API and its Popularity: -

In last 10 years APIs have gained so much popularity and become the most preferable and common way of exposing services, features and functions which one's business want to offer to the outer world. APIs are being used in almost all industries, especially the IoT industry which thrives and builds on the foundation of wide usage of APIs. This world is more connected after the emergence and adoption of APIs with open arms.

Salesforce, Uber, Amazon, Facebook, Twitter adopted APIs at very early stage and their large chunk of revenue comes from their API business. Netflix receives 2+ billion public API requests every day which supports the popularity of APIs and shows how beneficial they can be for any business.

Financial Institutions, Healthcare, Educational Institutions, Energy sector and many more sectors, which are at the core of any country, are using APIs and handling very Sensitive to Top Secret information every second of every day therefore API security becomes more important than it has ever been.

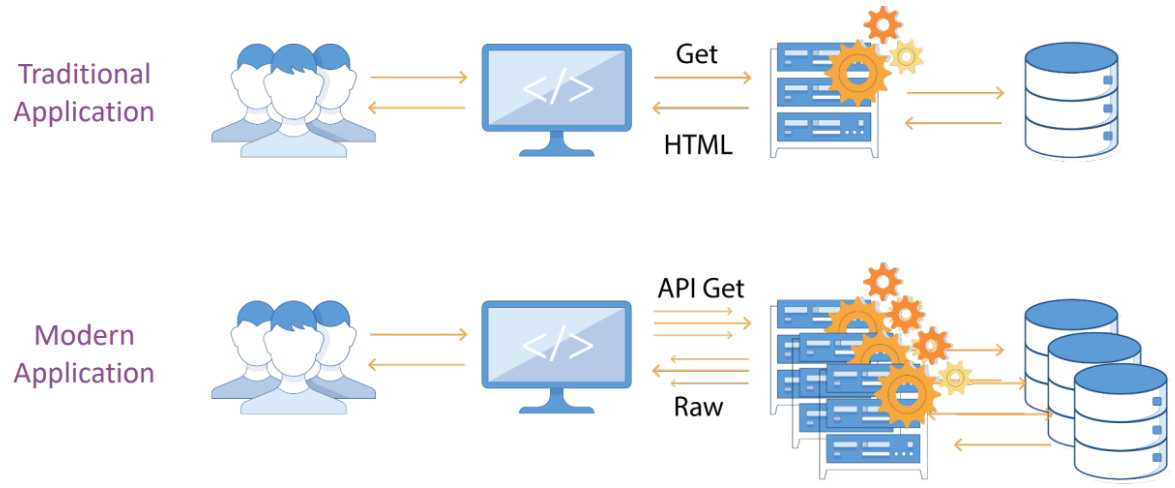
Security: -

It is also very common that security is an afterthought. Security folks are seen as road blocks who nit-pick at design flaws and are responsible for delaying the project.

However, the status quo has been changing and security is getting involved from the planning stage of the project, establishing relationships with broader team to help them adopt security first mindset and baking the security controls into the design at the very early stage so that manoeuvring becomes manageable at the end if required.

Industries reliance on APIs have increased at a very fast pace in last 10 years and is driven by business need therefore security was not a priority at the design phase of an API. However, security standards for API security have also been developed and released, OAuth 2.0 is one of the widely known and adopted standards used to ensure security of APIs. A detailed and comprehensive security review on APIs going into production was/is still missing.

WHY??? :-

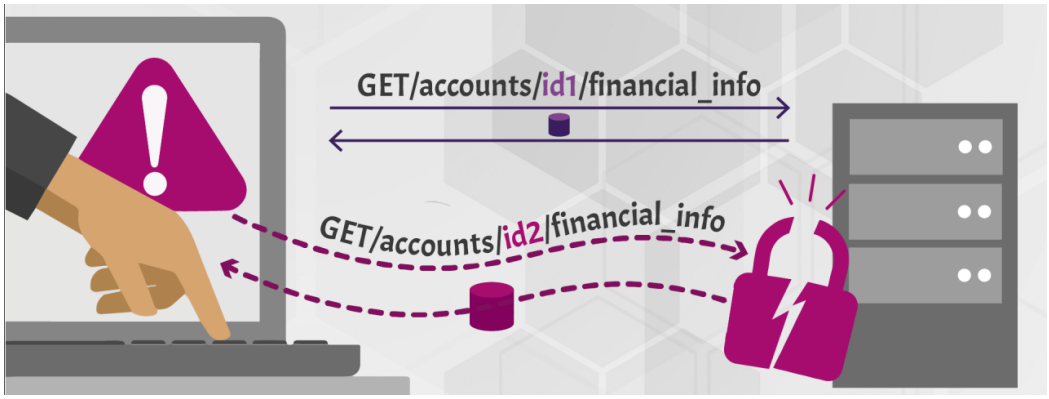


Top 10 API Security Flaws

1. **Broken Object Level Authorization aka IDOR (Insecure Direct Object Reference)**: -

   Attackers can exploit API endpoints that are vulnerable to broken object level authorization by manipulating the ID of an object that is sent within the request. This may lead to unauthorized access to sensitive data.

   This issue is extremely common in API-based applications because the server component usually does not fully track the client's state, and instead, relies more on parameters like object IDs, that are sent from the client to decide which objects to access.



   How it's related to API: -

   Every API endpoint that receives an ID of an object and performs any type of action on the object based on the input received without validating the input could be vulnerable to this attack.

   Mitigation: -

   - Implement authorization checks with user policies and hierarchy
   - Don't rely on IDs sent from client instead use IDs stored in session object
   - Check authorization each time there is a client request to access database
   - Randomization of UUIDs is also a deterrent control

2. **Broken Authentication**: -

API Authentication is one of the more complex and confusing topics. Often developers and security folks carry different approach in terms of deciding the boundary and the point where the authentication must happen and how to implement it correctly.

Therefore, bad actors love to exploit flaws in authentication process. Additionally, two more configuration/design flaws also make it an easier target. These are Lack of protection Mechanism and incorrect implementation of such mechanisms. Having one blanket rule is not helpful at all the time, authentication solution must be tweaked to cater for different authentication mechanism under different applications and workflows.

How it's related to API: -

- APIs permits Credential Surfing, Brute force attempts and supports weak passwords
- Does not validate tokens, accepts unsigned tokens, does not validate expiry of tokens,
- Use weak encryption, send sensitive data unencrypted and weak hashing algorithms.

Mitigation: -

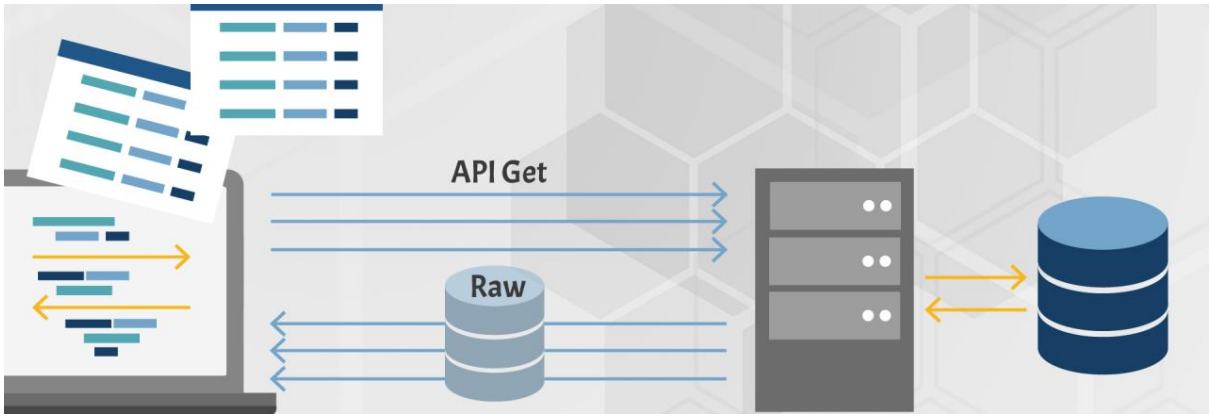
- Know all possible flow (Mobile, Web, social media) of authentication, overlooking does not bring best results.
- Use established standards for authentication, token generation, Password storage, MFA etc.
- Password reset API and one-time links are the keys to citadel need to be carefully protected
- Short lived access tokens, Rate limit your authentication and **Captcha**
- Authenticate your apps to tie all the loose ends

3. **Confidential Data Exposure: -**

This is a very simple attack to exercise as bad actors do not even need any fancy tools to exploit this vulnerability. In laymen terms when an application server returns too much information to the client who have not requested that data, it is susceptible to this vulnerability. That data could be sensitive in nature, confidential and might have some information which could help the attacker to weaponized their exploit.

APIs rely on clients to perform the data filtering. Since APIs are used as data sources, sometimes developers try to implement them in a generic way without thinking about the sensitivity of the exposed data.

Automatic tools sometimes are not that effective to detect this vulnerability. Manual fuzzing an application by going through the response line by line could hand out very useful information to bad actors.



How it's related to API: -

APIs tend to do maximum processing at client end therefore once the data is already on client's turf it's too late to put any barricade, therefore APIs which are weak by design tend to provide sensitive data inadvertently.

Mitigation: -

- Do not rely on client to filter data
- Only return the data what API consumers requested
- Mask error message
- Define schemas of all the API response
- Identify PII, mask it or remove it of not required
- Enforce Response check to prevent accidental data and exception leaks

4. **Lack of Resource and Rate limiting aka DoS** :-

This is like any denial-of-service attack. Resource exhaustion of an application server by sending enormous amounts of API requests by bad actors could end up putting that application server out of service and legitimate API requests would be waiting for long to get required information.

To exercise this attack no authentication is required which empowers even script kiddies to perform such attacks. Attacks can be generated from a single machine using very basic knowledge of linux or using cloud computing resources.

How it's related to API: -

There are certain API configuration items which would come in picture when we deal with APIs. All these configuration items are the gate keepers to avoid resources from being exhausted.

- Execution timeouts
- Max allocable memory
- Number of file descriptors
- Number of processes
- Request payload size (e.g., uploads)
- Number of requests per client/resource
- Number of records per page to return in a single request response

Mitigation: -

- Rate limiting, Notify the client when the limit is exceeded by providing the limit number and the time at which the limit will be reset.
- Payload size limits
- Limits on container/Docker resources,
- Add proper server-side validation for query string and request body parameters, specifically the one that controls the number of records to be returned in the response.

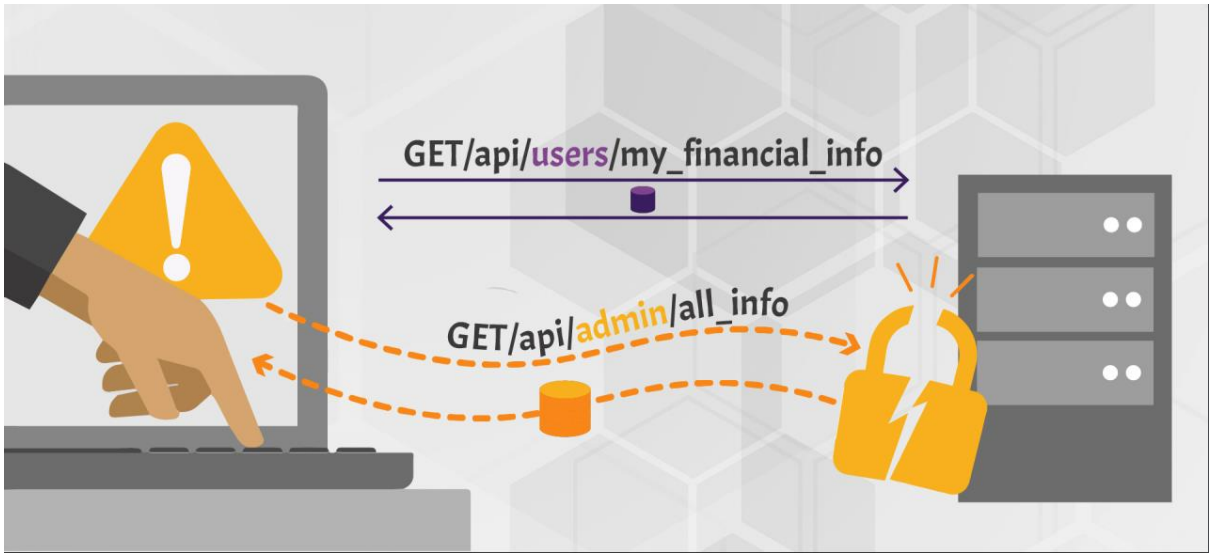
- CAPTCHA

5. **Function level Authorization: -**

This vulnerability is related to IDOR which is at number one the only difference here is that the attacker tries to impersonate someone with higher or different level of privileges by changing the parameters after making an educational guess.

The attacker needs to send a legitimate request to API endpoint by predicting the string or hit & trial method to retrieve something which he/she is not unauthorized.

Authorization checks for a function or resource are usually managed via configuration, and sometimes at the code level. Implementing proper checks can be a confusing task, since modern applications can contain many types of roles or groups and complex user hierarchy



How it's related to API: -

If a bad actor manages to access administrative API endpoints after changing certain parameters in the URL then that API is vulnerable to this specific vulnerability. Additionally, if a standard user can perform sensitive functions and is able to access resources which only certain groups or users can access make this API vulnerable to broken function level authorization attacks.

Mitigation: -

- Don't rely on app /client to enforce to admin access this has to be done at backend
- Deny all access by default, make default access to minimum / read only access
- RBAC principals must be implemented followed by the business logic of the app

6. **Mass Assignment: -**

Certain application and their structures are built in such a way that when a request gets submitted to them they process such request as a whole object without validating each and every line embedded in that Object.

Example: - My application has an object which will return my First Name, Last Name and Credit Balance in my account

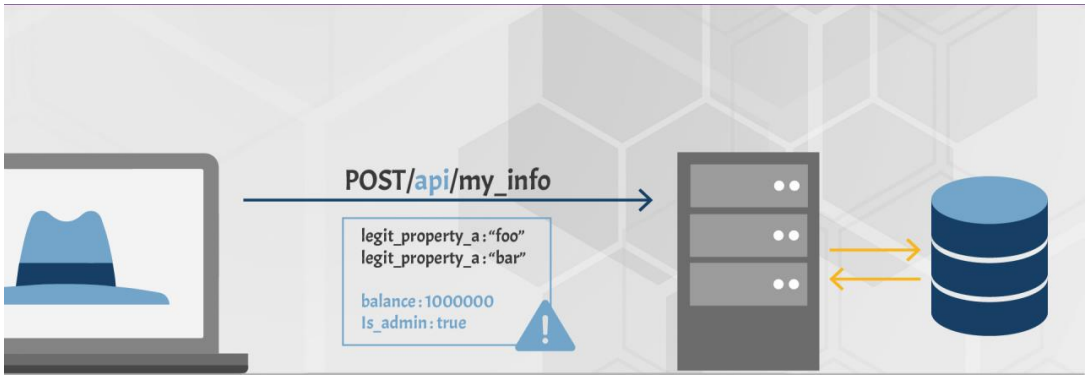
Benign Result should look like this after making a Get request to vulnerable end point

```
{"user_name":"inons","age":24,"credit_balance":10}
```

If an endpoint is vulnerable and we know processing happens at client side, I would use a proxy to catch the response, change it and replay it to make it like this

```
{"user_name":"attacker","age":60,"credit_balance":99999}
```

Data Validation is the main reason behind this attack which is not happening at backend and everything is getting treated as one object, too much trust.
Exploitation of mass assignment is easier in APIs, since by design they expose the underlying implementation of the application along with the properties' names



How it's related to API: -

An API endpoint is vulnerable if it automatically converts client parameters into internal object properties, without considering the sensitivity and the exposure level of these properties. This could allow an attacker to update object properties that they should not have access to.

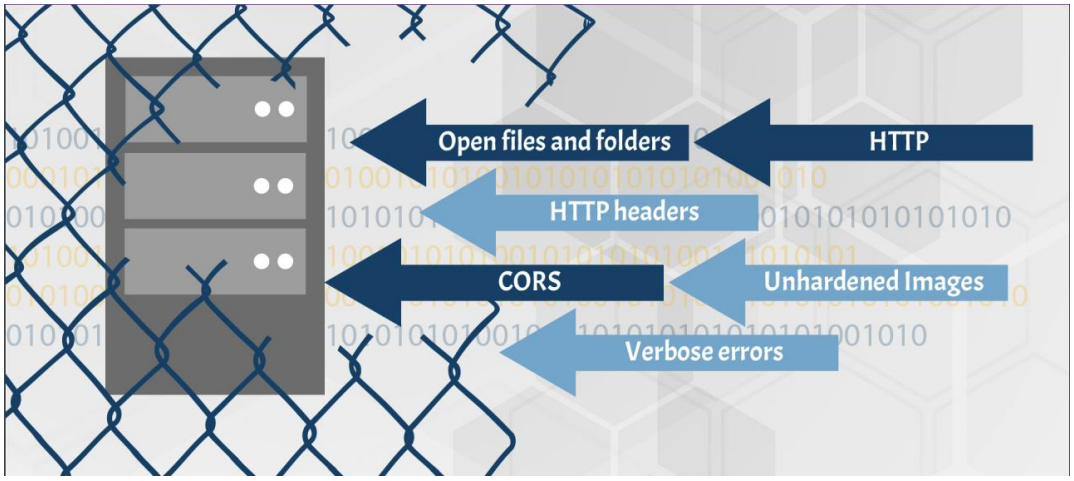
Mitigation: -

- Don't automatically bind incoming data provided by client to an internal code / object
- Explicitly define all the payload and parameters
- Read-only property for object schema should be set true and should never be modified

- Precisely define at design time the schemas, types, patterns you will accept in requests
- Use allow list and block list to reign on user input

7. **Security Misconfiguration: -**

I think this is not new to most of us in security world. Misconfiguration always is first and easiest method to get a foothold in any system, automatic tools would be able to detect it, fingerprinting would be able to reveal some info to dig more in that direction, unpatched systems, legacy endpoints, unprotected directories etc. It can happen at any layer of the stack under API or the system API is relying upon.



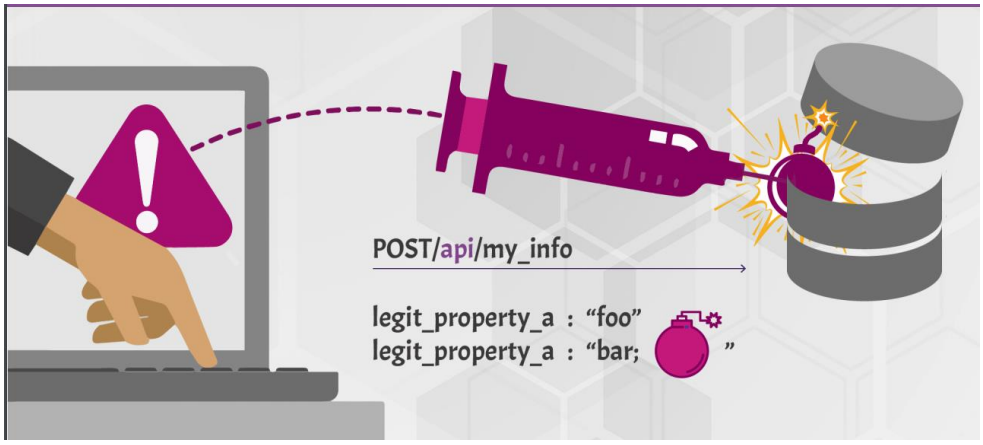
Mitigation: -

- Patching and Hardening
- Configuration scanners
- Disable features that are not being used
- Restrict administrative access
- Define and enforce all output and errors to not reveal sensitive information.

8. **Injection: -**

Under this vulnerability our usual suspects are SQL/LDAP/NoSQL injection attacks however this time an API call is the beacon for these attacks. Considering 80% of processing is happening at client side it becomes even easier to inject/craft such attacks. Attackers will leverage APIs to feed malicious input to the backend/database and wait to trigger unintended behaviours.

Scanners and fuzzers can be used to discover such flaws and after discovering attacker usually starts to weaponized their exploit and keeps on trying till succeeded.

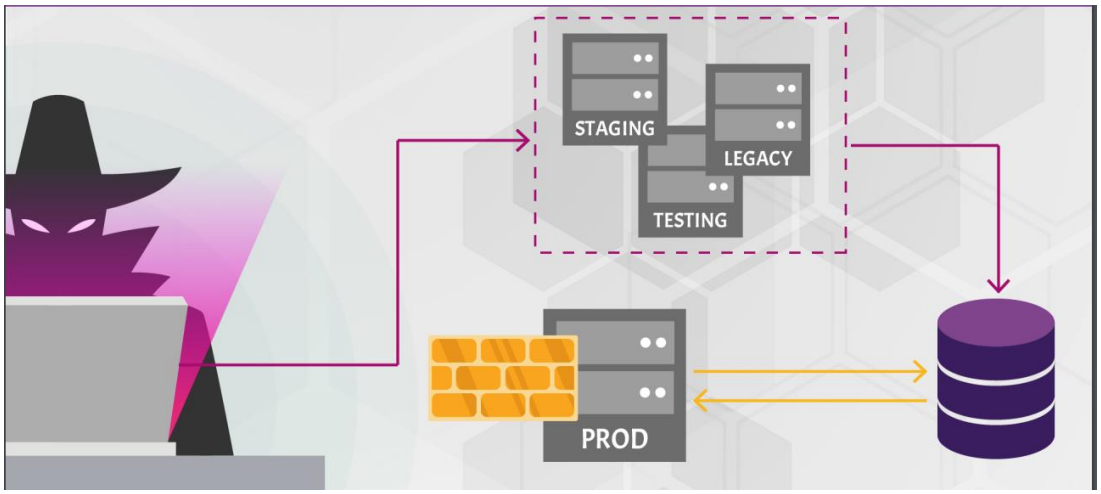


Mitigation: -

- Always validate and sanitize the input
- Strictly define input data schemas, types, string patterns, and char limit
- Enforce implicit limitation at run time
- Define, limit and enforce outputs to prevent data leak

9. **Improper Asset Management: -**

Organization can be very agile with their solution offerings. They keep adding new features and functionality while often keeping older versions of the same solution is running in Dev/Test/Staging. These systems often have access to production systems providing avenues of attack. Other scenarios include bad actors extracting the source code of an earlier version of current software which they can weaponize to get the hang of how the current application works.



Mitigation: -

- Inventory all API Hosts
- Limit access to anything that should not be public
- Limit access to production data, segregate access to production and non-production data
- Implement extra control i.e. API firewall
- Properly retire old version
- Implement strict authentication, redirects and CORS

10. **Insufficient Logging and Monitoring: -**

    It's more of a best practice and security hygiene and applicable to everything now a day. Logs hold the keys to citadel and must be always protected just like we protect our passwords. Logs contain sensitive, confidential, secretive and what not kinds of information so must be secured.

    Additionally, Logs carries proof that something bad/good happened. There is a term chain of custody which mean the sanity of logs must be maintained so that it can be used evidence in court so that bad actors can be penalized.

    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.

Mitigation or Hardening: -

- Log all failed authentication attempts, denied access, and input validation errors.
- Logs should be written using a format suited to be consumed by a log management solution and should include enough detail to identify the malicious actor.
- Logs should be handled as sensitive data, and their integrity should be guaranteed at rest and transit.
- Configure a monitoring system to continuously monitor the infrastructure, network, and the API functioning.
- Use a Security Information and Event Management (SIEM) system to aggregate and manage logs from all components of the API stack and hosts.
- Configure custom dashboards and alerts, enabling suspicious activities to be detected and responded to earlier.

Ref:-

1. API Security Articles, News, Vulnerabilities & Best Practices
2. api.gov.au