So you thought you were safe using AngularJS. . . . Think again!
Who Am I?

- Lewis Ardern
- Ph.D. candidate Leeds Beckett University
- Security Consultant at Synopsys, previously Cigital

- Twitter @LewisArdern

Research Interests:
- Browser Security
- JavaScript
- HTML5
- Static analysis
Agenda

• AngularJS In A Nut Shell
• AngularJS Security Protections
• AngularJS Security Issues
• Third-Party Library Security Issues
• Look To The Future
AngularJS In A Nut Shell

• AngularJS is an open source front-end JavaScript framework
• What is the current version of AngularJS:
  – AngularJS 1.6.5
  – Angular 4.3.0
• Angular
  – MVC - Model View Controller
  – MVVM - Model View ViewModel
  – MVW - Model View Whatever
• Originally developed by Miško Hevery, then open sourced, and now maintained by Google
• What are the benefits of AngularJS?
  – Separation of HTML, CSS, and JavaScript logic
  – Convenience in DOM manipulations
  – Performance
• If AngularJS is on the front-end, what technologies are used on the back end?
  – Whatever: NodeJS, Java, C#, you name it
• A lot of Angular applications are built as single-page applications (SPA)
Angular and OWASP Top 10

- OWASP Top 10 issues that Angular code may have:

<table>
<thead>
<tr>
<th>OWASP Top 10</th>
<th>Kinda</th>
</tr>
</thead>
<tbody>
<tr>
<td>Injection (SQL, Command, LDAP)</td>
<td></td>
</tr>
<tr>
<td>Broken AuthN and Session Management</td>
<td></td>
</tr>
<tr>
<td>Cross-site scripting</td>
<td></td>
</tr>
<tr>
<td>Insecure Direct Object Reference</td>
<td></td>
</tr>
<tr>
<td>Security Misconfiguration</td>
<td></td>
</tr>
<tr>
<td>Sensitive Data Exposure</td>
<td></td>
</tr>
<tr>
<td>Missing Function Level Access Control</td>
<td></td>
</tr>
<tr>
<td>CSRF</td>
<td></td>
</tr>
<tr>
<td>Using Components with Known Vulnerabilities</td>
<td></td>
</tr>
<tr>
<td>Unvalidated Redirects and Forwards</td>
<td></td>
</tr>
</tbody>
</table>

Kinda
AngularJS Security Protections
XSS Protection: Output Encoding

• Automatic output encoding
  – Encoding is context aware (HTML element, attribute, URL)
  – All unsafe symbols are encoded, nothing is removed
  – Used with ng-bind

```html
<p ng-bind="htmlCtrl.welcome"></p>
```

```html
<p class="ng-binding" ng-bind="htmlCtrl.html">
  &lt;p style="color:blue"&gt;Hey!! Come and &lt;em style="color:Red" onmouseover="this.textContent='Click'" &gt;Mouse Hover&lt;/em&gt; Over Me&lt;/p&gt;
</p>
```
XSS Protection: Strict Contextual Escaping

• Before AngularJS version 1.2
  – ng-bind-html-unsafe directive

• SCE (Strict Contextual Escaping) – uses ngSanitize module
  – Sanitization for a particular context: HTML, URL, CSS
  – Used with ng-bind-html
  – Enabled by default in versions 1.2 and later, but can be disabled
    • $sceProvider.enabled(false)
    • $sce.trustAs(type, value) or $sce.trustAsHtml(value)
    • Other $sce.trustAs methods can be in custom directives
XSS Protection: Content Security Policy

• CSP disallows the use of eval() and inline scripts by default

• CSP is configurable

• Angular separates HTML, CSS, and JavaScript > no inline scripts!

• Angular code is compatible with CSP out of the box

• Caveats:
  – Angular uses eval() internally to parse expressions
    • https://github.com/angular/angular.js/blob/0694af8fc4c856f5174545450091602e51f02a11/src/Angular.js#L1120
  – Angular may use inline styles, not inline scripts (for ngCloack, ngHide)
    • https://github.com/angular/angular.js/blob/0694af8fc4c856f5174545450091602e51f02a11/src/Angular.js#L1111
  – Angular without unsafe eval() runs 30% slower when parsing expressions
## XSS Protection: Enforcing Content Security Policy

<table>
<thead>
<tr>
<th>Angular Setting</th>
<th>Code</th>
<th>Angular Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nothing</td>
<td><code>&lt;body ng-app&gt;</code></td>
<td>Use inline scripts, check for unsafe eval in the CSP header</td>
</tr>
<tr>
<td>Default CSP</td>
<td><code>&lt;body ng-app ng-csp&gt;</code></td>
<td>No inline scripts, no eval</td>
</tr>
<tr>
<td>No-unsafe-eval</td>
<td><code>&lt;body ng-app ng-csp=&quot;no-unsafe-eval&quot;&gt;</code></td>
<td>Eval cannot be used, but it’s ok to use inline styles CSP must have: style-src ‘unsafe-inline’</td>
</tr>
<tr>
<td>No-inline-style</td>
<td><code>&lt;body ng-app ng-csp=&quot;no-inline-style&quot;&gt;</code></td>
<td>Angular can use eval, but cannot use inline styles CSP must have: script-src ‘unsafe-eval’</td>
</tr>
</tbody>
</table>

**Note:** inline styles may be abused by attackers

- See Mario Heiderich’s paper on scriptless attacks
  - [https://www.nds.rub.de/media/emma/veroeffentlichungen/2012/08/16/scriptless Attacks-ccs2012.pdf](https://www.nds.rub.de/media/emma/veroeffentlichungen/2012/08/16/scriptless Attacks-ccs2012.pdf)

Instead of allowing ‘unsafe-inline’ for styles, developers can include angular-csp.css in the HTML for ngShow and ngHide directives to work.
XSS Protection: Bypassing The Content Security Policy

- Injected content can abuse Angular to execute code despite the CSP

```html
<html>
<head>
  <meta http-equiv=content-security-policy content="object-src 'none';script-src 'nonce-secret';">
  <script nonce-secret src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.4/angular.min.js"></script>
</head>
<body>
  <div ng-app ng-csp>
    <div ng-focus="x$event; id? tabindex=0">foo</div>
    <div ng-repeat="{key, value} in x.view">
      <div ng-if="key == 'window'">{{ [1].reduce(value, 1337); }}</div>
    </div>
  </div>
</body>
</html>
```

Slightly modified CSP bypass example from http://sirdarckcat.github.io/csp/angular.html#f

http://sebastian-lekies.de/csp/bypasses.php
XSS Protection: Sandbox? Not Really

- All versions of Angular up to 1.6 executed Angular Expressions in a sandbox
- Every version had a sandbox escape “vulnerability”
- Sandbox was never considered to protect code for security reasons
- What does it mean “to escape a sandbox”?
  - Directly manipulate the DOM
  - Execute plain old vanilla JavaScript

- Example payload:
  
  ```javascript
  {{x = {'y':".constructor.prototype}; x['y'].charAt=[].join;eval('x>alert(1)');}}
  ```
  
  - [http://blog.portswigger.net/2016/01/xss-without-html-client-side-template.html](http://blog.portswigger.net/2016/01/xss-without-html-client-side-template.html)

- As of Angular 1.6 sandbox has been completely removed
  
  - [https://blogs.synopsys.com/software-integrity/2016/12/28/angularjs-1-6-0-sandbox/](https://blogs.synopsys.com/software-integrity/2016/12/28/angularjs-1-6-0-sandbox/)
**AngularJS Security**

LiveOverflow • 5 videos • 4,247 views • Last updated on 14 Oct 2016

XSS with AngularJS. Bypassing the JavaScript security sandbox.

<table>
<thead>
<tr>
<th>No.</th>
<th>Title</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introducing the AngularJS Javascript Framework - XSS with AngularJS 0x00</td>
<td>7:50</td>
</tr>
<tr>
<td>2</td>
<td>Sandbox Bypass in Version 1.0.8 - XSS with AngularJS 0x1</td>
<td>8:02</td>
</tr>
<tr>
<td>3</td>
<td>Previous Bypass is now fixed in version 1.4.7 - XSS with AngularJS 0x2</td>
<td>6:03</td>
</tr>
<tr>
<td>4</td>
<td>New Sandbox Bypass in 1.4.7 - XSS with AngularJS 0x3</td>
<td>11:42</td>
</tr>
<tr>
<td>5</td>
<td>Sandbox bypass for the latest AngularJS version 1.5.8 - XSS with AngularJS 0x4</td>
<td>6:35</td>
</tr>
</tbody>
</table>
CSRF Protection: Help from the Client

- CSRF token must be generated and validated on the server side
- Angular automatically reads a cookie sent from the server and appends the value to an HTTP header
- What a developer needs to do:
  - Securely generate CSRF token on the server-side
  - Add a cookie XSRF-TOKEN with the token value
  - Angular will add a custom header X-XSRF-TOKEN with the token value
  - Verify on the server if the X-XSRF-TOKEN value matches the cookie XSRF-TOKEN value
  - If the token and the cookie values do not match, reject the request
  - The cookie and header values may be changed in Angular via the $http.xsrfHeaderName and $http.xsrfCookieName options to support whatever backend solution

AngularJS Security Issues
Loading Angular templates insecurely

- The templateURL which is used to render angular templates for routing, directives, ngSrc, ngInclude, etc.

- By default resources are restricted to the same domain and protocol as the application document.

- To load templates from other domains or protocols you may either whitelist or wrap them as trusted values.

- You can change these by setting your own custom whitelists and blacklists for matching such URLs.
Loading Angular templates insecurely

- To solve the problem of not being able to load resources from another domain, an insecure whitelist may have been created in which any domain is allowed by configuring the $sceDelegateProvider.resourceUrlWhitelist using wildcards like the example below:

```javascript
angular.module('myApp', []).config(function($sceDelegateProvider) {
  $sceDelegateProvider.resourceUrlWhitelist([ // Insecure - the wildcard allows resource loading from any domain using any protocol
    '**'
  ]);});
```

```javascript
angular.module('myApp', []).config(function($sceDelegateProvider) {
  $sceDelegateProvider.resourceUrlWhitelist([ // Insecure - loads over HTTP, wildcard allows for any subdomain and any directory
    'http://**.example.com/**'
  ]);});
```


Loading Angular templates securely (Remediation)

- Configure the specific protocol and domain or sub domain(s) for the resources you trust

- Never use just the double asterisk (**) wildcard to allow arbitrary domains and protocols

- Never use the double asterisk (**) wildcard as part of the protocol or domain, only at the end of a URL

- Ensure resources are loaded over a secure protocol (e.g., only allow https:// URLs)

- The blacklist can be used as a defense-in-depth measure to prevent resourcing templates that have known vulnerabilities within your application
Open Redirect

- The $window.location property enables developers to read/write to the current browser location

- The API exposes the "raw" object with properties that can be directly modified

- By setting the $window.location.href property to a URL, the browser will navigate to that page, even if it is outside of the domain of the current application

- An attacker could use this vulnerability to perform a XSS attack by using a URL that starts with javascript:
Open Redirect (Remediation)

• Open redirects can be prevented by hardcoding the URLs.

```javascript
var redirecturl = 'welcome.html';
```

• Use a whitelist of accepted URLs

```javascript
if(redirecturl != 'welcome.html')
    return;
```

• Use indirect reference maps

```javascript
var dict = {
    'welcome': "welcome.html"
};
if(dict[redirecturl])
    redirecturl = dict[redirecturl];
else
    redirecturl = 'welcome.html';
```

• If absolute URLs need to be used, verify that they start with http(s):

```javascript
var pattern = /(^((http|https):\/\/));
if(!pattern.test(redirecturl))
    return;
```
Open Redirect

DEMO
Expression Injection

- Mixing server-side and client-side templates can cause XSS without the need to inject HTML tags
- User input added to server-side template and then sent to client-side template:
  - Server-side template engine only escapes malicious HTML characters (e.g., <, >, “, ‘)
  - Attacker can place AngularJS expression language within {{ }}
    - Will not be escaped by server-side code
    - Will be executed by the client-side AngularJS template
    - Will run within a sandbox with limited execution of JavaScript (prior to version 1.6)
    - Sandbox bypass is always possible!

- Avoid using both client-side and server-side templates!
  - Keep app logic on server side and presentation on client side
Expression Injection

Malicious AngularJS code is injected through input

Only HTML special characters are encoded

AngularJS code is injected through input

AngularJS expression written to template

Angular engine renders AngularJS expressions including malicious code

Malicious code executes within the view
Expression Injection (Remediation)

• Where possible re-write Angular templates to be purely an AngularJS page instead of being rendered from the server
  – Assign the returned data to a $scope object and display that data within an expression
  – Return data to ng-bind or ng-bind-html
• Reduce the scope of the ng-app directive.
  ➢ Bind to a specific <div>, <table>, etc. rather than <body>

```html
<body>
  ...
  <div ng-app='myApp'>
    ...
  </div>
</body>
```

• Use the ng-non-bindable directive

```html
<p ng-non-bindable id='message'></p>
```

• Sanitize untrusted input to remove curly braces

• **Note: An attacker with the ability to inject HTML markup could bypass these controls**
Expression Injection

DEMO
Untrusted input treated as Angular expressions

- Angular expressions are code snippets (similar to JavaScript) that can be executed through various methods in Angular
  - AngularJS can evaluate expressions
  - AngularJS can order data using expressions
  - AngularJS can parse expressions
### Untrusted input treated as Angular expressions

#### $scope Methods
- $eval([expression], [locals]);
- $evalAsync([expression], [locals]);
- $apply([exp]);
- $applyAsync([exp]);
- $watch(watchExpression, listener, [objectEquality]);
- $watchGroup(watchExpressions, listener);
- $watchCollection(obj, listener);

#### orderBy
- `{ { collection | orderBy: expression : reverse : comparator } }
- $filter('orderBy')(collection, expression, reverse, comparator)
- angular.controller('ExampleController', ['$scope', 'orderByFilter', function($scope, orderByFilter) {
    $scope.friends = orderByFilter(collection, expression, reverse, comparator);
}])

#### Services
- $compile(element, transclude, maxPriority);
- $parse(expression);
- $interpolate(text, [mustHaveExpression], [trustedContext], [allOrNothing]);

---

http://blog.portswigger.net/2017/05/dom-based-angularjs-sandbox-escapes.html
Untrusted input treated as Angular expressions (Remediation)

- If possible, avoid using user-input to create expressions.

- If user-input needs to be used in expressions, only use it as data within those expressions, not as part of the expression code.

```javascript
$scope.$evalAsync('result = "Hello " + userInput + "!"');
```

- If user-input needs to be evaluated as part of the expression code, strict input validation must be used to prevent arbitrary code execution.

```javascript
if(window.location.search) {
  var orderby = decodeURIComponent(window.location.search.split("=")[1]);
  //Using the external Object.prototype.hasOwnProperty.call() in the unlikely event that 'hasOwnProperty'
  //has been overwritten on the object we check
  //In most cases, the simpler $scope.friends[0].hasOwnProperty(orderby) would work fine.
  if($scope.friends[0] !== undefined && Object.prototype.hasOwnProperty.call($scope.friends[0], orderby)) {
    $scope.orderby = orderby;
  }
}
```
OrderBy Filter

DEMO
angular.element

• Angular provides its own subset of the JQuery language that is accessible via the angular.element global function

• Using untrusted input in some of the element functions may lead to XSS:
  – angular.element.after
  – angular.element.append
  – angular.element.html
  – angular.element.prepend
  – angular.element.replaceWith
  – angular.element.wrap

• As a developer you must validate the input before sending it to the angular.element functions with functions such as $sce.getTrustedHtml or $sanitize.
XSS in angular.element

- Reading data from user

```html
<form>
  <label>After:</label><input type="text" ng-model="afterinput" />
  <button type="submit" ng-click="aftersubmit()">Submit</button>
</form>

<div ng-controller="View1Ctrl">
  <div id="testDiv">{{name}}</div>
</div>

- Inserting data in Angular code

```javascript
controller('View1Ctrl', ['$scope', '$document', function($scope, $document) {
  $scope.name = "ChangeMe";
  var element = angular.element($document[0].querySelector('#testDiv'));
  $scope.aftersubmit=function()
  {
    if($scope.afterinput) element.after($scope.afterinput);
  }
```
XSS in angular.element

- Payload: `<p onmouseover>alert('after');>After</p>`

- Why is there an injection?
- SCE is not automatically applied to angular.element
Third-Party Library Security Issues
Third-Party Libraries

• Third-party libraries enhance our applications

• There is always a risk with using third-party code

• AngularJS libraries are no different

• When looking at incorporating libraries into your application you should:
  – Review the projects Github issue list
  – Use OSS tools such as ESLint (eslint-plugin-scanjs-rules)
  – Identify components with known vulnerabilities using Retire.js and Snyk
  – Look for XSS with tools such as Blue Closure Detect
  – Manually review the code (time consuming)
XSS in angular-translate

• Plugin angular-translate is used for pages internationalization

```html
<div translate="GREETING" translate-values="{translateValues.name}"/>
```

• Setting translation strategy to ‘null’ or leaving it out (default) leads to XSS

```javascript
angular.module('app').config(function($translateProvider) {
    $translateProvider.translations('en', {GREETING: 'Hello <b>{{name}}</b>'});
    $translateProvider.translations('de', {GREETING: 'Hallo <b>{{name}}</b>'});
    $translateProvider.preferredLanguage('en');
});

angular.module('app').controller('Ctrl', function($scope, $translate, $routeParams, $route, $translateSanitization) {
    $translateSanitization.useStrategy();
    $scope.translateValues = {name: $routeParams.name};
    var lang = $routeParams.lang;
    if (lang !== undefined) {
        $translate.use(lang);
    }
    ...
});
```

Angular Translate Example:

- XSS

![Angular Translate Example](image-url)
textAngular

- The textAngular module is a WYSIWYG editor with collaborative editing functionality

- The editor processes the input and displays it (including HTML tags)

- textAngular uses textAngular-sanitizer module
  - Only verifies that an href starts with “http”
  - The string is then encoded and saved on the server

- textAngular parses the link and creates a new element with the content of the link as an unencoded HTML element
XSS in TextAngular

- Sample payload:
  http://A/A<img src=x onerror=alert('XSS_Success')>

```html
<p>
Enter your comment
<a target="" href="http://A/A<img src=x onerror=alert('XSS_Success')">">here!</a>
</p>
```

![XSS_in_TextAngular](image.png)
XSS in TypeAhead

- TypeAhead module shows hints as the user starts typing in a text field
- The list of hints is not sanitized if at least one condition is met:
  - ui.bootstrap version prior to 0.13.4 is used
  - ngSanitize is not included

```html
<form ng-submit="submit()">
  <input type="text" ng-submit="submit()"
    ng-model="search_val"
    typeahead="search_val for search_val in searches"
    class="form-control">
  <input type="submit" value="Search"/>
</form>

var module = angular.module('app', ['ui.bootstrap'])

module.controller('TypeaheadCtrl',
  function($scope,$http) {
    $scope.selected = undefined;
    $scope.searches = [
      decodeURIComponent(window.location.search.split('?' )[1])
    ];
  })
```

<script src="http://angular-ui.github.io/bootstrap/ui-bootstrap-tpls-0.13.3.js"></script>
Angular 2,4,*,*,*

Look To The Future
Angular 2, 4 and beyond

• It’s difficult to write complex but secure applications

• Angular 1.X contained many features that could introduce security problems

• Angular 2 attack surface is much smaller
Angular 2, 4 and beyond

- Unidirectional data binding
  - Interpolation, One/two way binding, Event Binding

- No more watchers, $apply/Async, $compile, $interpolate, $eval/Async

- Vulnerable features not introduced

Appendix: No FilterPipe or OrderByPipe

Angular doesn't provide pipes for filtering or sorting lists. Developers familiar with AngularJS know these as filter and orderBy. There are no equivalents in Angular.

- ES6
Angular 2, 4 and beyond

- Encoding and Sanitization by default
- Harmonizes with the Content Security Policy (CSP)
- Better naming conventions
  - `bypassSecurityTrustHtml(value: string)`
  - `bypassSecurityTrustStyle(value: string)`
  - `bypassSecurityTrustScript(value: string)`
  - `bypassSecurityTrustUrl(value: string)`
  - `bypassSecurityTrustResourceUrl(value: string)`
- Build-time security
  - Precompiled templates (see AoT [https://angular.io/docs/ts/latest/cookbook/aot-compiler.html](https://angular.io/docs/ts/latest/cookbook/aot-compiler.html))
Angular 2, 4 and beyond

Important notes:

• AngularJS is a client-side framework
  – The production flag can be disabled by the user
  – Client elements can be modified
    • ngShow and ngHide
    • RouteGuards are boolean
  – Sensitive data can be retrieved from localStorage and sessionStorage

• Security should be enforced on the server
  – Access control
  – AuthN/AuthZ
  – Strict input validation
  – Escaping/Encoding/Sanitization
Important notes:

- XSS can still occur through
  - Explicitly trusting user data
  - Expression injection
  - Third-party libraries

```php
<?php
    echo htmlentities($_GET['myParameter']);
?>
```

- Server-side interaction
Conclusion

- Use Angular, as it is a very secure framework:
  - Contextually-aware encoding
  - Strict contextual escaping
  - Separation of HTML and JavaScript – CSP compatible
- Do not mix server-side and client-side templates
- Do not directly use user-input in expressions
- Check plugins for security issues and use the latest version
- Embrace the Angular Migration from 1 to 4
- ...
- Profit
Thank you!

Questions?

Lewis Arden
Lewis.Ardern@Synopsys.com
Twitter: @LewisArdern
https://www.synopsys.com/software