**REST API Security Best Practices**

Information that has value to the organization should be safeguarded. Security is not an afterthought, it has to be an intergral part of any application development program (SDLC Phases). This principle is also applicable for REST API's.

**REST API Security Design Principles**

Kindly find below eight design principles for securing information in computer systems, as described in the following sections:

1. **Least Privilege:** limiting access rights for users to the bare minimum permissions they need to perform their work
2. **Fail-Safe Defaults:**Unless a subject is given explicit access to a resource, it should be denied access to that resource
3. **Economy of Mechanism:**The design of the system should be as simple as possible. All the component interfaces and the interactions between them should be simple enough to understand.
4. **Complete Mediation:**A system should validate access rights to all its resources to ensure that they’re allowed and should not rely on the cached permission matrix. If the access level to a given resource is being revoked, but that isn’t reflected in the permission matrix, it would violate the security.
5. **Open Design:**Building a system in an open manner—with no secret, confidential algorithms.
6. **Separation of Privilege:**Granting permissions to a resource or an entity should not be purely based on a single condition, a combination of conditions based on the type of resource is a better idea.
7. **Least Common Mechanism:**Those mechanisms that are used to access resources should not be shared
8. **Psychological Acceptability:** Security mechanisms should not make the resource more difficult to access than if the security mechanisms were not present.

**Best Practices to Secure REST APIs**

Kindly find below the checklist I use for designing the security mechanism for REST APIs.

**Always Use HTTPS**

* By always using SSL, the authentication credentials can be simplified to a randomly generated access token that is delivered in the username field of HTTP Basic Auth. It’s relatively simple to use, and you get a lot of security features for free.
* If you use HTTP 2 to improve performance – you can even send multiple requests over a single connection, that way you avoid the complete TCP and SSL handshake overhead on later requests.

**Use Password Hash**

* Passwords must always be hashed to protect the system (or minimize the damage) even if it is compromised in some hacking attempts.
* e.g. PBKDF2, bcrypt and scrypt algorithms.

**Never expose information on URLs**

* Usernames, passwords, session tokens, and API keys should not appear in the URL, as this can be captured in web server logs, which makes them easily exploitable.

**Consider OAuth**

* Though basis auth is good enough for most of the APIs and if implemented correctly, it’s secure as well – yet you may want to consider OAuth as well.
* The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf.

**Consider Adding Timestamp in Request**

* Along with other request parameters, you may add a request timestamp as an HTTP custom header in API requests. The server will compare the current timestamp to the request timestamp and only accepts the request if it is within a reasonable timeframe (1-2 minutes, perhaps).
* This will prevent very basic replay attacks from people who are trying to brute force your system without changing this timestamp.

**Input Parameter Validation**

* Validate request parameters on the very first step, before it reaches to application logic. Put strong validation checks and reject the request immediately if validation fails.
* In API response, send relevant error messages and example of correct input format to improve user experience.

**Audit logs**

* Write audit logs before and after security related events.
* Consider logging token validation errors in order to detect attacks.
* Take care of log injection attacks by sanitising log data beforehand.

**Management endpoints**

* Avoid exposing management endpoints via Internet.
* If management endpoints must be accessible via the Internet, make sure that users must use a strong authentication mechanism, e.g. multi-factor.