What is the OWASP ?

- ➢ The Open Web Application Security Project or OWASP
- ➢ OWASP is a nonprofit foundation that was created to help improve application security.
- ➢ The OWASP API Security Top 10 is a list of the most critical security risks for application programming interfaces.

CLASSİC CYBER ATTACK

1. Reconnaissance
2. Weaponize
3. Infiltration
4. Lateral Movement
5. Privilege Escalation
6. BREACH

API ATTACK

1. Find Vulnerability
2. BREACH

API1:2023 Broken Object Level Authorizaiton(BOLA)

One of the most prevelent and severe vulnerabilities for APIs. BOLA vulnerabilities occur when an API provider does sufficient controls in place to enforce authorization.

- In other words, API users should only have access to sensitive resources that belong to them.
- When BOLA is present, an attacker will be the access to sensitive data of other users.

Attack Vector Description :

*"*Attacker can exploit API endpoints that are vulnerable to a broken level object authorization by manipulating the ID of an object that is sent within the request Object IDs can be anything from sequential integers, UUIDs or generic strings. Regardless of the data type, they're easy to identify in the request target , either through the path or a query string parameter request headers ,or even as a part of the request payload."

Security Weakness Description :
“This has been the most common and impactful attack on APIs. Authorization and access control mechanism in modern web applications are complex and widespread. Even if the application implements a proper infrastructure for Authorization checks, developer might forget the use these checks before accessing a sensitive object. Access controlled detection is not typically amenable to automated static or dynamic testing”

Impacts Description :
“Unauthorized access can result in data disclosure to unauthorized parties, data loss , or data manipulation. Unauthorized access to objects can also lead to full account takeover”

BOLA EXAMPLE

ACCESSİNG RESOURCES

OWASP PREVENTATİVE MEASURES

- Implement authorization with user policies and hierarchy
- Verify users have access to the resources they're accessing
- Use random object IDs wherever possible
- Test APIs to identify any authorization vulnerabilities

API2:2023 Broken Authentication

Authentication is the process that is used to verify the identify of an API users. In other words, authentication is the process of verifying that an entity is who that entity claims to be.
This verification process is normally done with the use of a username and password combination, a token and or multifactor authentication.

Authentication related vulnerabilities typically occur when an API provider either doesn't implement a strong authentication mechanism or implements an authentication process incorrectly.

Attack Vector Description :
" The authentication mechanism is an easy target for attackers since it's exposed to everyone. although more advanced technical skills maybe required to exploit some authentication issues, exploitation tools are generally available."

Impact Description :
" Attackers can gain complete control of other users' accounts in the system, read their personal data, and perform sensitive actions on their behalf. System are unlikely to be able to distinguish attackers' actions from legitimate users ones".

Weak Password Policy

- ➔ Allow users to create simple passwords
- ➔ Allows brute force attempts against user accounts
- ➔ Allows users to change their passwords without asking for password confirmation
- ➔ Allows users to change their account email without asking for password confirmation
- ➔ Discloses token or password in the URL
- ➔ GraphQL queries allow for many authentication attempts in a single request
- ➔ Lacking authentication for sensitive requests

Credential Stuffing

- ➢ Attack using large number of username and password combinations
- ➢ Credentials used in these types of attacks are typically collected from data breaches
- ➢ Allows users to brute force many username and password combinations

Predictable Tokens

- ❖ Tokens obtained through a week token generation authentication process
- ❖ Weak tokens can easily be guessed, deduced, or calculated by an attacker
- ❖ Using incremental or guessable token IDs

Misconfigured JSON Web Tokens

- JWTs commonly used for API authentication and authorization
- Provide flexibility to customize algorithms used for signing the token, the key/secret that is used, and the information used in the payload
- Security Misconfigurations:
    - API Provider accepts unsigned JWT tokens

- API Provider does not check JWT expiration
- API Provider discloses sensitive information within encoded JWT Payload
- JWT signed with a weak key

API Authentication

★ Rest requires APIs to be stateless
★ APIs, therefore, require users to register to acquire tokens
★ Token is used for future requests

Issues With Authentication

★ Insufficient token randomness(entropy)
★ Vulnerabilities in registration process
★ password reset process
★ Multi-factor authentication features

OWASP Preventative Measures

★ Know all the possible flows to authenticate to the API
★ Understand your authentication mechanism and how they are used - OAuth is not authentication, and neither are API keys
★ Use standards for authentication, token generation, or password storage
★ treat credential recovery/forgot password endpoint as login endpoints in terms of brute force, rate limiting, and lockout protection
★ Require re-authentication for sensitive operations
★ Use the OWASP Authentication cheatsheet
★ Implement multi-factor authentication, where possible
★ Implement anti-brute force mechanism to mitigate credential stuffing, dictionary attacks, and brute force attacks.
★ Implement account lockout/captcha mechanism to prevent brute force attakcs against specific users
★ Implement weak password checks
★ API keys should not be used for user authentication - they should only be used for API clients authentication

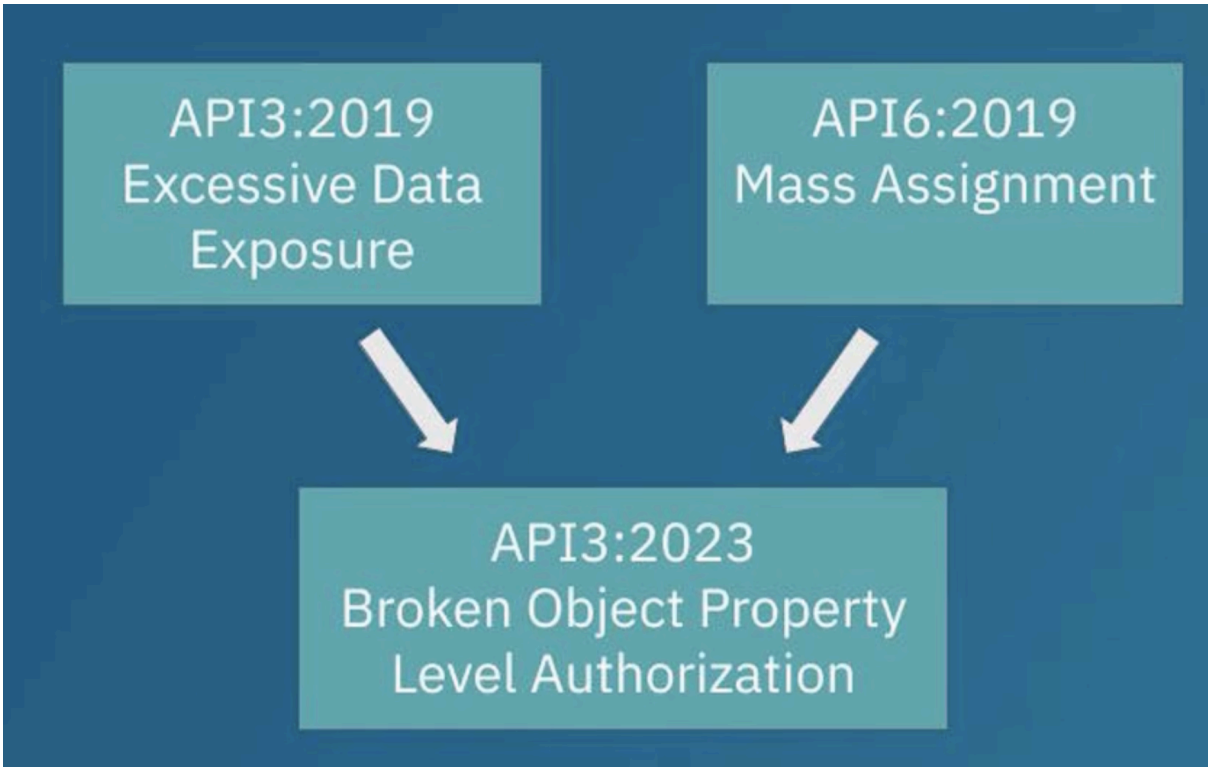
API3:2023 Broken Object Property Level Authorization(BOPLA)

Excessive Data Exposure:

- API responds with an entire data object
- Usually APIs filter data to what is being requested
- Relying on consumer/UI to filter creates risk of exposing sensitive information

Mass Assigment:

- Mass assignment allows user input to alter sensitive object properties
- E.g, API uses a special property to create admin accounts
    - only authorized users should be able to make requests
    - If no restrictions in place then attacker could elevate privileges and perform administrative actions.



Attack Vectore Description

“APIs tend to expose endpoints that return all object’s properties. this is particularly valid for Rest APIs. For other protocols such as GraphQL, it may require crafted requests to specify which proporties should be returned. identifying these additional proporties that can be manipulated requires more effort, but there are a few automated tools available to assist in this task”.

Security Weakness Description

“Inspecting API responses is enough to identify sensitive information in returned objects’ representations. Fuzzing is usually used to identify additional(hidden) properties. whether they can be changed is a matter of crafting an API request and analyzing the response. Side-effect analysis may be required if the target property is not returned in the API response”.

Impacts Description

“Unauthorized access to private/sensitive object properties may result in data disclosure, data loss, or data corruption. Under certain circumstances, unauthorized access to object properties can lead to privilege escalation or partial/full account takeover.”

BOPLA Vulnerabilities

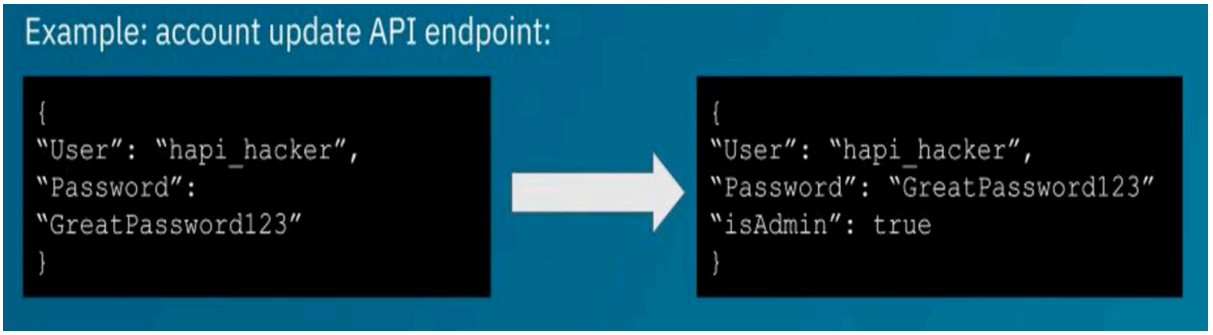
- The API endpoint exposes properties of an object that are considered sensitive and should not be read by the user
- The API endpoint allows a user to change, add/or delete the value of a sensitive object’s property which the user shoul not be able to access

Test For Data Exposure

- ➔ Excessive Data Exposure is very difficult to detect with automated scanners.
- ➔ Often bypasses every security control in place to protect sensitive information
- ➔ Can deliver sensitive data to an attacker simply because they use the API
- ➔ API providers must test API endpoints and review information in response

Mass Assignment

- ➔ API consumer includes more parameters than the application intended
- ➔ Application adds these parameters to code variables or internal objects
- ➔ A consumer may be able to edit object properties or escalate privileges



OWASP Preventative Measures

- ➔ Always make sure users should have access to the object' properties

➔ Avoid using generic method such as to_json() and to_string()
➔ Avoid functions that automatically bind input into code variables, objects, or properties
➔ Only allow changes to object's properties that should be updated by the client
➔ Implement response validation mechanism as an extra layer of security
➔ define and enforce data returned by all API methods
➔ minimize returned data structures, according to the business/functional requirements

API4:2023 Unrestricted Resource Consumption

- Formerly "Lack of Resources and Rate Limiting"

Attack Vector Description

➔ Exploited via simple API requests
➔ Multiple request from single source, or across clouds
➔ Automated tools, simulate DoS via high traffic loads

Security Weakness Description

➢ APIs commonly do not limit interactions or resource limits
➢ Test for resource consumption weakness via:
    - API requests that specify resource quantities
    - Analyzing response status/time/length

➢ Also valid for batched operations
➢ Evaluate cost impact based on service provider pricing

Impacts Descriptions

➢ Exploitation can lead to Denial of Service
➢ Economic impact due to higher infrastructure demands

Summary

- Every API request has technical & financial cost
- Lack of rate controls increase risk of :
  - ➔ Denial of Service
  - ➔ Unnecessary financial cost
  - ➔ Degradation of service to users

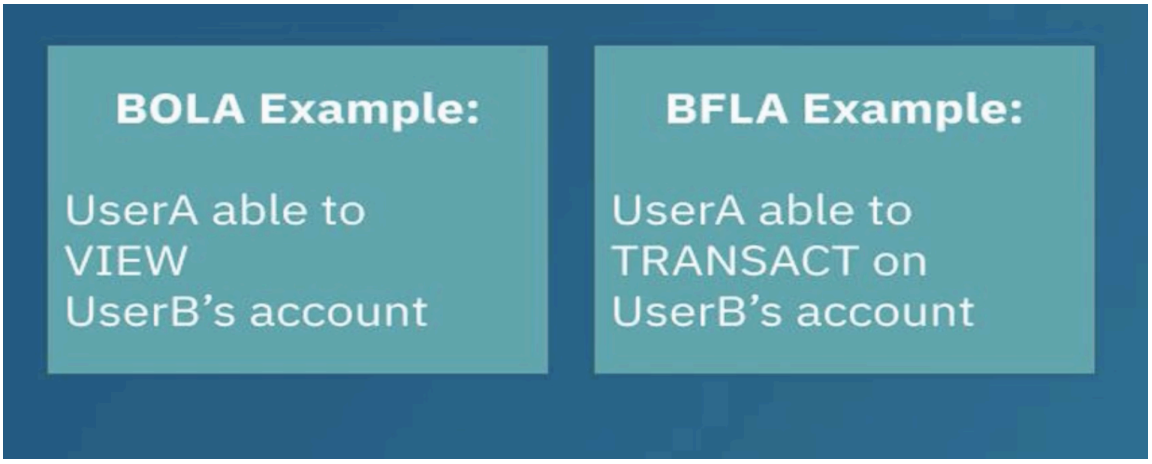
OWASP Presentative Measures

➔ Docker offers controls for memory, GPU,restarts, file descriptors, processes
➔ Limit numbers of API calls within timeframe
➔ Notify client when limit is exceeded
➔ Add server-side validation for inputs
➔ Define, enforce max size for inputs&payloads

API5:2023 Broken Function Level Authorization

➔ BFLA is a vulnerability where API functions have insufficient access controls. Where BOLA is about access to data, BFLA is about altering or deleting that data.
➔ In addition, a Vulnerable API would allow an attacker to perform actions of other roles, including administrative actions.

BOLA vs BFLA



Attack Vector Description

"Exploitation requires the attacker to send legitimate API calls to an API endpoint that they should not have access to as anonymous users or regular, non-privileged users. Exposed endpoints will be easy exploited."

Security Weakness Description

"Authorization checks for a function or resource are usually managed via configuration or code level.Implementing proper checks can be a confusing task since modern applications can contain many types of roles, groups, and complex user hierarchies. It's easier to discover these flaws in APIs since APIs are more structured, and accessing different functions is more predictable."

Impacts Description

"Such flaws allows attackers to access unauthorized functionality.Administrative functions are key targets for this type of attack and may lead to data disclosure, data loss, or data corruption. Ultimately, it may lead to service disruption.

BFLA Vulnerability

- ➢ User of one privilege level can use functionality of another user
- ➢ API Providers often have different privilege levels
  - Public users, merchants, partners,vendors, admins, ….

➢ Can be exploited for :
- Unauthorized use of lateral functions
- Privilege Escalation

➢ Targeted API functions- endpoint with:
- Sensitive Information
- Resources that belong to another group
- Administrative functionality like user account management

BFLA Example

➢ Different endpoints for privileged actions
- /{userid}/account/balance - user account information
- /admin/account/{userid} - administrator endpoint

➢ Admin endpoint can be exploit if API has improper access controls
- An attacker can perform admin actions, account takeovers

➢ APIs don’t always have admin endpoints
- HTTP request methods can control access to functionality
- E.g. , GET,POST,PUT, and DELETE

➢ Attack can exploit unrestricted methods to exploit API functionality

Testing for BFLA

➢ Identify endpoints an attacker could use to their advantage
➔ Altering user accounts
➔ Deleting user resources
➔ Gaining access to restricted endpoints

OWASP Preventative Measures

➢ Have consistent, easy-to-analyze authorization module for all business functions

- ➢ Deny all access by default, require explicit grants to specific roles for access to every function
- ➢ Review endpoints against function level authorization flaws
- ➢ Ensure admin controllers inherit from an administrative abstract controller that implements authorization checks based on the user's group/role
- ➢ Ensure admin functions inside a regular controller implement authorization checks based on the user's group/role

API6:2023 - Unrestricted Access to Sensitive Business Flows

- Unrestricted Access to Sensitive Business Flows represents the risk of an attacker being able to identify and exploit API driven workflows. If vulnerable an attacker will be able to leverage an organization's API request structure to obstruct other users.
- This obstruction could come in the form of spamming other users, depleting the stock of highly sought after items, or preventing other users from using expected application functionality.

Attack Vector Description

"Exploitation usually involves understanding the business model backed by the API, finding sensitive business flows, and automating access to these flows, causing harm to the business."

Security Weakness Description

"Lack of a holistic view of the API in order to fully support business requirements tends to contribute to the prevalence of this issue. Attackers manually identify what resources are involved in the target workflow and how they work together. If mitigation mechanism are already in place, attackers need to find a way to bypass them.

Impact Description

“In general technical impact is not expected. Exploitation might hurt the business in different ways, for example : prevent legitimate users from purchasing a product, or lead to inflation in the internal economy of a game.”

OWASP Preventative Measures

➔ The mitigation planning should be done in two layers:
- Business - identify the business flows that might harm the business if they are excessively used
- Engineering - choose the right protection mechanism to mitigate the business risk

➔ Slow down automated threats
- Device fingerprinting: deny service to unexpected client devices
- Human detection: use captcha or more advanced biometric solutions
- Non-humans patterns: analyze traffic to detect non-human patterns
- Consider blocking IP addresses of Tor exit nodes, well-known proxies

➔ Secure and limit access to APIs consumed directly by machines

API7:2023 - SSRF(Server Side Request Forgery)

- SSRF is a vulnerability that takes place when a user is able to control the remote resources retrieved by an application.

- An attacker can use an API to supply their own input in the form of a URL to control the remote resources that are retrieved by the targeted server.
- An attacker could supply URLs that expose private data, scan the targets internal network, or compromise the target through remote code execution.

Attack Vector Description

“Exploitation requires the attacker to find an API endpoint that accesses a URL that’s provided by the client. In general, basic SSRF(when the response is returned to the attacker), is easier to exploit that Blind SSRF in which the attacker has no feedback on whether or not the attack was successful.

Security Weakness Description

“Modern concepts in application development encourage developers to access URLs provided by the client. Lack of or improper validation of such URIs are common issues. Regular API requests and response analysis will be required to detect the issue. When the response is not returned(Blind SSRF) detecting the vulnerability requires more effort and creativity.”

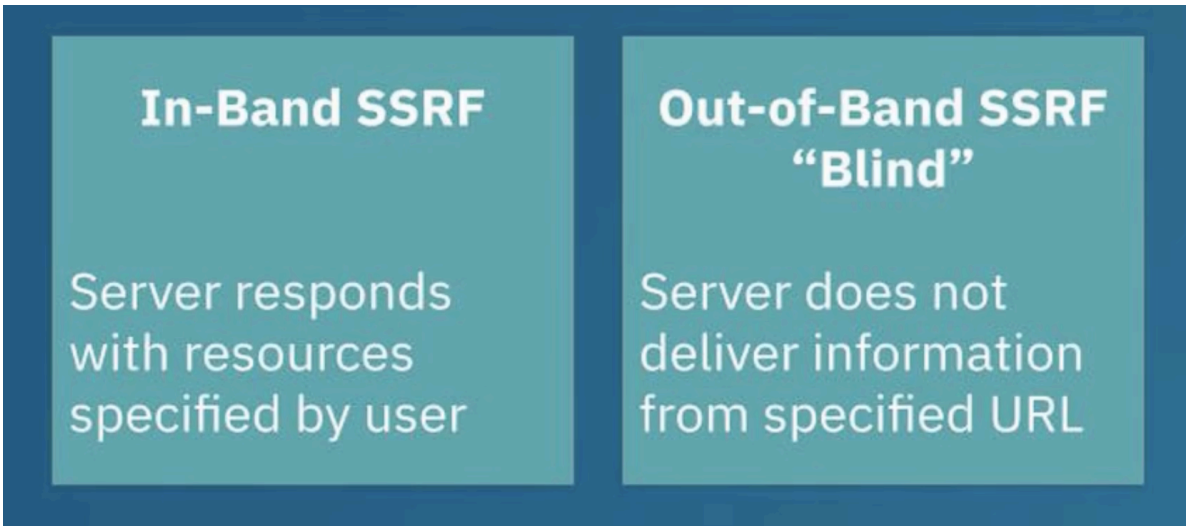
Impacts Description

“Succesfull exploitation might lead to internal services enumeration, information disclosure, bypassing firewalls, or other security mechanisms. In some cases, it can lead to DoS or the server being used as a proxy to hide malicious activities.”

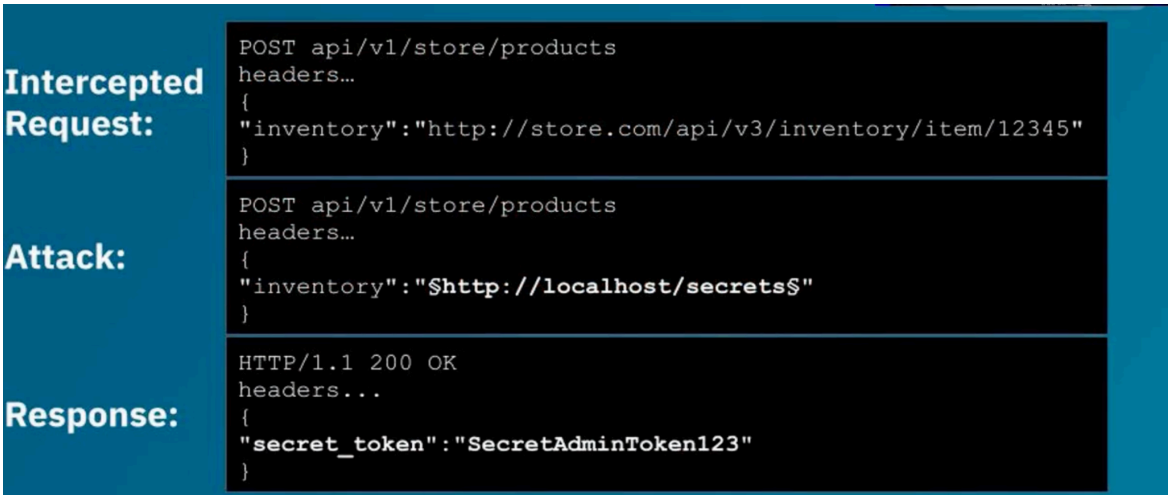
SSRF Summary

- App retrieves remote resources without input validation
- Attacker can then control what resources a server requests
- Allows attackers to access sensitive data, compromise host
- Attacker leverages target server to process their requests
- SSRF bug bounty payout based on level of impact

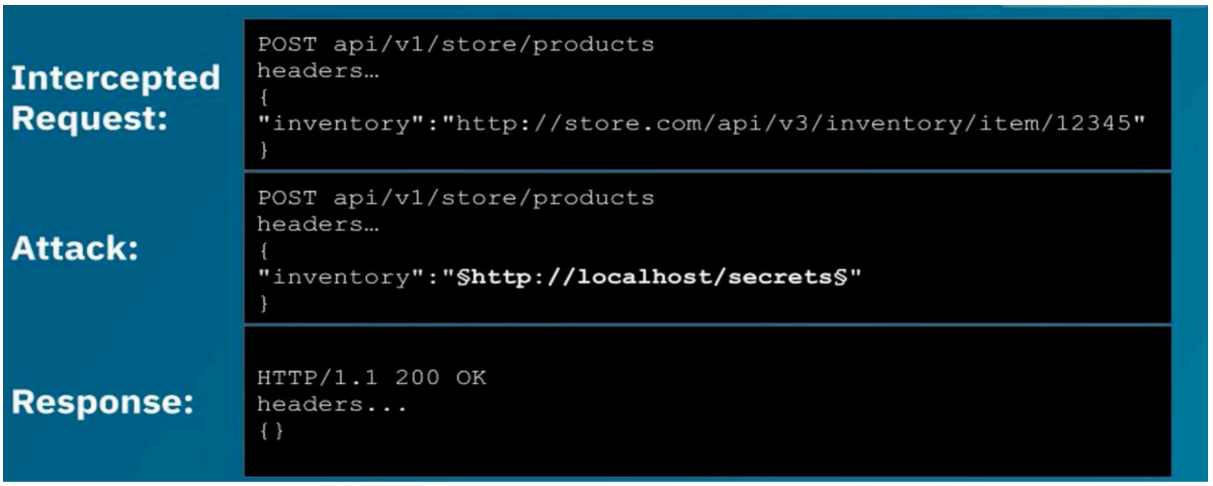
SSRF Types



In-Band SSRF



Out-of-Band SSRF(Blind)



OWASP Preventative Measures

- Isolate the resource fetching mechanism in your network
- Use allow list of 1) remote origins 2) URL schemes, ports, 3) accepted media types
- Disable HTTP redirections
- Use URL parser
- Validate and sanitize all client-supplied input data
- Do not send raw responses to clients

API8:2023 SECURITY MISCONFIGURATION

Security Misconfiguration represents a catch-all for many vulnerabilities related to the systems that host APIs. When an APIs security is misconfigured, it can be detrimental to the confidentiality, integrity , and availability of the API provider’s data.

Attack Vector Description

“Attackers will often attempt to find unpatched flaws, common endpoints, or unprotected files and directories to gain unauthorized access or knowledge of the system.”

Security Weakness Description

“Security Misconfiguration can happen at any level of the API stack, from the network level to the application level. Automated tools are available to detect and exploit misconfiguration such as unnecessary services or legacy options.

Impact Description

“Security Misconfiguration can not only expose sensitive user data, but also system details that can lead to full server compromise.”

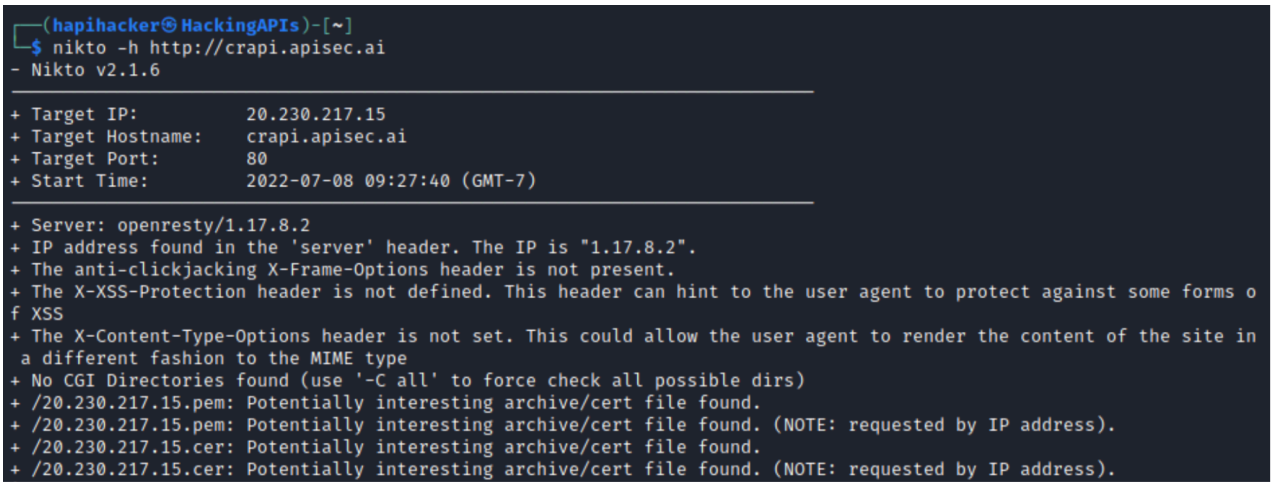
Misconfiguraiton Examples

- Misconfiguration Headers
- Misconfigured transit encryption
- Use of default accounts
- Acceptance of unnecessary HTTP Headers
- Lack of Input sanitization
- Verbose error messaging

API Header Misconfiguration

- API providers use headers to provide the consumer with instructions for handling the response and security requirements.
- Misconfigured headers can result in sensitive information disclosure, downgrade attacks, and cross-site scripting attacks.
- Many API providers will use additional services alongside their API to enhance API-related metrics or to improve security. It is fairly common that those additional services will add headers to

requests for metrics and perhaps as some level of assurance to the consumer.



X-Powered-BY Header

- The X-Powered-By header reveals backend technology.
- Headers like this one will often advertise the exact supporting service and its version.
- You could use information like this to search for exploits published for that version of software.

X-XSS-Protection Header

- X-XSS-Protection is exactly what it looks like: a header meant to prevent cross-site scripting (XSS) attacks.
- XSS is a common type of injection vulnerability where an attacker could insert scripts into a web page and trick end-users into clicking on malicious links.
- An X-XSS-Protection value of 0 indicates no protections in place and a value of 1 indicates that the protection is turned on. This header, and others like it, clearly reveals whether or not a security control is in place.

X-Response-Time Header

- The X-Response-Time header is middleware that provides usage metrics.

- If API isn’t configured properly this header reveal existing resources
- X-Response-Time may differ for existing vs non-existing records
- Example:

    - /user/account/thisdefinitelydoesnotexist -responce time 25.5ms
    - /user/account/1021(valid account) -responce time 510 ms

TRANSPORT LAYER SECURİTY

- APIs with sensitive info should use Transport Layer Security
- TLS is fundamental way to protect data communication
- Misconfigured encryption can pass sensitive API info in cleartext
- Attacker could capture the responses and requests MITM attack.

Default Attacks & Credentials

- Attackers will attempt default credentials
- Could allow:
    - Acces to sensitive information
    - Access to administrative functionality

- Potentially lead to compromise of supporting systems

HTTP Methods

- Unnecessary HTTP Methods can increase risk
- Application may not handle these methods properly
- Potential disclosure of sensitive information

Testing for Misconfiguration

- Some Security Misconfiguration can be detected by web app scanners
- Automated scanners will automatically check responses to determine :
    - Version information , Headers ,cookies, Transit encryption configuration, parameters

- Security misconfigurations can also be checked manually

OWASP Preventative Measures

The API life cycle should include:

- A repeatable hardening process leading to fast and easy deployment of a properly locked down environment
- A task to review and update configurations across the entire API stack. The review should include: orchestration files, API components, and cloud services (e.g. S3 bucket permissions)
- An automated process to continuously assess the effectiveness of the configuration and settings in all environments

Furthermore:

- Ensure that all API communications from the client to the API server and any downstream/upstream components happen over an encrypted communication channel (TLS), regardless of whether it is an internal or public-facing API.
- Be specific about which HTTP verbs each API can be accessed by: all other HTTP verbs should be disabled (e.g. HEAD).
- APIs expecting to be accessed from browser-based clients (e.g., WebApp front-end) should, at least:
    - implement a proper Cross-Origin Resource Sharing (CORS) policy
    - include applicable Security Headers

- Restrict incoming content types/data formats to those that meet the business/ functional requirements.
- Ensure all servers in the HTTP server chain (e.g. load balancers, reverse and forward proxies, and back-end servers) process incoming requests in a uniform manner to avoid desync issues.
- Where applicable, define and enforce all API response payload schemas, including error responses, to prevent exception traces and other valuable information from being sent back to attackers.

API9:2023 IMPROPER INVENTORY MANAGEMENT

Improper Inventory Management represents the risks involved with exposing non production and unsupported API versions. When this is present, the non production and unsupported version of the API are often not protected by the same security rigor as the production versions This makes improper inventory management a gateway to other API security vulnerabilities.

Attack Vector Description

“Threat agents usually get unauthorized access through old API versions or endpoints left running unpatched and using weaker security requirements. Alternatively, they may get access to sensitive data through a 3rd party with whom there's no reason to share data with.”

Security Weakness Description

“Outdated documentation makes it more difficult to find and/or fix vulnerabilities. Lack of assets inventory and retirement strategies leads to running unpatched systems, resulting in leakage of sensitive data. It's

common to find unnecessarily exposed API hosts because of modern concepts like microservices, which make applications easy to deploy and independent (e.g. cloud computing, K8S). Simple Google Dorking, DNS enumeration, or using specialized search engines for various types of servers (webcams, routers, servers, etc.) connected to the internet will be enough to discover targets."

Impact Description

"Attackers can gain access to sensitive data, or even take over the server. Sometimes different API versions/deployments are connected to the same database with real data. Threat agents may exploit deprecated endpoints available in old API versions to get access to administrative functions or exploit known vulnerabilities."

Summary

- Improper Inventory Management Exposes unsupported APIs
- Old API versions are more likely to contain Vulnerabilities
- Can lead to other vulnerabilities:
    - excessive data exposure, information disclosure, mass assignment, improper rate-limiting, API injection

Detecting Improper Inventory

- Test with outdated API documentation, changelog, and version history on repositories
- Version information in endpoints: /v1/,/v2/,/v3/
- APIs development paths: /alpha/, /beta/, /test/, /uat/, /demo/
- Techniques: guessing, fuzzing or brute force requests
- Testing focuses on unsupported and non-production API versions
- Typical new version paths:
    - api.target.com/v3
    - /api/v2/accounts
    - /api/v3/accounts
    - /v2/accounts

- API versioning could also be maintained as a header:
  - Accept: version=2.0
  - Accept api-version=3.0
- Versioning can be set within query parameter or request body:
  - /api/accounts?ver=2
  - POST /api/accounts

    ```
    {
     “ver”:1.0,
     “user”:”hapihacker”
    }
    ```

- Non-production API versions examples:
  - api.test.target.com
  - api.uat.target.com
  - beta.api.com
  - /api/private/
  - /api/partner
  - /api/test

OWASP Preventative Measures

- Inventory all API hosts and document important aspects of each one of them, focusing on the API environment (e.g., production, staging, test, development), who should have network access to the host (e.g., public, internal, partners) and the API version.
- Inventory integrated services and document important aspects such as their role in the system, what data is exchanged (data flow), and its sensitivity.
- Document all aspects of your API such as authentication, errors, redirects, rate limiting, cross-origin resource sharing (CORS) policy and endpoints, including their parameters, requests, and responses.

- Generate documentation automatically by adopting open standards. Include the documentation build in your CI/CD pipeline.
- Make API documentation available to those authorized to use the API.
- Use external protection measures such as API security firewalls for all exposed versions of your APIs, not just for the current production version.
- Avoid using production data with non-production API deployments. If this is unavoidable, these endpoints should get the same security treatment as the production ones.
- When newer versions of APIs include security improvements, perform risk analysis to make the decision of the mitigation actions required for the older version: for example, whether it is possible to backport the improvements without breaking API compatibility or you need to take the older version out quickly and force all clients to move to the latest version.

API10:2023 UNSAFE CONSUMPTİON of APIs

Unsafe Consumption of APIs is the only item on the top ten list that focuses less on the risks of being an API provider and more on the API consumer. Unsafe consumption is really a trust issue. When an application is consuming the data of third-party APIs it should treat those with a similar trust to user input. By that, I mean, there should be little to no trust. So, data consumed from third-party APIs should be treated with similar security standards as end-user-supplied input. If a third-party API provider is compromised then that insecure API connection back to the consumer becomes a new vector for the attacker to leverage. In the case of an insecure API connection, that could mean the complete compromise of organizations insecurely consuming data from that provider.

Attack Vector Description

“Exploiting this issue requires attackers to identify and potentially compromise other APIs/services the target API integrated with. Usually, this information is not publicly available or the integrated API/service is not easily exploitable.”

Security Weakness Description

“Developers tend to trust and not verify the endpoints that interact with external or third-party APIs, relying on weaker security requirements such as those regarding transport security, authentication/authorization, and input validation and sanitization. Attackers need to identify services the target API integrates with (data sources) and, eventually, compromise them.”

Impact Description

“The impact varies according to what the target API does with pulled data. Successful exploitation may lead to sensitive information exposure to unauthorized actors, many kinds of injections, or denial of service.”

OWASP Preventative Measures

- When evaluating service providers, assess their API security posture.
- Ensure all API interactions happen over a secure communication channel (TLS).
- Always validate and properly sanitize data received from integrated APIs before using it.
- Maintain an allowlist of well-known locations integrated APIs may redirect yours to: do not blindly follow redirects.

Injection

Injection vulnerabilities have plagued web applications for over two decades. They take place when an attacker is able to send commands that are executed by the systems that support the web application. The most common forms of injection attacks are SQL injection, Cross-site scripting (XSS), and operating system command injection. APIs are yet

another attack vector for these critical attacks to be communicated from an attacker to the supporting databases and systems.

Attack Vector Description

“Attackers will feed the API with malicious data through whatever injection vectors are available (e.g., direct input, parameters, integrated services, etc.), expecting it to be sent to an interpreter.”

Security Weakness Description

“Injection flaws are very common and are often found in SQL, LDAP, or NoSQL queries, OS commands, XML parsers, and ORM. These flaws are easy to discover when reviewing the source code. Attackers can use scanners and fuzzers.”

Impact Description

“Injection can lead to information disclosure and data loss. It may also lead to DoS, or complete host takeover.”

Summary

- Injections occur when API don’t filter input to remove unwanted characters
- Infrastructure might treat data from the request as code and run it
- Enables: SQL injection, NoSQL injection, system command injection
- Example:
    - Attacker sends SQL commands in payload to vulnerable API
    - API passess commands to the database, which performs the commands
- Clues: verbose error messaging,HTTP response codes,unexpected behavior

OWASP Preventative Measures

- Perform data validation using a single, trustworthy, and actively maintained library.
- Validate, filter, and sanitize all client-provided data, or other data coming from integrated systems.
- Special characters should be escaped using the specific syntax for the target interpreter.
- Prefer a safe API that provides a parameterized interface.
- Always limit the number of returned records to prevent mass disclosure in case of injection.
- Validate incoming data using sufficient filters to only allow valid values for each input parameter.
- Define data types and strict patterns for all string parameters.

Insufficient Logging and Monitoring

Logging and monitoring are necessary and important layer of API security. In order to detect an attack against an API an organization must have monitoring in place. Without sufficient logging and monitoring an API provider is operating in the dark and API attacks are guaranteed to go unnoticed until it is far too late.

Attack Vector Description

“Attackers take advantage of lack of logging and monitoring to abuse systems without being noticed.”

Security Weakness Desciption

“Without logging and monitoring, or with insufficient logging and monitoring, it is almost impossible to track suspicious activities and respond to them in a timely fashion.”

Impact Description

“Without visibility over ongoing malicious activities, attackers have plenty of time to fully compromise systems.”

Summary

- Logs reveal patterns in API usage, can also identity abuse
- Logs provide audit trail of activities
- Often required for compliance purposes
- Ensure logs cannot be altered by attackers
- Monitoring helps detect suspicious activities, anomalous behavior

OWASP Preventative Measures

- Log failed authentication, denied access, input validation errors
- Write logs for a log management solutions, include detail to identify malicious actor
- Logs should be handled as sensitive data
- Continuously monitor infrastructure, network, API functioning
- Use a Security Information and Event Management(SIEM)
- Configure custom alerts to detect suspicious activities earlier

Business Logic Flaws

Business logic vulnerabilities are weaknesses within applications that are unique to the policies and features of a given API provider. The exploitation of business logic takes place when an attacker leverages misplaced trust or features of an application against the API. Identifying business logic vulnerabilities can be challenging due to the unique nature of each business. The impact of these vulnerabilities can range based on the severity of the vulnerable policy or feature.

Attack Vector Description

“Business logic vulnerabilities are unique to each application and exploit the normal intended functioning of an application's business processes. They often require specific knowledge of the application's functionality and the flow of transactions or data. Since these vulnerabilities are specific to the business logic of each application, there's no one-size-fits-all approach to identifying them.”

Security Weakness Description

“Business logic vulnerabilities arise when the assumptions and constraints of a given business process aren't properly enforced in the application's control structures. This allows users to manipulate the application's functionality to achieve outcomes that are detrimental to the business. These weaknesses typically occur when developers fail to anticipate the various ways that an application's features can be misused or when they don't consider the wider context of the business rules. This is often due to a lack of comprehensive understanding of the application's business logic, a lack of input validation, or incomplete function-level authorization checks.”

Impact Description

“Business logic vulnerabilities can cause a variety of technical impacts, depending on the specific flaw and the systems involved. These impacts can range from unauthorized access to data or functionality to a total bypass of system controls.”

Summary

- Business Logic Flaws are misuse/abuse of intended app functionality
- Example, an API allows only upload certain encoded payloads
    - But doesn’t validate the encoded payloads
    - Allowing a user could upload any encoded file
    - Users to upload and potentially execute arbitrary code
- BLF come from assumption that API consumers will follow directions, be trustworthy, or only use the API in a certain way
- Organization rely on trust as a security control, expecting users to act benevolently
- Unfortunately, even good-natured API consumers make mistakes

BLF Example

- Experian Partner API leak: example of an API trust failure
- Experian partner authorized to use Exprians API to perform credit checks
- Partner added the APIs credit check functionality to their web application
- Inadvertently exposed all partner-level requests to users
- Request could be intercepted, Experian API would respond with credit scores
- Experian trusted the partner to not expose the API

Finding Vulnerabilites

- Credentials(API keys, tokens,passwords) constantly stolen and leaked
- Logic vulnerabilities often have most significant impact
- Examine API documentation for signs of business logic vulnerabilities.

  Statements Like:

  - “Only use feature x to perform function y. ”
  - “Do not do x with endpoint y. ”
  - “Only admins should perform request x.”
- These statements may indicate reliance on trust
- Attacker will disobey such requests and to test for technical security controls.

Bypassing Web UIs

- Cannot assume users will exclusively use a browser
- Attacker can easily intercept requests and alter the API request
- This could allow attackers to capture shared API keys or abuse parameters

```
POST /api/v1/login HTTP 1.1
Host: example.com
--snip--
UserId=hapihacker&password=arealpassword!&MFA=tru
```

- Attacker could bypass MFA by simply altering the parameter MFA to false

Preventative Measures

- Use a Threat Modeling Approach: Understand the business processes and workflows your API supports. Identifying the potential threats, weaknesses, and risks during the design phase can help to uncover and mitigate business logic vulnerabilities.

- Reduce or remove trust relationships with users, systems, or components. Business logic vulnerabilities can be used to exploit these trust relationships, leading to broader impacts.

- Regular training can help developers to understand and avoid business logic vulnerabilities. Training should cover secure coding practices, common vulnerabilities, and how to identify potential issues during the design and coding phases.
- Implement a bug bounty program, third-party penetration testing, or a responsible disclosure policy. This allows security researchers, who are a step removed from the design and delivery of an application, to disclose vulnerabilities they discover in APIs.