



OWASP

Open Web Application
Security Project

OWASP Portland Chapter

Welcome to OWASP Portland's
monthly chapter meeting!



OWASP

Open Web Application
Security Project

Twitter: @PortlandOWASP

Website: pdxowasp.org

Meetup: <https://www.meetup.com/OWASP-Portland-Chapter/>

Slack: [#chapter-pdx](https://owasp.slack.com)



OWASP

Open Web Application
Security Project

May Chapter Meeting

Leadership Panel

Recruiting, Interviewing, and Hiring Q&A

Tuesday May 14th, 2019 from 6–8pm

Hosted by: Zapproved 1414 NW Northrup St #700, Portland

<https://www.meetup.com/OWASP-Portland-Chapter/events/260486258/>



OWASP

Open Web Application
Security Project

Project Focus: ASVS 4.0

- Application Security Standard for Web Apps and Services of All Types
- Covers requirements for addressing OWASP Top 10 2017
- Three Verification Levels
 1. Low Assurance Level, Completely Penetration Testable
 2. Applications Containing Sensitive Data - Recommended For Most Apps
 3. Critical Applications Performing High Value Transactions

<https://github.com/OWASP/ASVS>



OWASP

Open Web Application
Security Project

Become A Member

- Support Web Application Security Education
- Get involved in OWASP Projects
- Direct Your Membership Dues to Your Home Chapter or Project You Care About

<https://www.owasp.org/index.php/Membership>

OWASP TOP 10 For JavaScript Developers

OWASP Portland

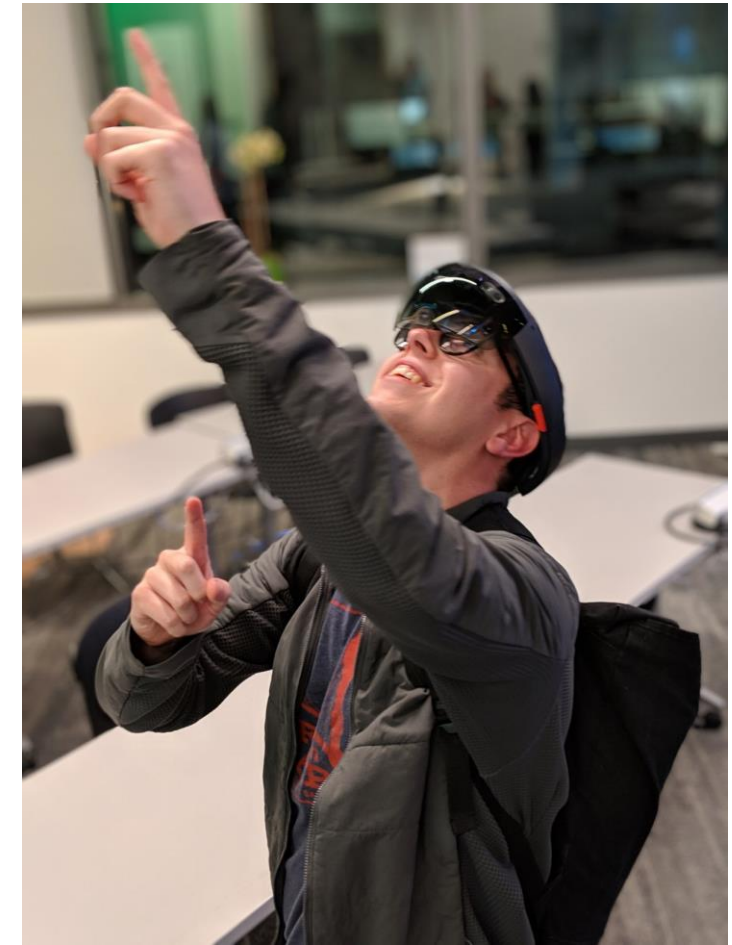
Lewis Arden

April 10, 2019

<https://twitter.com/LewisArden>

About Me

- Sr. Security Consultant @ Synopsys Software Integrity Group (SIG)
 - Formerly Cigital
- Prior to Cigital
 - B.Sc. in Computer Security and Ethical Hacking
 - Founder of the Leeds Ethical Hacking Society
 - Software Developer
 - Security Consultant
- Synopsys
 - Historically all about hardware
 - SIG formed to tackle software
 - Team consisting of well-known organizations
 - BlackDuck
 - Coverity
 - Codenomicon
 - Cigital
 - Codiscope



SYNOPSYS®

What is the OWASP Top 10?

OWASP Top 10 2017	
A1	Injection
A2	Broken Authentication
A3	Sensitive Data Exposure
A4	XML External Entities (XXE)
A5	Broken Access Control
A6	Security Misconfiguration
A7	Cross-site Scripting
A8	Insecure Deserialization
A9	Using Components with Known Vulnerabilities
A10	Insufficient Logging and Monitoring

- 10 critical web application security risks
- Common flaws and weaknesses
- Present in nearly all applications

Modern, evidence-based risks. Data covers 2014-2017:

- 114,000 apps
- 9000 bug bounties
- 40 security consultancies and 1 bug bounty firm
- 50+ CWEs accepted in raw data

Community-chosen risks

- 500 survey responses

Why is the OWASP Top 10 Important?

Unofficial Web Application Security Standard

Specific

Relates to specific weaknesses with a common understanding and taxonomy

Measurable

Common metric, allowing year-on-year comparisons between security vendors

Actionable

Documents not just flaws and defects, but also solutions

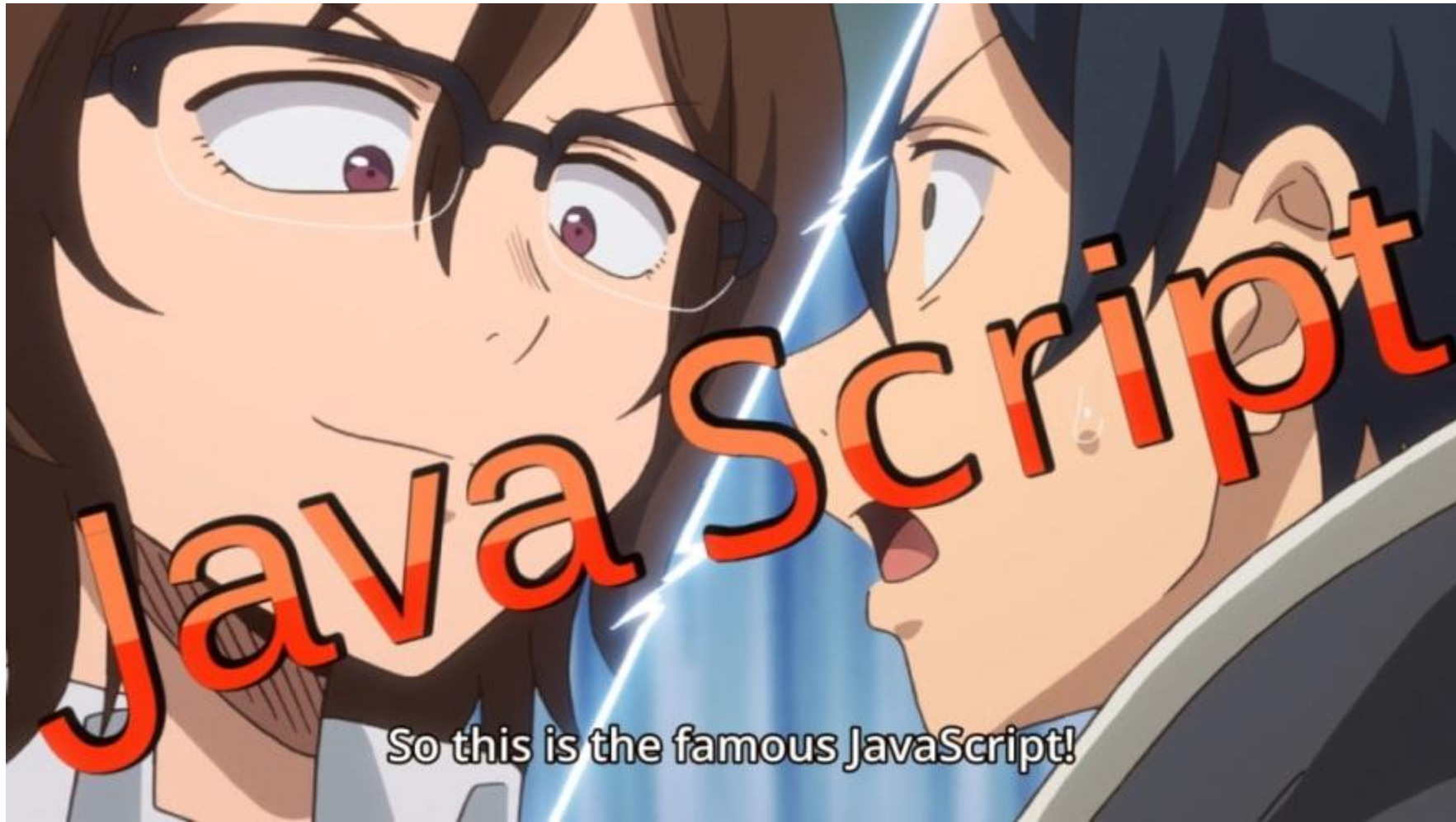
Realistic

Just 10 weaknesses, approachable, and scalable

Timely

Regularly updated every 2-3 years using latest data

「JavaScript」



A1:2017 Injection

The Dangers of Mixing Data and Code

Common Injection Issues

Purple – Covered today

SQL injection (SQLi)

NoSQL Injection

XML/XPath injection

HTTP header injection

Log injection

Command injection

LDAP injection



A1
:2017

Note: This is a non-exhaustive list.

SQL Injection


It exists in the Node ecosystem



```
var mysql = require('mysql');  
// ... snip ...  
const obj = JSON.parse(req.body)  
  
connection.query(  
  "SELECT * from users where userid='" + obj.user + "'",  
  function (err, results) {  
    console.log(results);  
  }  
);
```

SQL Injection - Prevention

Parameterized queries to the rescue



```
var mysql = require('mysql');  
// ... snip ...  
const obj = JSON.parse(req.body)  
  
connection.query(  
  'SELECT * from users where userid=?', [obj.user],  
  function (err, results) {  
    console.log(results);  
  }  
);
```



```
connection.query(  
  "SELECT * from users where userid='" + obj.user + "'",  
  function (err, results) {  
    console.log(results);  
  }  
);
```



```
connection.query(  
  'SELECT * from users where userid=?', [obj.user],  
  function (err, results) {  
    console.log(results);  
  }  
);
```

MongoDB Injection Attacks

Mongo says no SQL Injection:

How does MongoDB address SQL or Query injection?

BSON

As a client program assembles a query in MongoDB, it builds a BSON object, not a string. Thus traditional SQL injection attacks are not a problem. More details and some nuances are covered below.

<https://docs.mongodb.com/manual/faq/fundamentals/#how-does-mongodb-address-sql-or-query-injection>

No SQL Injection != NoSQL Injection

How Does The Attack Work?

User input includes a [Mongo Query Selector](#) such a:


\$ne, \$lt, \$gt, \$eq, \$regex, etc

User input is directly included into a collection [method](#) as part of the query such as:

find, findOne, findOneAndUpdate, etc

Source: req.query

Sink: findOne



```
db.collection('collection').findOne({  
  username: req.query.user,  
  password: req.query.pass,  
  isActive: true  
})
```

Vulnerable Login Example

```
db.collection('collection').findOne({
  username: req.query.user,
  password: req.query.pass,
  isActive: true
}, function (err, result) {
  if (err) {
    console.log('Query error...');
    return err;
  }
  if (result !== null) {
    req.session.authenticated = true;
    res.redirect('/');
  }
  else
    res.redirect('/login?user=' + user);
});
```

Injection:

[https://url.to/login?user=admin&pass\[\\$ne\]=](https://url.to/login?user=admin&pass[$ne]=)

Query Output:

```
db.collection('collection').findOne({
  username: "admin",
  password: {
    $ne: "",
  },
  isActive: true
})
```


Demo

MongoDB Injection - <https://github.com/dbohannon/MEANBug>

MongoDB Injection Prevention

Always validate and sanitize user input!

Ensure user-input is a [String](#) inside a collection method



```
db.collection('collection').findOne({  
    username: String(req.query.user),  
    password: String(req.query.pass),  
    isActive: true  
})
```


A2:2017 Broken Authentication

Broken Authentication and Session Management

Insecure Object Comparisons

What happens if you create your own Authentication middleware?



```
const SESSIONS = {}

const mustBeAuthenticated = (req, res, next) => {
  if(req.cookies) {
    const token = req.cookies.token

    if(token && SESSIONS[token]) {
      //allow it
      next()
    }
  }
  res.send('not authorized!')
}
```

Comparison Table

Value	Return
SESSIONS[' <i>invalidString</i> ']	False
SESSIONS['']	False
SESSIONS[' <i>constructor</i> ']	True
SESSIONS[' <i>hasOwnProperty</i> ']	True

What Happens When You Create an Object in JavaScript?



```
const SESSIONS = {}
```

```
__proto__:
```

```
  constructor: f Object()
```

```
  hasOwnProperty: f hasOwnProperty()
```

```
  isPrototypeOf: f isPrototypeOf()
```

```
  [...]
```

```
SESSIONS['constructor'] === SESSIONS.constructor //returns true
```


Exploit

This issue is trivial to exploit.

Using cURL we can simply run the following command:

- `curl https://localhost:9000 -H "Cookie: token=constructor"`

Alternatively, you can just set the *document.cookie* value via the browser.

Demo

Insecure Object Comparisons

How Do We Correctly Check?

ES6 Map



```
SESSIONS.has('__proto__')  
  
// false  
  
SESSIONS.has('validString')  
  
// true
```

Object Comparisons



```
SESSIONS.hasOwnProperty('__proto__')  
  
// false  
  
SESSIONS.hasOwnProperty('validString')  
  
// true
```

A3:2017 Sensitive Data Exposure

RESTful API Data Leakage

Its easy to write the following code:



```
try { /*stuff*/ } catch(e) { return e; }
```

But what can happen on exposed routes?



```
{"user": {"id": "DB2 SQL-Error: -104 ILLEGAL SYMBOL SOME SYMBOLS THAT MIGHT BE LEGAL ARE USED AFTER  
'USERID' COLUMN."}}
```

Do Not Include Verbose Error Messages in JSON

Verbose error messages lead to system information disclosure.

Although the client-side application displays a generic error message, the JSON response might still contain full error messages.


Malicious users may use a web proxy to read the stack trace output in JSON.

Caution: Detailed system information might not seem that significant at first sight. However, it can inform attackers on the internals of the system or infrastructure, and help them drive further attacks.

Disclosing Information via Error Messages in JSON

This code shows an SQL error message passed in JSON and the JavaScript code used to mask it.

JavaScript code to handle the error:



```
connection.query('SELECT * from users where userid=?', [obj.user], (err, results) => {  
  if (err) {  
    logger.log(`SQL Error:  ${err}`);  
    return 'Could not complete query';  
  }  
  return results;  
})
```


A4:2017 XML External Entities (XXE)

XML

eXtensible Markup Language

Markup language

(similar to HTML) for storing data

- Element
- Attribute

Often used to exchange data
between systems, e.g., as SOAP
messages in web services

Processed by an XML parser (examples):

- [libxmljs](#) – 42,000 weekly downloads
- [node-expat](#) – 58,000 weekly downloads



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<myelement myattribute="1">
  <data>somedata...</data>
</myelement>
<hello> OWASP Portland! </hello>
```

XML External Entities (XXE) Injection

XML External Entity (XXE) injection is when an attacker injects a malicious external entity definition to take control of what the XML parser tries to resolve.

Using external entity resolution, attackers can:

- Access local or remote files and services
- Port scan internal or adjacent hosts
- Perform denial of service (DoS)

Attacker Controlled

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE hello [
  <!ELEMENT hello ANY >
  <!-- ENTITY portland SYSTEM
        "https://attacker.com/t.dtd" -->
-->
  <myelement myattribute="1">
    <data>somedata...</data>
  </myelement>
  <hello>&portland;</hello>
```

Local File Inclusion (XXE)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [ <!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<myelement myattribute="1">
  <data>somedata...</data>
</myelement>
<hello> Portland &xxe; </hello>
```

Portland
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr
/sbin/nologin bin:x:2:2.....

XML External Entities (XXE) Vulnerable Example


Libxmljs can be vulnerable to XXE

User-input:
req.files

```
module.exports.bulkProducts = function (req, res) {  
  if (req.files.products && req.files.products.mimetype == 'text/xml') {  
    var products = libxmljs.parseXmlString(req.files.products.data.toString('utf8'),  
      { noent: true, no blanks: true })  
    products.root().childNodes().forEach(product => {  
      var newProduct = new db.Product()  
      // ..snip..  
      newProduct.description = product.childNodes()[3].text()  
      newProduct.save()  
    })  
    res.redirect('/app/products')  
  } // ...snip...  
}
```

Misconfiguration:
noent: true

XML Entity Expansion (Billion Laughs)



```
<?xml version="1.0"?>
<!DOCTYPE lolz [
  <!ENTITY lol "lol">
  <!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
  <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
  <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
  <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
  <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
  <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
  <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
  <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
  <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

XML Entity Expansion (Billion Laughs)

Third-party parsing libraries

node-expat

- Vulnerable by default
- No way to configure parser to disable
- <https://help.semmle.com/wiki/display/JS/XML+internal+entity+expansion>

libxmljs

- Vulnerable if noent is set to true
- <https://help.semmle.com/wiki/display/JS/XML+external+entity+expansion>



```
var expat = require('node-expat')
var parser = new expat.Parser('UTF-8')

parser.on('text', function (text) {
  console.log(text)
})

parser.write(xmlSrc)
```



```
const libxml = require('libxmljs');
var doc = libxml.parseXml(xmlSrc, { noent: true });
```


Demo

XXE

XML Injection Prevention

Use libraries with safe defaults, such as libxmljs (apart from its sax parser)

If entities do need to be expanded use [lodash](#), [underscore](#), or [he](#)

If neither can be used, strict input validation/output encoding must be performed before parsing

A5:2017 Broken Access Control

Do Not Rely on Client-Side Controls

Client-side routing and authorization should only be implemented for user experience.

Authentication and authorization controls implemented client-side can be bypassed.

All authorization, authentication, and business logic controls must be enforced server-side.

Caution: Never trust the client!



Angular Example

[Angular Route Guards are for Boolean display aesthetics](#)



```
import { Injectable } from '@angular/core';
import { Router, CanActivate } from '@angular/router';

@Injectable()
export class AuthGuardService implements CanActivate {

  constructor(public router: Router) {}

  canActivate(): boolean {
    alert("Unauthorized! Only administrators are allowed.");
    return false;
  }
}
```

Demo

Bypassing Angular Route Guards

Authorization Checks



```
app.get('/api/users/:id', auth.requiresRole('admin'), admin.getUserById);  
app.get('/api/users/edit/:id', auth.requiresRole('admin'));  
app.put('/api/users/:id', auth.requiresRole('admin'), admin.updateUser);  
app.delete('/api/users/:id', auth.requiresRole('admin'), admin.deleteUser);  
app.post('/api/admin/logs', auth.requiresRole('admin'), admin.checkTmpFolder);
```

Implement proper checks on API endpoints

node-casbin - <https://github.com/casbin/node-casbin>

passport - <https://github.com/jaredhanson/passport>

auth0 - <https://github.com/auth0/passport-auth0>

A6:2017 Security Misconfiguration

Ensure Secure Practices for Session Cookies

Various security restrictions can be applied to cookies:

- *Secure*: Prohibits a cookie from being transported over an unsecured (non-SSL) connection.
- *HttpOnly*: Prohibits client-side code from accessing the cookie. As a result, even if a cross-site scripting (XSS) flaw exists, the browser will not reveal the cookie to a 3rd party.

Caution: Accessing cookies using *document.cookie* prevents you from marking them as *HttpOnly* and therefore exposes them to hijacking. This is particularly important for sensitive cookies such as session IDs.

Ensure Secure Practices for Session Cookies

SameSite: Prevents a cookie from being sent with requests from another domain to avoid Cross-Site Request Forgery (CSRF) attacks

- Attributes

- Lax: Can use it for some requests such as 'GET', but not 'action' requests such as 'POST'
- Strict: The cookie will not get attached to any cross-site requests

Cookie Prefixes: A prefix on a cookie name defines that a cookie should have certain attributes included

- `__Secure`: Informs the browser that the *Secure* attribute must be set
- `__Host`: Informs the browser that both the *Path* and *Secure* attribute must be set

Caution: *SameSite* flag and *Cookie Prefixes* are not adopted by all browsers.

Using *express-session* Securely

Default configuration of *express-session* is **not** secure

- Sets *httpOnly* flag to true, but *Secure* flag to false
- Uses unsafe memory-backed store

Insecure:

```
session = require('express-session')
app.use(session({ secret: 'secret123' }));
```

Note: Explicitly set the cookie options when initializing the module.

Secure:

```
session = require('express-session')
app.use(session({
  secret: config.SESSION_SECRET,
  name: '__Host-auth-cookie',
  cookie: { httpOnly: true, secure: true, sameSite: 'strict', path: '/' },
  store: new MySQLStore({}, sqlConnection)
}));
```

Using *cookie-session* Securely

Avoid placing sensitive user data within *req.session*.

- Data is stored client-side unencrypted and can easily be viewed.

Cookie-backed stores that contain no timestamps inherently do **not** expire.

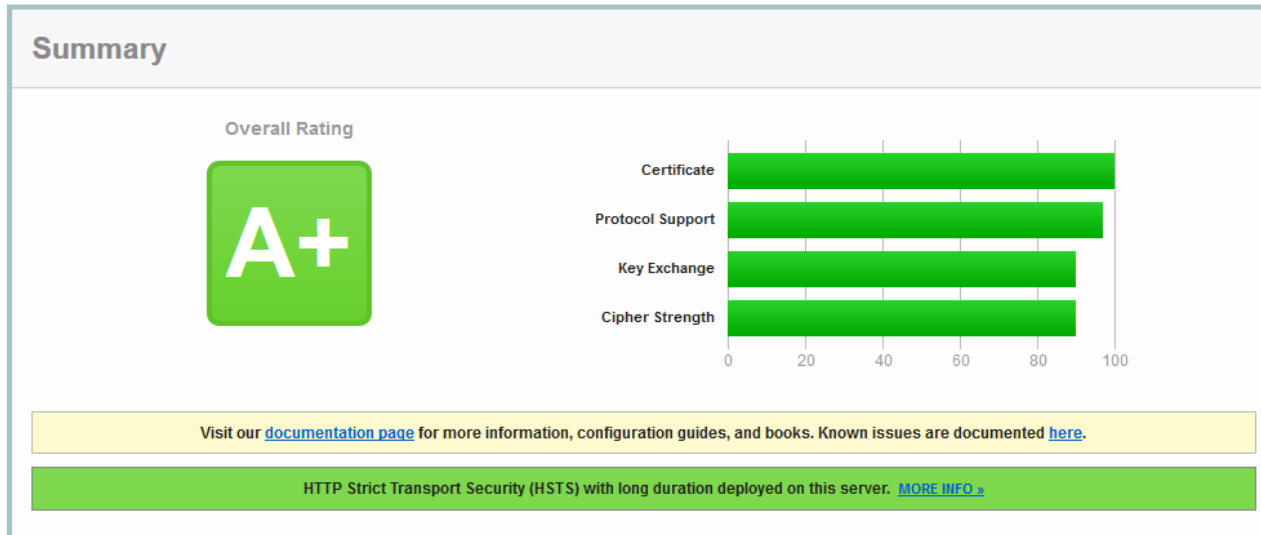
Cookie replay attacks are possible if state data is stored within *req.session*.

Online Tools for Testing Security Configuration

SSL Report: www.synopsys.com (184.86.122.144)

Assessed on: Wed, 05 Sep 2018 17:01:43 UTC | [Hide](#) | [Clear cache](#)

[Scan Another »](#)



<https://www.ssllabs.com/ssltest/>
(online tool to check SSL/TLS configuration)

<https://securityheaders.com/>
(online tool to check security-related HTTP headers)

Security Report Summary	
	Site: https://www.synopsys.com/software-integrity.html
	IP Address: 23.9.32.58
	Report Time: 05 Sep 2018 16:59:05 UTC
	Headers: <div><div>✓ X-Frame-Options</div><div>✓ X-XSS-Protection</div><div>✓ X-Content-Type-Options</div><div>✓ Content-Security-Policy</div><div>✓ Strict-Transport-Security</div><div>✗ Referrer-Policy</div><div>✗ Feature-Policy</div></div>

A7:2017 Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS)

User-controlled content in response gives an attacker the opportunity to insert HTML and scripts.

The inserted code gets rendered in a victim's browser.

Types of cross-site scripting:

- Reflected
- Stored or persistent
- DOM-based



<https://www.youtube.com/watch?v=IG7U3fuNw3A>

XSS Is Easy To Introduce



```
const userName = location.hash.match(/userName=([^;&]*)/)[1];  
// ...  
div.innerHTML += `Welcome ${userName}">`
```

Script Execution:

[http://www.vulnerable.site?userName=<img src=x onerror='alert\(document.cookie\)'](http://www.vulnerable.site?userName=<img src=x onerror='alert(document.cookie)')

There Are So Many Execution Contexts!

HTML5 Security Cheat Sheet


- Payloads can be used for security unit tests
- Useful in dynamic testing


Self-executing focus event via autofocus


This vector uses an input element with autofocus to call its own focus event handler - no user interaction required


```
<input onfocus=write(1) autofocus>
```


User submitted markup should not contain "autofocus" attributes.

▪ Firefox 4.0
▪ Firefox Latest

▪ Opera 9.0
▪ Opera Latest

▪ Safari 4.0
▪ Safari Latest

▪ Chrome 4.0
▪ Chrome Latest

▪ Internet Explorer 10.0
▪ Internet Explorer Latest

xss autofocus chrome opera

- http://www.w3.org/Bugs/Public/show_bug.cgi?id=9602
- <http://www.whatwg.org/specs/web-apps/current-work/multipage/association-of-controls-and-forms.html#autofocusing-a-form-control>

<https://html5sec.org>

XSS Prevention Is HARD

Each browser is different in how it parses and renders HTML

Execution Contexts

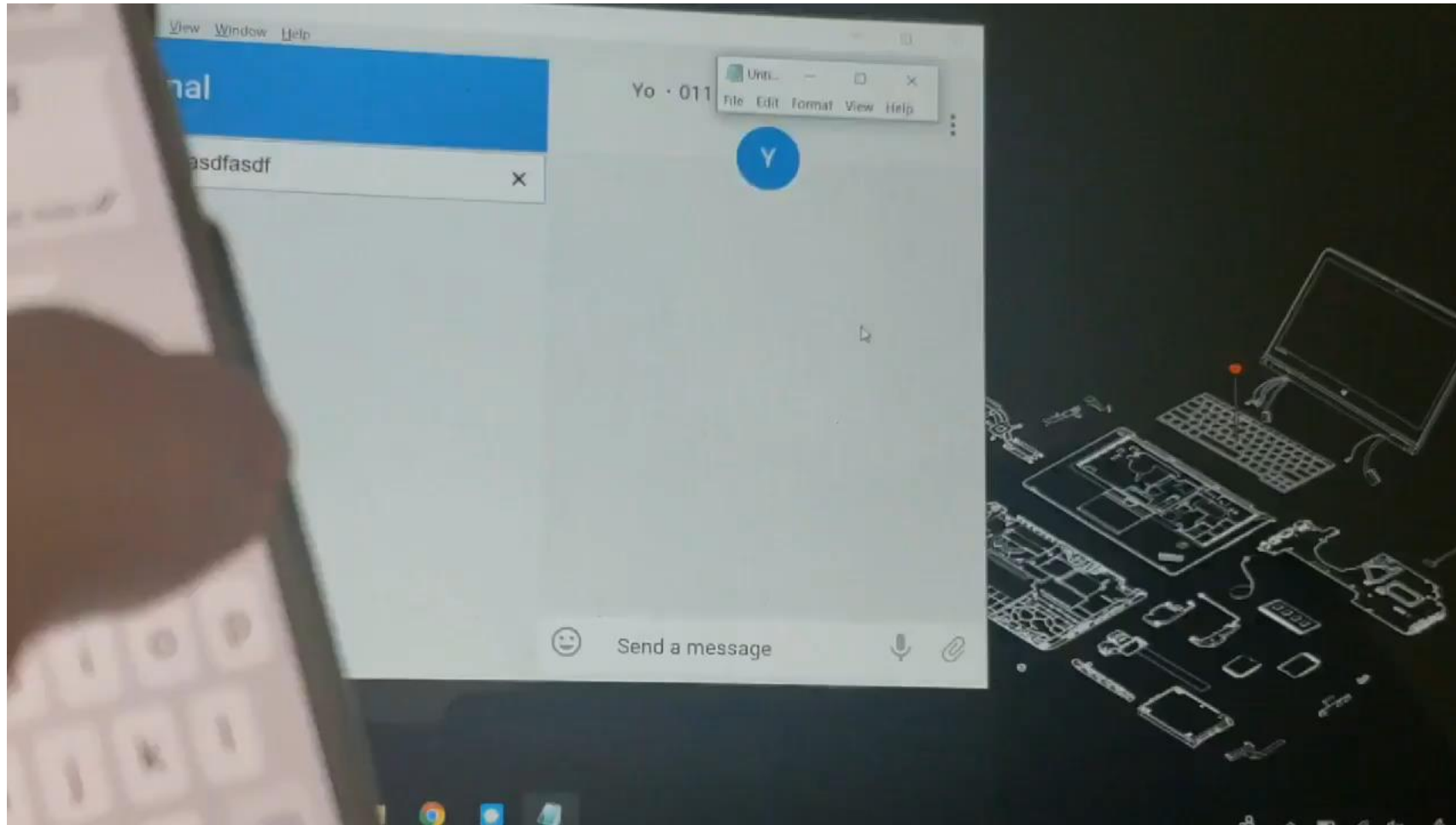
E2E prevents server inspection

Frameworks reduce XSS Attacks until

- Combining various [third-party libraries](#)
- Disabling [security controls](#)
- Strictly trusting user input - [trustAsHtml](#), [v-html](#), [bypassSecurityTrust](#), or [dangerouslySetInnerHTML](#)
- Include user-input as a prop in [<a href >](#) in React
- Direct access to the [DOM](#)
- Server Side [Rendering](#)
- Caching mechanisms such as [\\$templateCache](#)
- And many more!

Signal Creates a Lot of Noise

What happens if you bypass React controls for insecure use?



Source: https://ivan.barreraoro.com.ar/wp-content/uploads/2018/05/poc1.mp4?_=1

What Went Wrong?

Signal developers utilized *dangerouslySetInnerHTML* for phone and desktop leading to RCE in the desktop and Cross-Site Scripting (XSS) in iOS/Android

```
7 + import { MessageBody } from './MessageBody';
8 +
7 9 interface Props {
8 10   attachments: Array<QuotedAttachment>;
9 11   authorColor: string;
@@ -111,7 +113,9 @@ export class Quote extends React.Component<Props, {}> {
111 113
112 114   if (text) {
113 115     return (
114 -     <div className="text" dangerouslySetInnerHTML={{ __html: text }} />
116 +     <div className="text">
117 +       <MessageBody text={text} />
118 +     </div>
115 119   );
116 120 }
117 121
```

General Prevention Techniques

- Libraries and Frameworks for automatic Output Encoding and Sanitization
 - [Pug](#), [Mustache](#), [EJS](#)
 - [Angular](#), [React](#), [Vue](#)
 - [secure-filters](#)
- Sanitization for HTML, MathML and SVG with DOMPurify
 - <https://github.com/cure53/DOMPurify>
- Default to safe APIs
 - [innerText](#)
 - [encodeURIComponent](#)
 - [appendChild](#)

Templating Engine	HTML Output
Mustache {{code}}	Input
Jade/Pug #{code}	Input
EJS <%=code%>	Input

```
const createDOMPurify = require('dompurify');
const { JSDOM } = require('jsdom');

const window = (new JSDOM('')).window;
const DOMPurify = createDOMPurify(window);

const clean = DOMPurify.sanitize(dirty);
```

Caution: Always use the correct encoding context, [in the correct order](#)

Content Security Policy

Content Security Policy (CSP) can help prevent XSS by:

- Restricting ad-hoc XSS vectors such as inline scripts, third-party scripts, CSS, and eval()
- Imposing restrictions on resources based on their origin

CSP is set through a HTTP response header:

```
Content-Security-Policy: script-src 'self'
```

Other CSP directives include: `connect-src`, `font-src`, `frame-src`, `img-src`, `media-src`, `object-src`, `style-src` `default-src`

Note: It is still possible to allow the execution of inline scripts and eval() statements with CSP by specifying “unsafe-inline” and “unsafe-eval,” however, this is not recommended.

Creating and Evaluating CSP Policies

<https://csp.withgoogle.com/docs/index.html>

<https://csp-evaluator.withgoogle.com>

CSP Evaluator



CSP Evaluator allows developers and security experts to check if a Content Security Policy (CSP) serves as a strong mitigation against [cross-site scripting attacks](#). It assists with the process of reviewing CSP policies, which is usually a manual task, and helps identify subtle CSP bypasses which undermine the value of a policy. CSP Evaluator checks are based on a [large-scale study](#) and are aimed to help developers to harden their CSP and improve the security of their applications. This tool (also available as a [Chrome extension](#)) is provided only for the convenience of developers and Google provides no guarantees or warranties for this tool.

<https://chrome.google.com/webstore/detail/csp-mitigator/gijlobangojajlbodabkjpheeeokhfa>



CSP Mitigator

Offered by: David Ross

★★★★★ 8

Developer Tools

1,393 users

Strong CSP

CSP Engineers at Google recommend the following CSP to be strong:

- Backwards Compatible

```
script-src 'strict-dynamic' 'nonce-rAnd0m123' 'unsafe-inline' http: https;;  
object-src 'none';  
base-uri 'none';  
report-uri https://csp.example.com;
```

- CSP3 Only

```
script-src 'strict-dynamic' 'nonce-rAnd0m123';  
object-src 'none';  
base-uri 'none';  
report-uri https://csp.example.com;
```

Useful Twitter Thread: <https://twitter.com/LewisArdern/status/1112926476498698240>

Trusted Types

“Developers can have a high degree of confidence that the risk of DOM-based XSS remains low”

Trusted Types is an experimental browser feature

- <https://developers.google.com/web/updates/2019/02/trusted-types>
- `chrome --enable-blink-features=TrustedDOMTypes`
- `Content-Security-Policy: trusted-types *`

Attempts to lock down dangerous injection sinks:

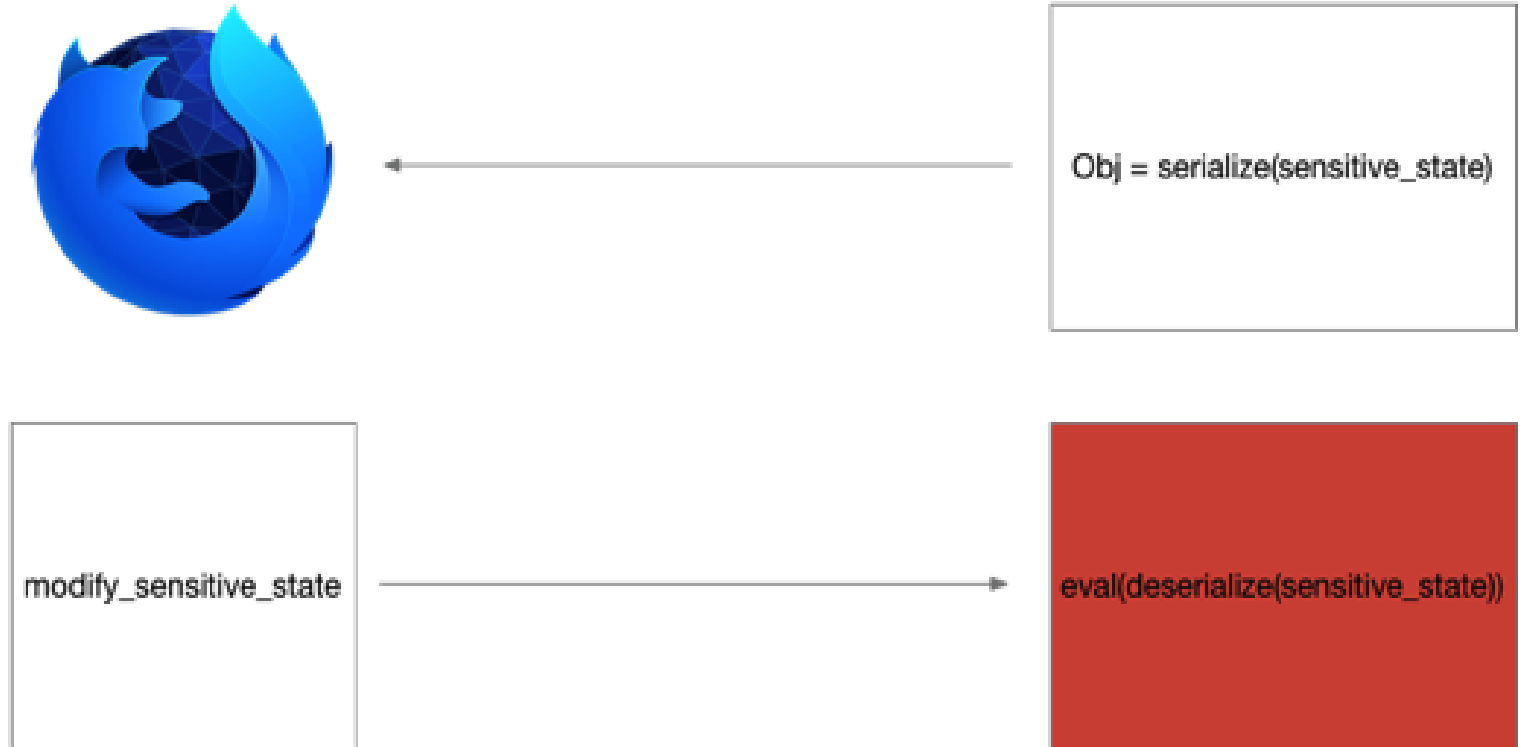
```
const templatePolicy = TrustedTypes.createPolicy('template', {
  createHTML: (templateId) => {
    const tpl = templateId;
    if (/^[0-9a-z-]$/i.test(tpl)) {
      return `
```

A8:2017 Insecure Deserialization

Insecure Deserialization

Serialized state is sent to the browser and then returned and deserialized on the server, which could lead to:

- Privileged state modification
- Remote code execution
- Denial of service



Avoid Unsafe Deserialization

NodeJS applications are vulnerable to RCE (Remote Code Execution) exploits if attacker-controlled data is deserialized via reflection.

- NPM (Node Package Manager) offers multiple packages to serialize or deserialize data.
- Avoid passing untrusted data to deserialization functions.

```
var serialize = require("node-serialize");
... snip ...
app.get("/", function(req, res){
  if(req.cookies.user){
    var user = Buffer.from(req.cookies.user, 'base64').toString();
    console.log("Received user object %s", user);

    var userObj = serialize.unserialize(user);

    res.send("Hello " + escape(userObj.username) + "!");
  }
}
```

Eval is Evil

But can be new Function, setTimeout, setInterval



The core issue behind [node-serialize](#) is the use of eval

```
for(key in obj) {
  if(obj.hasOwnProperty(key)) {
    if(typeof obj[key] === 'object') {
      obj[key] = unserialize(obj[key], originObj);
    } else if(typeof obj[key] === 'string') {
      if(obj[key].indexOf(FUNCFLAG) === 0) {
        obj[key] = eval('(' + obj[key].substring(FUNCFLAG.length) + ')');
      } else if(obj[key].indexOf(CIRCULARFLAG) === 0) {
        obj[key] = obj[key].substring(CIRCULARFLAG.length);
        circularTasks.push({obj: obj, key: key});
      }
    }
  }
}
```

Mitigate Deserialization

Safer Deserialization

If possible, avoid deserialization of data from clients and other untrusted sources

Use JSON data transfer objects (DTOs), as this format is simpler to protect

Encrypt or hash serialized objects

- Prevents tampering, but not replay

Check the object type is as expected

Apply security patches for software that contains known deserialization vulnerabilities

Popular libraries exist, but use with caution:

- <https://github.com/commenthol/serialize-to-js/blob/master/lib/deserialize.js#L33>

Demo

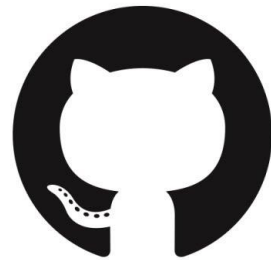
Node Deserialization

A9:2017 Using Components with Known Vulnerabilities

Security Issues with Third-Party Components

Reusable components make development easier.

- The NPM site <https://www.npmjs.com> contains third-party components
- A high GitHub rating does not guarantee that the component is secure
- Perform a security audit against 3rd party code
- Subscribe to the project repository
 - If you find a security issue, notify the project maintainer
 - Watch the project for newly discovered issues



GitHub



snyk

Examples of Components with Known Vulnerabilities

These are examples of popular components with known vulnerabilities:

- **Prototype Pollution In Lodash:** [CVE-2018-3721](#) in [Lodash](#) impact in some cases was denial of service (DoS), Remote Code Execution (RCE), and even bypass security controls.
- **Directory Traversal in Next.js:** [CVE-2018-6184](#) in [Next.js](#) allowed for arbitrary read of the file system
- **Cross-Site-Scripting (XSS) in Next.js:** [CVE-2018-18282](#) in [Next.js](#) allowed for XSS on the `/_error` page
- **Privilege Escalation in auth0-js:** [CVE 2018-6873](#) in [auth0-js](#) did not validate JWT audience which allowed for Privilege Escalation
- **Arbitrary Command Injection in Kibana:** [CVE-2018-17246](#) in [Kibana](#) allowed for arbitrary command execution in the Console Plugin.

Mitigation Techniques

Track use of outdated third-party components and update where necessary:

- Maintain a technology assets inventory to track components and dependencies
 - [yarn why](#), [yarn list](#), [npm ls](#), [bower-list](#)
- Review the inventory on a regular basis for known vulnerabilities
- Track known risks and vulnerabilities in the environment
- Develop a process to update, and regression test external components
- Pin Dependency versions where possible
 - Reduce the risk of another [event-stream](#) affecting you
 - npm
 - [npm-shrinkwrap.json](#) and [package-lock.json](#)
 - Yarn
 - [yarn.lock](#)

Known Issues in JavaScript Libraries

- Always check for known security issues:
 - GitHub automatically reports security issues
 - Depending on project type utilize tools:

Example	Command
npm	npm audit --fix
yarn	yarn audit --fix
bower	auditjs --bower bower.json
Client-Side JavaScript	retire --js /path/
Node.js Open-Source	snyk test

A10:2017 Insufficient Logging and Monitoring


Avoid Log Injection Attacks

Log injection is a common attack method aimed at forging log data or attacking administrators viewing the logs.

JavaScript loggers log4js and console.log are vulnerable to log injection attacks.

Example payload:

`http://example.com/?text=hello%0D%0AReceived:%20fakeentry`



```
app.get('/', function (req,res) {  
  console.log("Received:"+req.query.text);  
});
```


```
Received:hello  
Received:fakeentry
```


Use Winston to Enable Secure Logging Practices

Winston is a logging library that escapes data to prevent log injection:

```
{ "level": "info", "message": "Received: hello\\r\\nReceived: fakeentry", "timestamp": "2015-11-16T21:09:57.024Z" }
```

Winston also provides various built-in security controls, such as content filters:



```
var logger = new Winston.logger({filters: [function(level, msg, meta){  
  return maskCardNumbers(msg);  
}]};  
winston.log('info', "Card 123456789012345 processed.");
```

Resulting in the following output:

```
info: Card 123456****2345 processed.
```

Note: [node-bunyan](#) is another logging library that logs JSON strings securely.

Recommended Reading:

OWASP Top 10

https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

OWASP Application Security Verification Standard

https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project

OWASP Proactive Controls

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

OWASP Testing Guide

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

OWASP Cheat Sheet Series

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

BSIMM

<https://www.bsimm.com/>

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

SafeCode

<https://safecode.org>

Microsoft Agile SDL

<https://www.microsoft.com/en-us/SDL/discover/sdlagile.aspx>



Pragmatic Web Security

Security training for developers

SECURITY CHEAT SHEET

Version 2018.002

ANGULAR AND THE OWASP TOP 10

The OWASP top 10 is one of the most influential security documents of all time. But how do these top 10 vulnerabilities resonate in a frontend JavaScript application?

This cheat sheet offers practical advice on handling the most relevant OWASP top 10 vulnerabilities in Angular applications.

DISCLAIMER This is an opinionated interpretation of the OWASP top 10 (2017), applied to frontend Angular applications. Many backend-related issues apply to the API-side of an Angular application (e.g., SQL injection), but are out of scope for this cheat sheet. Hence, they are omitted.

<https://cheatsheets.pragmaticwebsecurity.com/cheatsheets/angularOWASPTop10.pdf>

Recommended Open Source Analysis Tools

Referencing only projects that are either open-source or scan open-source:

Products that perform JavaScript data flow analysis:

- [Coverity Scan](#)
- [LGTM](#)

Tools that look for areas of interest:

- [Tarnish](#)
- [JSHint](#)
- [JSLint](#)
- [ESLint](#)
 - [Code Climate - nodesecurity plugin](#)
- [TSLint](#)
 - [tslint-config-security](#)
 - [tslint-angular-security](#)

Tools that look for known issues in JavaScript libraries:

- [Retire.js](#)
- [npm audit](#)
- [yarn audit](#)
- [GitHub](#)
- [Snyk](#)
- [auditjs](#)

Tools that deobfuscate JavaScript:

- [Closure Compiler](#)
- [JStillery](#)
- [Unminify](#)
- [Jsnice](#)
- [jsdetox](#)
- [prepack.io](#)

ESLint Security Rules

- ESLint can help security consultants look for points of interest
- Default security rule configs
 - NodeJS <https://github.com/nodesecurity/eslint-config-nodesecurity>
 - VanillaJS <https://github.com/mozfreddyb/eslint-config-scanjs>
 - AngularJS <https://github.com/LewisArdern/eslint-plugin-angularjs-security-rules>
 - React <https://github.com/yannickcr/eslint-plugin-react#list-of-supported-rules>
- Security rules
 - [eslint-plugin-scanjs](#)
 - [eslint-plugin-security](#)
 - [eslint-plugin-react](#)
 - [eslint-plugin-angularjs-security](#)
 - [eslint-plugin-no-wildcard-postmessage](#)
 - [eslint-plugin-no-unsafe-innerhtml](#)
 - [vue/no-v-html](#)
 - [eslint-plugin-prototype-pollution-security-rules](#)

Vulnerable Machines



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

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

<https://github.com/dbohannon/MEANBug>

<https://github.com/appsecco/dvna>

Thank you!

Questions?

Email: lewis@ardern.io

Website: <https://ardern.io>

Twitter: <https://twitter.com/LewisArdern>

GitHub: <https://github.com/LewisArdern>

LinkedIn: <https://www.linkedin.com/in/lewis-ardern-83373a40>