- B2B breaches can be most expensive most startups fails within 6 months

IDOR based breaches

- https://krebsonsecurity.com/2019/05/first-american-financial-corp-leaked-hundreds-of-millions-of-title-insurance-records/
- https://www.zdnet.com/article/shopify-api-flaw-offered-access-to-revenue-traffic-data-of-thousands-of-stores/

References:

4. https://owasp.org/www-project-api-security/
5. https://www.akamai.com/us/en/about/news/press/2019-press/state-of-the-internet-security-retail-attacks-and-api-traffic.jsp
6. https://www.gartner.com/doc/reprints?id=1-1OH7K9NT&ct=190910&st=sb
7. https://securityintelligence.com/posts/whats-new-in-the-2019-cost-of-a-data-breach-report/
8. https://community.apigee.com/questions/78098/owasp-api-security-top-10-circumventing-broken-obj.html