API testing information gathering

* Finding input data with compulsory and optional field
* Type of requests and supported methods like http or https including media types
  + Different type of requests are GET, PUT, POST, DELETE, PATCH
  + Same endpoint with different method can do different operation on the resource
  + Content-Type Header can be changed between json and XML to check for errors
    - application/json
    - application/xml

How does this helps?

* + Identify hidden endpoint by changing the common word in API, For example, you could add a payload to the /update position of the path with a list of other common functions, such as delete and add
    - Create the word list from the API URLs only
    - Use Param miner to get 65K names per request
    - Content discovery tool ???
  + Finding hidden parameter:
    - BurpIntruder to find out different hidden parameters
    - <https://portswigger.net/web-security/learning-paths/api-testing/api-testing-finding-hidden-parameters/api-testing/finding-hidden-parameters>
  + Mass Assignment:
    - Unintended binding of param like isAdmin
    - Hidden parameter if know also get binded.
* Rate limit and authentication mechanism
* Review documentation if available at some endpoint or anywhere hosted at docs
* Find all the parent endpoint to figure out using **intruder**
  + /api/swagger/v2
  + /api/swagger/
  + /api
    - If some request with removing some prefix still gives error but not 404 that means there are lot of endpoints available at that.
    - 302 means redirect
    - 400 means error occurred
* Server Side parameter pollution: calling internal API that is not directly accessible from internet but depends on the user input as given following
  + query parameters, form fields, headers, and URL path parameters may all be vulnerable.
  + Place #, &, and = : to check the vulnerabilities in query parameters, to check if the truncation is possible and if they are possible then they are vulnerable
    - Encode these parameters :
      * %23 for # FOR IGNORING THE REST OF QUERY PARAM
      * **&** (ampersand) is encoded as **%26** FOR ADDING EXTRA AGRUGMENT IF THE OUTPUT IS UNCHANGED THEN INJECTION WAS SUCCESSFUL
      * **Overwrite the existing parameter with the same name** GET /userSearch?name=peter%26name=carlos&back=/home
        + PHP parses the last parameter only. This would result in a user search for carlos.
        + ASP.NET combines both parameters. This would result in a user search for peter,carlos, which might result in an Invalid username error message.
        + Node.js / express parses the first parameter only. This would result in a user search for peter, giving an unchanged result.

 **=** (equal sign) is encoded as **%3D**.

* Testing for server-side parameter pollution in REST paths
  + Check for path traversal where parameters are encoded in paths instead of query parameter
  + To check user1/../user2 if resolves to /user2 then it’s vulnerable
  + Use #, & and = for unexpected result
* Testing for server-side parameter pollution in structured data format
  + POST /myaccount name=peter # request in browser
  + PATCH /users/7312/update {"name":"peter"} # request in server side for updating the peter profile
  + POST /myaccount name=peter","access\_level":"administrator # To check for the inspection
* JS link Finder which extracts all the links present from the js files also
  + <https://portswigger.net/bappstore/0e61c786db0c4ac787a08c4516d52ccf>
  + <https://github.com/portswigger>

Ref for the data available:

<https://github.com/antichown/burp-payloads/tree/master>

Detailed automation: <https://portswigger.net/web-security/learning-paths/api-testing/api-testing-testing-with-automated-tools/api-testing/server-side-parameter-pollution/testing-with-automated-tools>