ng-owasp: OWASP Top 10 for AngularJS Applications

2015 NDC

NORWEGIAN DEVELOPERS CONFERENCE

15-19 June·Oslo, Norway

Kevin Hakanson

Software Architect

THOMSON REUTERS

@hakanson

https://github.com/hakanson/ng-owasp

The OWASP Top 10 provides a list of the 10 most critical web application security risks. How do these relate to AngularJS applications? What security vulnerabilities should developers be aware of beyond XSS and CSRF?

This session will review the OWASP Top 10 with a front-end development focus on HTML and JavaScript. It will look at patterns to implement and others to consider avoiding. We will also explore several built-in features of AngularJS that help secure your application.

Housekeeping

After the session...

Hello

https://github.com/hakanson/ng-owasp

Examples

Feedback via Twitter

Slide Number

Comments on Video (future)

@hakanson

AngularJS FAQ

Do I need to worry about security holes in AngularJS?

Like any other technology, AngularJS is not impervious to attack. Angular does, however, provide built-in protection from basic security holes including cross-site scripting and HTML injection attacks. AngularJS does round-trip escaping on all strings for you and even offers XSRF protection for server-side communication.

AngularJS FAQ

Do I need to worry about security holes in AngularJS?

YES!

OWASP Top Ten

Powerful awareness document for web application security.

Represents a broad consensus about what the most critical web application security flaws are.

OWASP Top 10 - 2013

A1-Injection

A2-Broken Authentication and Session Management

A3-Cross-Site Scripting (XSS)

A4-Insecure Direct Object References

A5-Security Misconfiguration

A6-Sensitive Data Exposure

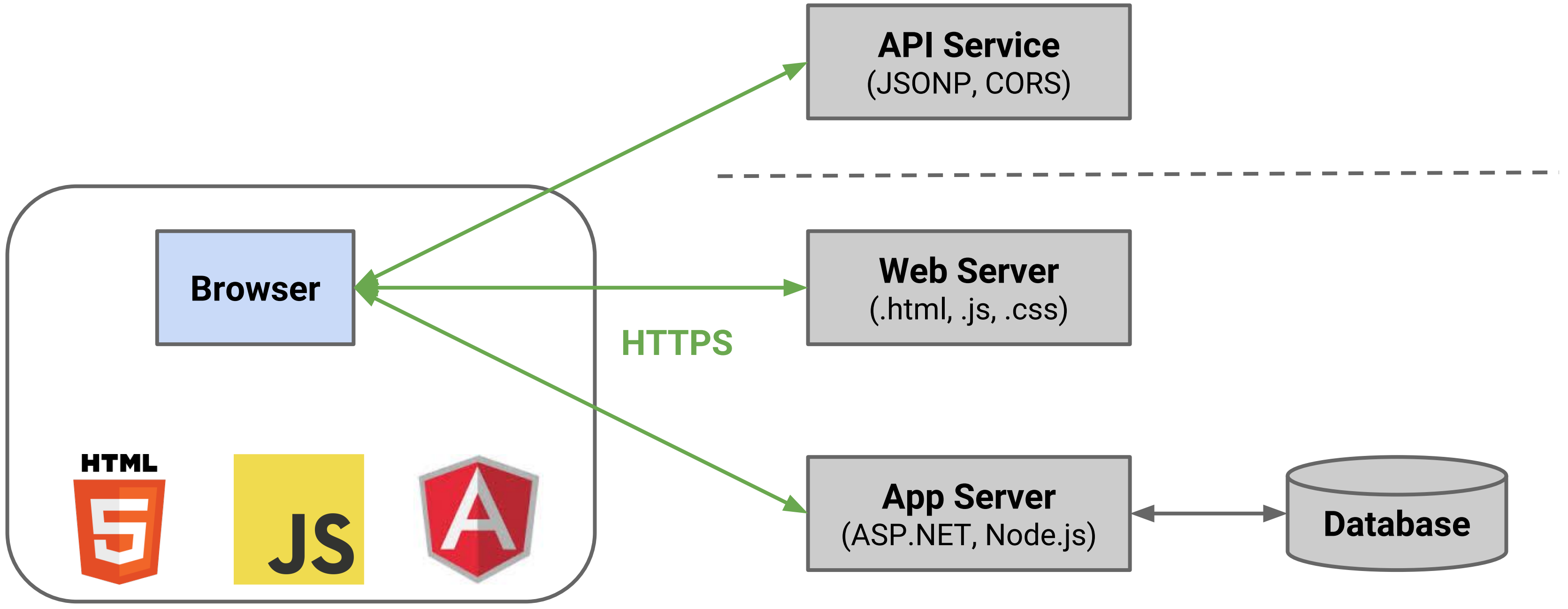
A7-Missing Function Level Access Control

A8-Cross-Site Request Forgery (CSRF)

A9-Using Components with Known Vulnerabilities

A10-Unvalidated Redirects and Forwards

html[ng-app]:focus, script:focus



List of useful HTTP headers

```
Strict-Transport-Security: max-age=16070400; includeSubDomains
X-Frame-Options: deny
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
Content-Security-Policy: default-src 'self'
Content-Security-Policy-Report-Only: default-src 'self'; report-
uri http://loghost.example.com/reports.jsp
```

A1-Injection

Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

Content Spoofing (Injection)

When an application does not properly handle user supplied data, an attacker can supply content to a web application, typically via a parameter value, that is reflected back to the user.

vs. Cross-site Scripting

Content spoofing is an attack that is closely related to Cross-site Scripting (XSS). While XSS uses `<script>` and other techniques to run JavaScript, content spoofing uses other techniques to modify the page for malicious reasons.

ngSanitize

The ngSanitize module provides functionality to sanitize HTML.

```
<script src="angular-sanitize.js"></script>
```

```
angular.module('app', ['ngSanitize']);
```

`$sanitize`

The input is sanitized by parsing the html into tokens. All safe tokens (from a whitelist) are then serialized back to properly escaped html string.

angular.js/src/ngSanitize/sanitize.js

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *     Any commits to this file should be reviewed with security in mind.  *
 *   Changes to this file can potentially create security vulnerabilities. *
 *          An approval from 2 Core members with history of modifying      *
 *                         this file is required.                          *
 *                                                                         *
 *  Does the change somehow allow for arbitrary javascript to be executed? *
 *    Or allows for someone to change the prototype of built-in objects?   *
 *     Or gives undesired access to variables likes document or window?    *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

`$sce`

$sce is a service that provides Strict Contextual Escaping (SCE) services to AngularJS.

SCE is a mode in which AngularJS requires bindings in certain contexts to result in a value that is marked as safe to use for that context. One example of such a context is binding arbitrary html controlled by the user via `ng-bind-html`.

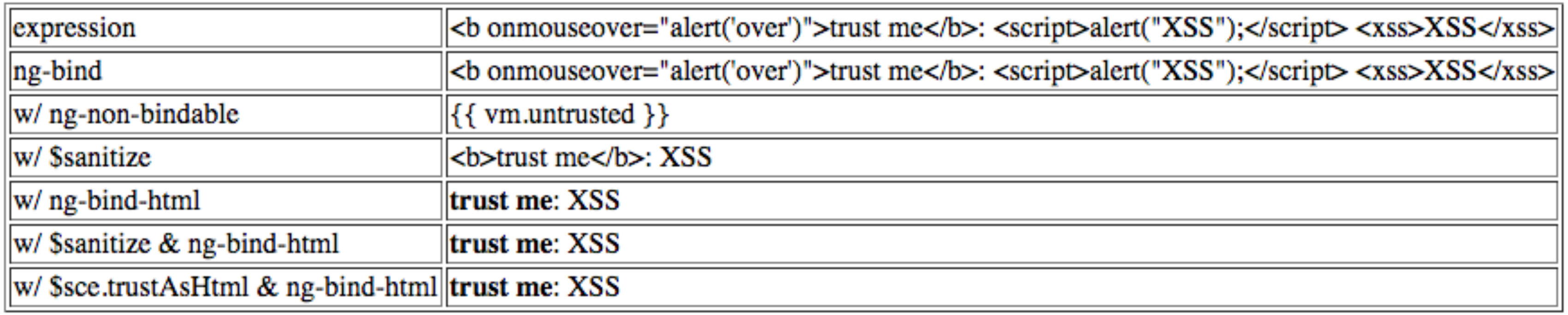
Example (JavaScript)

```
var data =
'<b onmouseover="alert(\'over\')">trust me</b>:
<script>alert("XSS");</script> <xss>XSS</xss>';

vm.untrusted = data;
vm.sanitized = $sanitize(data);
vm.trusted = $sce.trustAsHtml(data);
```

Example (HTML)

```
{{ vm.untrusted }}
<span ng-bind="vm.untrusted"></span>
<span ng-non-bindable>{{ vm.untrusted }}</span>
{{ vm.sanitized }}
<span ng-bind-html="vm.untrusted"></span>
<span ng-bind-html="vm.sanitized"></span>
<span ng-bind-html="vm.trusted"></span>
```



expression	<b onmouseover="alert('over')">trust me</b>: <script>alert("XSS");</script> <xss>XSS</xss>
ng-bind	<b onmouseover="alert('over')">trust me</b>: <script>alert("XSS");</script> <xss>XSS</xss>
w/ ng-non-bindable	{{ vm.untrusted }}
w/ $sanitize	<b>trust me</b>: XSS
w/ ng-bind-html	**trust me**: XSS
w/ $sanitize & ng-bind-html	**trust me**: XSS
w/ $sce.trustAsHtml & ng-bind-html	**trust me**: XSS

```
&lt;b onmouseover="alert('over')"&gt;trust me&lt;/b&gt;: &lt;script&gt;alert
("XSS");&lt;/script&gt; &lt;xss&gt;XSS&lt;/xss&gt;
```

```
&lt;b onmouseover="alert('over')"&gt;trust me&lt;/b&gt;: &lt;script&gt;alert
("XSS");&lt;/script&gt; &lt;xss&gt;XSS&lt;/xss&gt;
```

```
{{ vm.untrusted }}
```

```
&lt;b&gt;trust me&lt;/b&gt;:  XSS
```

```
<b>trust me</b>:  XSS
```

```
<b>trust me</b>:  XSS
```

```
<b onmouseover="alert('over')">trust me</b>: <script>alert("XSS");</script>
<xss>XSS</xss>
```

Demo

Expression Sandboxing

AngularJS's expressions are sandboxed not for security reasons, but instead to maintain a proper separation of application responsibilities.

This sandbox is not intended to stop attackers who can edit the template before it's processed by Angular.

https://docs.angularjs.org/guide/security

Expression Sandboxing

Design your application in such a way that users cannot change client-side templates. For instance:

- Do not mix client and server templates
- → Do not use user input to generate templates dynamically
- Do not run user input through `$scope.$eval`
- Consider using CSP (but don't rely only on CSP)

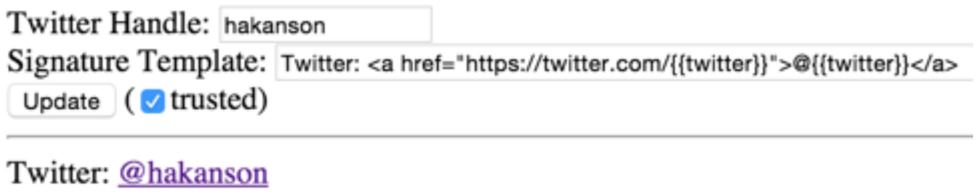
`$interpolate`

Compiles a string with markup into an interpolation function. This service is used by the HTML `$compile` service for data binding.

`$interpolate`

For security purposes, it is strongly encouraged that web servers escape user-supplied data, replacing angle brackets (<, >) with `&lt;` and `&gt;` respectively, and replacing all interpolation start/end markers with their escaped counterparts.

Example


Twitter Handle: hakanson
Signature Template: Twitter: <a href="https://twitter.com/{{twitter}}">@{{twitter}}</a>
Update (✓ trusted)

Twitter: @hakanson

```
vm.twitter = "hakanson";
vm.template = 'Twitter: <a href="https://twitter.com/{{twitter}}">@{{twitter}}</a>';
vm.trusted = false;
vm.rendered = "";

vm.update = function () {
  var context = { twitter : vm.twitter};
  i = $interpolate(vm.template);
  vm.rendered = i(context);
  if (vm.trusted == true) {
     vm.rendered = $sce.trustAsHtml(vm.rendered);
  }
};
```

Demo

angular-translate

```
pascalprecht.translate.$translateSanitization:
No sanitization strategy has been configured.
This can have serious security implications. See
http://angular-translate.github.
io/docs/#/guide/19_security  for details.
```

angular-translate

```
$translateProvider.useSanitizeValueStrategy()
```

- **sanitize**: sanitizes HTML in the translation text using `$sanitize`
- **escape**: escapes HTML in the translation
- **sanitizeParameters**: sanitizes HTML in the values of the interpolation parameters using `$sanitize`
- **escapeParameters**: escapes HTML in the values of the interpolation parameters

```
$translateProvider.translations('en', {
  GREETING: '<b>Hello</b> {{name}}',
  GREETINGX: '<b>Hello</b> {{name | uppercase}}'
});
$translateProvider.useSanitizeValueStrategy('sanitize');
```

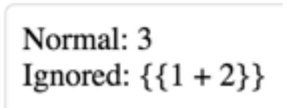
```
<div ng-bind-html="'GREETING' | translate:{name:'Kevin'}"></div>
<div ng-bind-html="'GREETINGX' | translate:{name:'Kevin'}"></div>
```

Demo

ngNonBindable

Tells Angular not to compile or bind the contents of the current DOM element.

```
<div>Normal: {{1 + 2}}</div>
<div ng-non-bindable>Ignored: {{1 + 2}}</div>
```



```
Normal: 3
Ignored: {{1 + 2}}
```

Use with server-side rendering of user input?

A2-Broken Authentication and Session Management

Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, or session ← tokens, or to exploit other implementation flaws to assume other users' identities.

Client State vs. Server State

- Session Keep Alive and Pinging
- Handle Session Timeouts
- OAuth Token Management
- Clear Storage on Logout
    - (Cookies, SessionStorage, LocalStorage)

Respond to Events

```
$rootScope.$on('Auth:Required', function() {
  $location.path('/login');
});

$rootScope.$on('Auth:Logout', function() {
  StorageService.clear();  // clear user info
  $rootScope.$broadcast('Auth:Required');
});
```

$httpProvider.interceptors

```
$httpProvider.interceptors.push(function($q, $rootScope) {
  return {
     'responseError': function(rejection) {
        if (rejection.status === 401) {
           $rootScope.$broadcast('Auth:Required');
        }
        return $q.reject(rejection);
     }
  };
});
```

Demo

A3-Cross-Site Scripting (XSS)

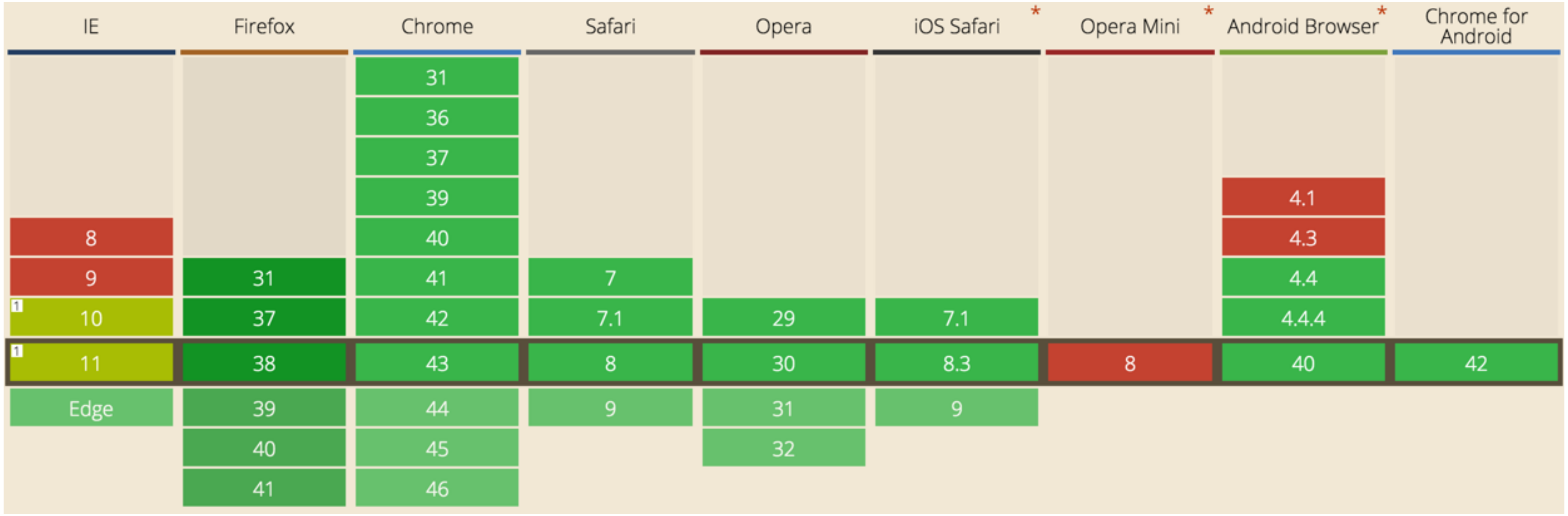
XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim’s browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

Content Security Policy

Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks.

Browser Support for CSP

http://caniuse.com/#feat=contentsecuritypolicy



IE	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
		31						
		36						
		37						
		39					4.1	
8		40					4.3	
9	31	41	7				4.4	
[1] 10	37	42	7.1	29	7.1		4.4.4	
[1] 11	38	43	8	30	8.3	8	40	42
Edge	39	44	9	31	9			
	40	45		32				
	41	46						

ngCsp

Enables CSP (Content Security Policy) support.

```
<!doctype html>
<html ng-app ng-csp>
...
...
</html>
```

ngCsp

AngularJS uses `Function(string)` generated functions as a speed optimization. Applying the ngCsp directive will cause Angular to use CSP compatibility mode. When this mode is on AngularJS will evaluate all expressions up to 30% slower than in non-CSP mode, but no security violations will be raised.

ngCsp

CSP forbids JavaScript to inline stylesheet rules. In non CSP mode Angular automatically includes some CSS rules (e.g. `ngCloak`). To make those directives work in CSP mode, include the `angular-csp.css` manually.

CSP 2 Inline `<script>` Hashes

```
<meta
     http-equiv="Content-Security-Policy"
     content="script-src 'self' 'sha256-qznLcsROx4GACP2dm0UCKCzCG+HiZ1guq6ZZDob/Tng='">
```

```
<script>alert('Hello, world.');</script>
```

```
$ echo -n "alert('Hello, world.');" | openssl
dgst -sha256 -binary | openssl enc -base64 -A
qznLcsROx4GACP2dm0UCKCzCG+HiZ1guq6ZZDob/Tng=
```

W3C Subresource Integrity (draft)

A validation scheme, extending several HTML elements with an integrity attribute that contains a cryptographic hash of the representation of the resource

```
<script src="alert.js"
        integrity="sha256-qznLcsROx4GACP2dm0UCKCzCG+HiZ1guq6ZZDob/Tng="
        crossorigin="anonymous"></script>
```

```
> document.body
    .appendChild(document.createElement("script"))
    .appendChild(document.createTextNode("alert('xss');"))

Refused to execute inline script because it violates the
following Content Security Policy directive: "script-src 'self'
'sha256-qznLcsROx4GACP2dm0UCKCzCG+HiZ1guq6ZZDob/Tng='". Either
the 'unsafe-inline' keyword, a hash ('sha256-LMD3-
o5SW1nMQnP8eqehN5Cf_fLDXyNLVQ1aWycvx8E='), or a nonce ('nonce-
...') is required to enable inline execution.
    (anonymous function)
    InjectedScript._evaluateOn
    InjectedScript._evaluateAndWrap
    InjectedScript.evaluate
```

Demo

A4-Insecure Direct Object References

A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.

$resource

A factory which creates a resource object that lets you interact with RESTful server-side data sources.

```
var CreditCard = $resource('/user/:userId/card/:cardId',
  {userId:123, cardId:'@id'}, {
     charge: {method:'POST', params:{charge:true}}
  });
```

A4 is Server Concern

Need server side validation of references

`$resource` makes it easier for attacker to understand and attempt to exploit

A5-Security Misconfiguration

Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. Secure settings should be defined, implemented, and maintained, as defaults are often insecure.
→ Additionally, software should be kept up to date.

Bower

- A package manager for the web
- Fetches, installs and updates packages
  - `bower install angular`
  - `bower update angular`
- Anyone can register a bower package
  - Angular registered their own bower packages https://github.com/angular?query=bower
- *see also: npm, jspm*

```
$ bower install angular
bower not-cached    git://github.com/angular/bower-angular.git#*
bower resolve       git://github.com/angular/bower-angular.git#*
bower download      https://github.com/angular/bower-angular/archive/v1.4.0.tar.gz
bower extract       angular#* archive.tar.gz
bower resolved      git://github.com/angular/bower-angular.git#1.4.0
bower install       angular#1.4.0

angular#1.4.0 bower_components/angular
```

```
$ bower install angular-sanitize
bower not-cached    git://github.com/angular/bower-angular-sanitize.git#*
bower resolve       git://github.com/angular/bower-angular-sanitize.git#*
bower download      https://github.com/angular/bower-angular-sanitize/archive/v1.4.0.tar.gz
bower extract       angular-sanitize#* archive.tar.gz
bower resolved      git://github.com/angular/bower-angular-sanitize.git#1.4.0
bower install       angular-sanitize#1.4.0

angular-sanitize#1.4.0 bower_components/angular-sanitize
└── angular#1.4.0
```

CORS “Credentials”

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

Should you send Cookies and HTTP Authentication data with all requests?

```
$httpProvider.defaults.withCredentials = true
```

A6-Sensitive Data Exposure

Many web applications do not properly protect
sensitive data, such as credit cards, tax IDs, and
authentication credentials. Attackers may steal or
modify such weakly protected data to conduct
credit card fraud, identity theft, or other crimes.
Sensitive data deserves extra protection such as
→ encryption at rest or in transit, as well as special
precautions when exchanged with the browser.

A6-Sensitive Data Exposure

- HTTPS / Strict-Transport-Security
- Putting sensitive data on URLs
- Storing sensitive data in LocalStorage without using encryption
    - Web Cryptography API

AngularJS 2.0 Data Persistence

The ngStore module contains services that provide access to local “disk” storage provided by native browser APIs like localStorage, sessionStorage and IndexedDB.

All services within the ngStore module will support encryption, managed through the `$localDB` service.

Design Doc (draft)

angular-cache

A very useful replacement for Angular's `$cacheFactory`.

The storage mode for a cache can be be "memory" , "localStorage", "sessionStorage" or a custom implementation.

When is storage cleared?

- variable / memory - when page closes
- session cookie - when browser closes
- sessionStorage - when browser closes
- persistent cookie - when expires
- localStorage - when explicitly cleared
- indexedDB - when explicitly cleared

Respond to Events

```
$window.addEventListener('beforeunload', function(event) {
  // clear user info
  StorageService.clear();
});
```

A7-Missing Function Level Access Control

Most web applications verify function level access rights before making that functionality visible in the UI. However, applications need to perform the same access control checks on the server when each function is accessed. If requests are not verified, attackers will be able to forge requests in order to access functionality without proper authorization.

A7 is Server Concern

Change from 2010 OWASP Top 10

A8: Failure to Restrict URL Access

AngularJS applications might not place access controls on static assets (html, css, js) hosted on web servers or content delivery networks.

Extend Route Configuration

```
$routeProvider
  .when('/contactus',
  {
    templateUrl: 'contactus.html',
    data : {
      myAnonymousAttr: true
    }
  })
```

Intercept Route Change

```
$rootScope.$on('$routeChangeStart', function(event, next, current) {
  var currentRoute = $route.routes[$location.path()];

  if (!AuthService.isAuthenticated()) {
    if (!(currentRoute && currentRoute.data
                      && currentRoute.data.myAnonymousAttr)) {
      $rootScope.$broadcast('Auth:Required');
      event.preventDefault();
    }
  }
});
```

Demo

angular-feature-flags

Control when you release new features in your app by putting them behind switches.

```
myApp.run(function(featureFlags, $http) {
  featureFlags.set($http.get('/data/flags.json'));
});
```

```
<div feature-flag="myFlag">
  I will be visible if 'myFlag' is enabled
</div>
```

A8-Cross-Site Request Forgery (CSRF)

A CSRF attack forces a logged-on victim’s browser to send a forged HTTP request, including the victim’s session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim’s browser to generate requests the vulnerable application thinks are legitimate requests from the victim.

`$http` Security Considerations

When designing web applications, consider security threats from:

- JSON vulnerability
- XSRF

Both server and the client must cooperate in order to eliminate these threats. Angular comes pre-configured with strategies that address these issues, but for this to work backend server cooperation is required.

JSON Vulnerability Protection

A JSON vulnerability allows third party website to turn your JSON resource URL into JSONP request under some conditions. To counter this your server can prefix all JSON requests with following string `")]}',\n"`. Angular will automatically strip the prefix before processing it as JSON.

JSON Vulnerability Protection

For example if your server needs to return:

```
['one','two']
```

which is vulnerable to attack, your server can return:

```
)]}',
['one','two']
```

Angular will strip the prefix, before processing the JSON.

Synchronizer Token Pattern

The synchronizer token pattern requires the generating of random "challenge" tokens that are associated with the user's current session.

When the user wishes to invoke these sensitive operations, the HTTP request should include this challenge token

XSRF Protection

- Angular provides a mechanism to counter XSRF. When performing XHR requests, the `$http` service reads a token from a cookie (by default, `XSRF-TOKEN`) and sets it as an HTTP header (`X-XSRF-TOKEN`).
- The header will not be set for cross-domain requests.

XSRF Protection

The name of the headers can be specified using the `xsrfHeaderName` and `xsrfCookieName` properties of either `$httpProvider.defaults` at config-time, `$http.defaults` at run-time, or the per-request config object.

CSRF Prevention Cheat Sheet

Any cross-site scripting vulnerability can be used to defeat token, Double-Submit cookie, referer and origin based CSRF defenses.

It is imperative that no XSS vulnerabilities are present to ensure that CSRF defenses can't be circumvented.

No Session Cookie?

- Then no CSRF?
- Pass authentication token on every HTTP request (`$httpProvider.interceptors`) `Authorization: Bearer <oauth-token>`
- Still need to protect against XSS!

A9-Using Components with Known Vulnerabilities

Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. → Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts.

Retire.js

The goal of Retire.js is to help you detect the use of JS-library versions with known vulnerabilities.

Retire.js has these parts:

1. A command line scanner
2. A grunt plugin
3. A Chrome plugin
4. A Firefox plugin
5. Burp and OWASP Zap plugin

Erlend Oftedal (@webtonull)

Main contributor to retire.js

OWASP Top 10 for JavaScript blog series

mustache-security

In AngularJS before 1.2.19 / 1.3.0-beta.14:

you can obtain a reference to Object with ({})["constructor"] (because although the name constructor is blacklisted, only the Function object was considered dangerous)

- all the interesting methods of Object are accessible
- fixed now

https://code.google.com/p/mustache-security/wiki/AngularJS

A10-Unvalidated Redirects and Forwards

Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

Sample Data

```
vm.links = [
  'http://angularjs.org/',
  'https://angularjs.org/',
  'https://t.co/rLBxlqZZ0c',
  'https://goo.gl/ZF7ddU',
  'https://www.google.com/#q=angular',
  'https://www.owasp.org/'
];
```

`$compileProvider`

```
aHrefSanitizationWhitelist([regexp]);
```

Retrieves or overrides the default regular expression that is used for whitelisting of safe urls during `a[href]` sanitization.

ngHref + aHrefSanitizationWhitelist

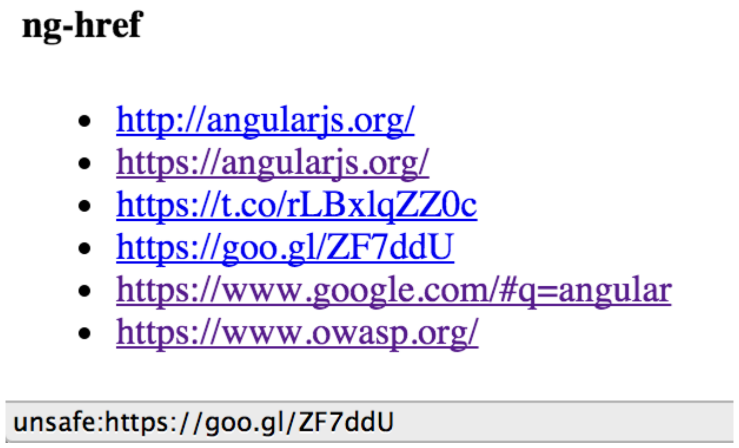
```
var whiteList = /(https\:\/\/[a-z\.]+\.(com|net|org))/;


app.config(['$compileProvider',
  function ( $compileProvider ) {
     $compileProvider.aHrefSanitizationWhitelist(whiteList);
  }
]);
```

ng-href

```
<h3>ng-href</h3>
<ul>
   <li ng-repeat="link in vm.links">
       <a ng-href="{{link}}">{{link}}</a>
   </li>
</ul>
```

unsafe:


ng-href

- http://angularjs.org/
- https://angularjs.org/
- https://t.co/rLBxlqZZ0c
- https://goo.gl/ZF7ddU
- https://www.google.com/#q=angular
- https://www.owasp.org/

unsafe:https://goo.gl/ZF7ddU

```
<li ng-repeat="link in vm.links"
    class="ng-scope">
   <a ng-href="https://goo.gl/ZF7ddU"
      class="ng-binding"
      href="unsafe:https://goo.gl/ZF7ddU">
         https://goo.gl/ZF7ddU
   </a>
</li>
```

`linky` filter

Finds links in text input and turns them into html links. Supports http/https/ftp/mailto and plain email address links.

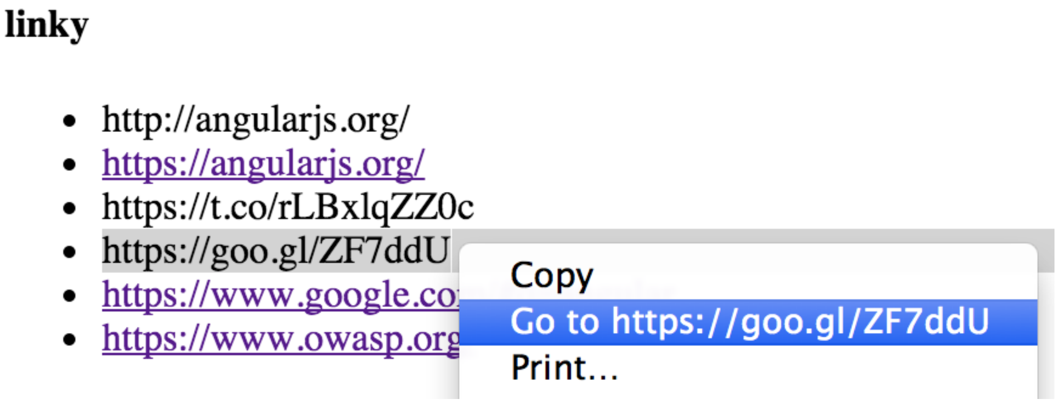
Requires the `ngSanitize` module to be installed.

linky + aHrefSanitizationWhitelist

```
<h3>linky</h3>

<ul>
   <li ng-repeat="link in vm.links"
       ng-bind-html="link | linky">
   </li>
</ul>
```

<a>



linky

- http://angularjs.org/
- https://angularjs.org/
- https://t.co/rLBxlqZZ0c
- https://goo.gl/ZF7ddU
- https://www.google.co
- https://www.owasp.org

```
<li ng-repeat="link in vm.links"
   ng-bind-html="link | linky"
   class="ng-binding ng-scope">
   <a>https://goo.gl/ZF7ddU</a>
</li>
```

Client Side Redirect

```html
<a href="#/redirect?url=https://angularjs.org">
   AngularJS (https)
</a>

<a href="#/redirect?url=http://angularjs.org">
   AngularJS (http)
</a>
```

```
.when('/redirect', {
  template: '<em>redirecting in 3 seconds...</em>',
  controller: function ($scope, $location, $timeout, $window) {
    $scope.url = $location.$$search['url'] || '';

    if ($scope.url.match(whiteList)) {
      $timeout(function () {
        $window.location.href = $scope.url;
      }, 3000)
    } else {
      $location.path('/redirectDenied');
    }
  }
})
```

Demo

Key Takeaways

- OWASP Top 10 applicable to front-end development
- AngularJS has many built-in security features
- Security landscape is continually changing

Thank You!

Questions?