REST

Web-based APIs
REST

- Representational State Transfer
- Style of web software architecture that simplifies application
- Not a standard, but a design pattern
REST

- Take all resources for web application (data, files, functions)
  - Identify each resource and action on resource via an HTTP method and URL.
  - Method selects action
  - Send arguments via the HTTP request (e.g. in URL, URL parameters, or payload)
REST toy example

- http://foo.com/bar/users
  - Server foo.com
  - Database bar
  - Table users
  - URL returns table users in database bar in a particular format (XML, JSON)

- Common examples
  - Twitter, Flickr, Amazon
REST and HTTP methods

- HTTP request methods indicate the desired action
- GET
  - Requests a representation of the specified resource.
  - Use for operations that have NO side-effects (safe operations)
  - Works with robots and crawlers.
- POST
  - Submits data to be processed (e.g., from an HTML form) to the identified resource. Data is included in the body of the request.
- PUT
  - Uploads a representation of the specified resource.
- DELETE
  - Deletes the specified resource.
REST and security

- Each API call must ensure request is authenticated and authorized
  - Requires attention to many of the OWASP Top 10
    - A4: Insecure Direct Object Access
    - A7: Missing Function Level Access Control
    - A2: Broken Authentication and Session Management
    - A1: Injection
  - Now in OWASP Top 10 for 2017 draft

<table>
<thead>
<tr>
<th>Top 10 2013</th>
<th>Top 10 2017</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1 – Injection</td>
<td>A1 – Injection</td>
</tr>
<tr>
<td>A2 – Broken Authentication and Session Management</td>
<td>A2 – Broken Authentication and Session Management</td>
</tr>
<tr>
<td>A3 – Cross-Site Scripting (XSS)</td>
<td>A3 – Cross-Site Scripting</td>
</tr>
<tr>
<td>A4 – Insecure Direct Object References</td>
<td>A4 – Broken Access Control</td>
</tr>
<tr>
<td>A5 – Security Misconfiguration</td>
<td>A5 – Security Misconfiguration</td>
</tr>
<tr>
<td>A6 – Sensitive Data Exposure</td>
<td>A6 – Sensitive Data Exposure</td>
</tr>
<tr>
<td>A7 – Missing Function Level Access Control</td>
<td>A7 – Insufficient Attack Protection</td>
</tr>
<tr>
<td>A8 – Cross-site Request Forgery (CSRF)</td>
<td>A8 - Cross-site Request Forgery (CSRF)</td>
</tr>
<tr>
<td>A9 – Using Components with Known Vulnerabilities</td>
<td>A9 – Using Components with Known Vulnerabilities</td>
</tr>
<tr>
<td>A10 – Unvalidated Redirects and Forwards</td>
<td>A10 – Unprotected APIs</td>
</tr>
</tbody>
</table>
JSON
JSON

- JavaScript Object Notation
  - De-facto web object data format
    - Subset of JavaScript
    - Minimal, lightweight, text-based syntax
    - Easy to parse and generate
  - Prevalent in most web sites
  - Prevalent in many web APIs, often as part of a REST architecture
  - Designed to enable stateful, real-time communication between browser and web application
    - Often used to allow web server to directly modify elements of a page without refresh
    - Initially AJAX (Asynchronous JavaScript and XML) where XML exchanged (e.g. homework site)
    - Now mostly ‘AJAJ’ where JSON exchanged instead
JSON objects

- Objects are unordered containers of key/value pairs
  - Keys are strings
  - Values are JSON values
  - Wrapped in `{ }`
    - , separates key/value pairs
    - : separates keys and values
- Parsed into local data structures as struct, record, hashtable, or dictionary
JSON Values

- **Strings**
  - Sequence of 0 or more Unicode characters wrapped in double quotes
- **Numbers**
  - Integer, Real, Scientific
  - No octal or hex
  - No NaN or Infinity (Uses null instead!)
- **Booleans**
  - true, false
  - null
  - A value that isn't anything
- **Objects**
- **Arrays**
Arrays

- Ordered sequences of values wrapped in 
  - , separates values
- JSON does not specify indexing.
  - Array is parsed by web program language
  - Implementation can start array indexing at 0 or 1.

```
["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"]
```

```
[
  [0, -1, 0],
  [1, 0, 0],
  [0, 0, 1]
]
```
JSON example

```json
{
    "firstName": "John",
    "lastName": "Smith",
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": 10021
    },
    "phoneNumbers": [ "212 555-1234", "646 555-4567"
    ]
}
```
JSON example

- **stockfighter.io stock order**

  ```json
  {
      'account': 'SWB1886430',
      'venue': 'ETKBEX',
      'symbol': 'EJYW',
      'price': 8100,
      'qty': 100,
      'direction': 'buy',
      'orderType': 'limit'
  }
  ```

- **Twitter direct message**
  - [https://dev.twitter.com/rest/reference/get/direct_messages](https://dev.twitter.com/rest/reference/get/direct_messages)
JSON in AJAX & JavaScript

- JSON often exchanged in JavaScript via XMLHttpRequest
- Example: obtain as `responseText`, then parse it

```javascript
responseText is '{ "name": "Jack B. Nimble", "at large": true, "grade": "A", "format": { "type": "rect", "width": 1920, "height": 1080, "interlace": false, "framerate": 24 } }';

jsonobject = JSON.parse(responseText);
document.write("The object<br>");
document.write("name: ", jsonobject.name, "<br>");
document.write("grade: ", jsonobject.grade, "<br>");
document.write("format: ", jsonobject.format, "<br>");
```
JSON and avoiding eval()

- JSON uses JavaScript syntax to specify objects in a serialized manner
- Can either write a parser to pull out key:value pairs from JSON string or simply “evaluate” JSON string via eval
  - Parse version
    
    ```javascript
    jsonobject = JSON.parse(responseText);
    ```
  - Eval version
    
    ```javascript
    jsonobject = eval('(' + responseText + ')');
    ```
- Which one is safer?
- What if JSON object contained rogue JavaScript code?
  - Deserialization attacks
  - Mixing code and data
JSON security

- Deserialization attacks
  - Dependent upon trust
  - On client, not an issue
    - JSON data came from the same server that vended the page.
    - `eval` of the data is no less secure than the original html (assuming sent over HTTPS)
JSON security

- What about on the server (i.e. Node.js)?
- Is it OK to ever use `eval` to generate object from client?
  - No
- Can never trust the client
  - The client cannot be trusted
  - Server must validate everything the client tells it.
  - Run-time evaluation of untrusted input extremely dangerous!
- Always use a parser on server running JavaScript (nodejs)
  - `JSON.parse(string)` instead of `eval`. 
eval is evil

- Avoid using it in your web applications
  - PHP eval and deserialization issues (picoCTF, natas)
  - Python eval issues (picoCTF)
  - JavaScript eval issues (Pentestlab exercises, deserialization)
# Security of JSON vs. XML

<table>
<thead>
<tr>
<th>JSON</th>
<th>XML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Structure</td>
<td>Data Structure</td>
</tr>
<tr>
<td>No validation system</td>
<td>XSD</td>
</tr>
<tr>
<td>No namespaces</td>
<td>Has namespaces (can use multiples)</td>
</tr>
<tr>
<td>Parsing is just an eval</td>
<td>Parsing requires XML document parsing</td>
</tr>
<tr>
<td>Fast</td>
<td>using things like XPath</td>
</tr>
<tr>
<td>Security issues</td>
<td></td>
</tr>
<tr>
<td>In JavaScript you can work with objects</td>
<td>In JavaScript you can work with strings</td>
</tr>
<tr>
<td>– runtime evaluation of types</td>
<td>– may require additional parsing</td>
</tr>
<tr>
<td>Security: eval() means that if the source</td>
<td>Security: XML is text/parsing – not</td>
</tr>
<tr>
<td>is not trusted anything could be put into it.</td>
<td>code execution.</td>
</tr>
<tr>
<td>Libraries exist to make parsing safe(r)</td>
<td></td>
</tr>
</tbody>
</table>
JSON vs Javascript

- Double quotes for strings
- No functions intended to be allowed (text, no code unless someone does an eval)
- No circular references (text, no references)
Questions

- https://sayat.me/wu4f