
# COMP9321:
# Data services engineering

## Week 5: RESTful API Security

## (Quick Overview)

**Term 1, 2022**

**By Mortada Al-Banna, CSE UNSW**

# Key Aspects Security

"Security provided by IT Systems can be defined as the IT system's ability to being able to protect **confidentiality** and **integrity** of processed data, provide **availability** of the system and data, **accountability** for transactions processed, and **assurance** that the system will continue to perform to its design goals"

http://www.opensecurityarchitecture.org/cms/definitions/6-foundations/definitions/3-it-security

# Security Design Principles

- Least Privilege
- Fail-Safe Defaults
- Economy of Mechanism
- Complete Mediation
- Open Design
- Separation Privilege
- Least Common Mechanism
- Psychological Acceptability
- Defense in Depth

http://web.mit.edu/Saltzer/www/publications/protection/

# Security Design Principles

- **Least privilege**: Every program and every user of the system should operate using the least set of privileges necessary to complete the job.
- **Fail-safe defaults**: Base access decisions on permission rather than exclusion. This principle means that the default situation is lack of access, and the protection scheme identifies conditions under which access is permitted.
- **Economy of mechanism**: Keep the design as simple and small as possible. This well-known principle applies to any aspect of a system

http://web.mit.edu/Saltzer/www/publications/protection/

# Security Design Principles

- **Complete mediation**: Every access to every object must be checked for authority. This principle implies that a foolproof method of identifying the source of every request must be devised.
- **Open design**: The design should not be secret. The mechanisms should not depend on the ignorance of potential attackers, but rather on the possession of specific, more easily protected, keys or passwords.
- **Separation of privilege**: Where feasible, a protection mechanism that requires two keys to unlock it is more robust and flexible than one that allows access to the presenter of only a single key.

http://web.mit.edu/Saltzer/www/publications/protection/

# Security Design Principles

- **Least common mechanism**: Minimize the amount of mechanism common to more than one user and depended on by all users.
- **Psychological acceptability**: It is essential that the human interface be designed for ease of use, so that users routinely and automatically apply the protection mechanisms correctly.
- **Defense in Depth**: an approach in which a series of defensive mechanisms are layered in order to protect valuable data and information. If one mechanism fails, another steps up immediately to thwart an attack.

http://web.mit.edu/Saltzer/www/publications/protection/

# REST API Security…Does it matter?

# REST API Security Matters

- Major security breaches happened due to unsecure/unprotected API (e.g., Venmo, Equifax, Impreva)
- It matter enough that OWASP included many instances in their web security **Top ten** related to APIs and they have the **REST Security cheat** sheet.
- REST relies on the elements of the Web for security too (Check OWASP top 10)

https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf
https://www.owasp.org/index.php/REST_Security_Cheat_Sheet

# HTTPS (TLS)

- "Strong" server authentication, confidentiality and integrity protection The only feasible way to secure against man-in-the-middle attacks
- Any security sensitive information in REST API should use TLS (formerly known as SSL)

See the OWASP Transport Layer Protection Cheat Sheet



https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html

# REST APIs Authentication

API developers at least must deal with authentication and authorisation:

Authentication (401 Unauthorized) vs. Authorisation (403 Forbidden):

Common API authentication options:

- HTTP Basic (and Digest) Authentication: IETF RFC 2617
- Token-based Authentication
- API Key [+ Signature]
- OAuth (Open Authorisation) Protocol - strictly uses HTTP protocol elements only



https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet

# Authentication

The basic idea revolves around: ”login credentials” (for app, not human)

The questions: (i) what would the credentials look like and how would you pass them around “safely”? (ii) how to ensure stateless API interactions? (no ‘session’)

HTTP Basic Authentication Protocol (HTTP Specification)

Initial HTTP request to protected resource

```
GET /secret.html HTTP/1.1
Host: example.org
```

Server responds with

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm=“ProtectedArea”
```

Client resubmits request

```
GET /secret.html HTTP/1.1 Host: example.org
Authorization: Basic Qm9iCnBhc3N3b3JkCg==
```

Further requests with same or deeper path can include the additional Authorization header pre-emptively

# HTTP Basic Auth

Issues with HTTP Basic Auth as an API authentication scheme

- The password is sent over the network in base64 encoding - which can be converted back to plain text
- The password is sent repeatedly, for each request - larger attack window
- HTTP Basic Auth combined with TLS could work for some situations … But normally this scheme is not recommended and considered not secure “enough”

# What else you need to know about in regard to REST API security?

- Rate limiting
- Restrict HTTP methods
- Input validation
- Sensitive information in HTTP requests
- Audit logs
- Send Appropriate Error logs

# Rate Limiting

- Prevent farming (manage cost)
- Control possible DDoS attacks
- API keys are useful in this regard:
  - Require API keys for every request to the protected endpoint.
  - Return 429 Too Many Requests HTTP response code if requests are coming in too quickly.
  - Revoke the API key if the client violates the usage agreement.

# Restrict HTTP methods

- Do Consumers need to use all HTTP methods?
- Apply a whitelist of permitted HTTP Methods e.g. GET, POST, PUT.
- Reject all requests not matching the whitelist with HTTP response code 405 Method not allowed.
- Make sure the caller is authorized to use the incoming HTTP method on the resource collection, action, and record

# Input Validation

- Do not trust input parameters/objects.
- Validate input: length / range / format and type.
- Achieve an implicit input validation by using strong types like numbers, booleans, dates, times or fixed data ranges in API parameters.
- Constrain string inputs with regex.
- Reject unexpected/illegal content.
- Make use of validation/sanitation libraries or frameworks in Python (e.g., Validator Collection)

## Sensitive information in HTTP requests

- Sensitive information should not be appearing in the URL
  - In POST/PUT requests sensitive data should be transferred in the request body or request headers.
  - In GET requests sensitive data should be transferred in an HTTP Header.
- Example:

https://example.com/controller/123/action?apiKey=a53f435643de32

## Audit logs

- Nothing is 100% secure
- Logs help detect quickly if there is an incident
- 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 sanitizing log data beforehand.

# Send Appropriate Error logs

- Use the semantically appropriate status code for the response (remember the API design lecture)
- Respond with generic error messages - avoid revealing details of the failure unnecessarily.
- Do not pass technical details (e.g. call stacks or other internal hints) to the client.

# Q&A