JS-Security

Securing JavaScript based web apps

Thomas Gøytil and Erlend Oftedal
BEKK

Who are we?

Thomas Gøytil

Erlend Oftedal

Agenda

- JavaScript based web apps
- Backbone primer
- HTML templating
- Cross-Site Scripting
- Safe HTML templating
- CSRF and CS#RF
- Clickjacking
- Content-sniffing Demo
- Promiscuous services
- Insecure Storage

Hello JS!

10000000 === 10000001 : false
100000000 === 100000001 : false
1000000000 === 1000000001 : false
10000000000 === 10000000001 : false
100000000000 === 100000000001 : false
1000000000000 === 1000000000001 : false
10000000000000 === 10000000000001 : false
100000000000000 === 100000000000001 : false
1000000000000000 === 1000000000000001 : false
1000000000000000 === 1000000000000001 : true

Motivation for JS-based apps

- Functionality is moved from server side to client side code
- Allows for Rapid development

→ **New security issues arise**

The need for JavaScript frameworks

```
$(document).ready(function() {
    $('#new-status form').submit(function(e) {
        e.preventDefault();

        $.ajax({
            url: '/status',
            type: 'POST',
            dataType: 'json',
            data: { text: $('#new-status').find('textarea').val() },
            success: function(data) {
                $('#statuses').append('<li>' + data.text + '</li>');
                $('#new-status').find('textarea').val('');
            }
        });
    });
});
```

Source: Step by step from jQuery to Backbone

#booster2013 #js

JavaScript Frameworks

- Not just jQuery
- New JS rammeverk for structuring code
- Examples: Backbone, Angular, Knockout, Ember, Agility, Maria, ExtJS, Kendo UI, Spine, Sammy, Stapes....

Single Page Web App

- Browser loads a single HTML-file
- File includes references to JavaScript
- The JavaScript loads data and templates
- Navigation without reload

Single Page Web App - state

- State is maintained via #-URLs

  ```
  http://example.com/#/inbox/32
  ```

- Allows bookmarking
- New alternative feature: `pushState()`
- Change URL without reload

Backbone.js

- Lightweight JS framework
- MVC
- Loads data via REST API
- Built with Ruby on Rails in mind

Backbone.js - Models

```
App.Models.Email = Backbone.Model.extend({
  urlRoot: '/emails/'
  ...
});

App.Models.EmailCollection = Backbone.Collection.extend({
  model: App.Models.Email,
  url: '/emails/'
  ...
});
```

Backbone.js - Routers

```
App.Models.Email  = Backbone.Router.extend({
  routes: {
    "emails"     : "index",
    "emails/:id" : "show_email"
  },

  index: function() {
    var emails = new App.Models.EmailCollection();
    emails.fetch({
      success: function(emails) {
        ...
      }
    });
  },

  show_email: function(id) {
    var email = new App.Models.Email({ 'id' : id });
    email.fetch({
      success: function(email) {
        ...
      }
```

Backbone.js - Views

```
App.Views.NewEmailView = Backbone.View.extend({
    template : JST["templates/email/new"],
    events: {
      "submit #new-email-form" : "send_email"
    },
    initialize : function() {
      this.model = new App.Models.Email();
    },
    render: function() {
      $(this.el).html(this.template());
      this.$("form").backboneLink(this.model);
      return this;
    },
    send_email: function() {
      this.model.save({
        success: function() {
          ...
        }
      });
    }
```

JS HTML Templating

- Mix between JavaScript and HTML
- Typically compiled to javascript (server-side or client-side)
- Templating languages: mustache.js, underscore.js etc.

```
<ul>
  <% emails.each(function(email) { %>
    <li><a href="#/emails/<%= email.get("id") %>"><%= email.get("subject") %></a></li>
  <% }); %>
</ul>
```

```
<ul>
  <% emails.each(function(email) { %>
    <li><a href="#/emails/{{ email.get("id") }}">{{ email.get("subject") }}</a></li>
  <% }); %>
</ul>
```

Underscore.js

- Utilities and HTML templating
- Templating has three functions:
  - <% %> - evaluate code
  - <%= %> - output
  - <%– %> - HTML-escaped output

Playtime

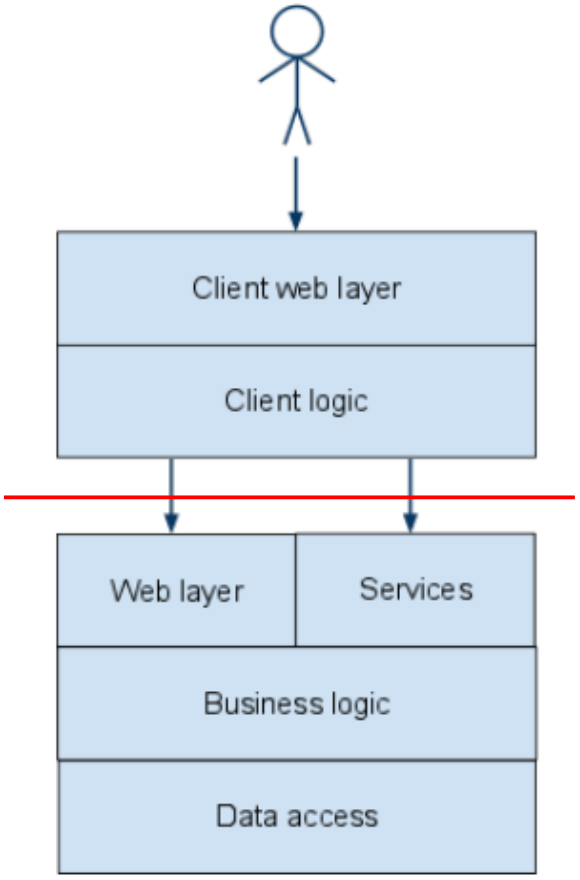
Get the app running

1. Download app:
   http://kurs:jssecurity@kurs.insecurelabs.org
2. bundle install
3. rake db:migrate
4. rails server
5. http://localhost:3000 (alt http://kurs.insecurelabs.org:3000)
6. Brukernavn og passord i INSTALL.txt

Trouble?

- `rake db:reset && rake db:migrate && rails server`

The invisible security barrier



Input validation

- HTML5 supports new kinds of input validation:
  - `<input type="number">`
  - `<input type="email" required>`
  - `<input type="text" required pattern="(\+?\d[- .]*){7,13}">`
- Of course all of these are cosmetic - **usability only**

Playtime

Circumvent input validation on profile page

Hello JS

```
if (a == "Hello" && a == "world") { //Huh?
  document.write("Hello world")
}
```

```
var a = {
  t: false,
  valueOf: function() {
    return (this.t = !this.t) ? "Hello" : "world"
  }
}
```

Circumventing client side validation

- Changing HTML
- Changing javascript data within the browser
- Changing request in proxy
- Changing response in proxy

Playtime

Change another user's message through:

- breakpoint
- proxying request
- proxying response

Reflected Cross Site Scripting

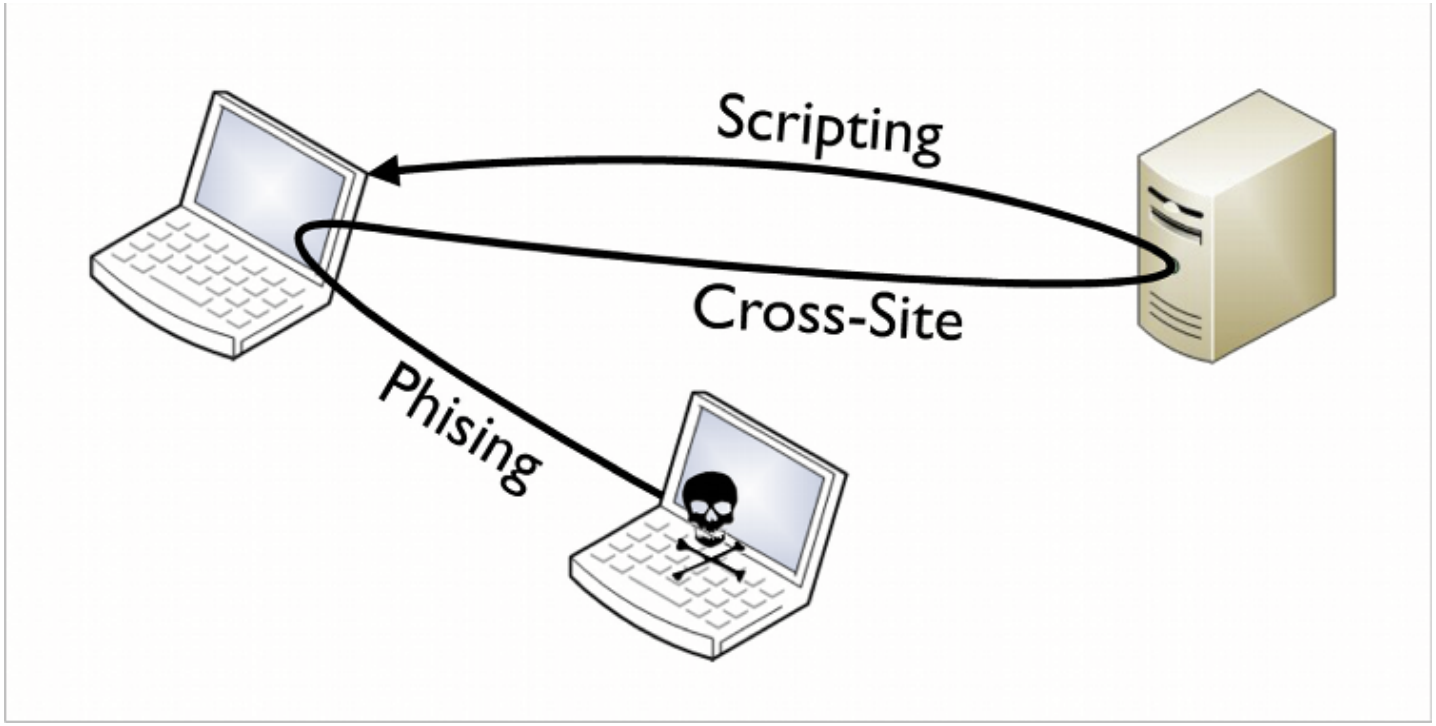


Image source: @johnwilander

Reflected - Example

```
http://example.com/?error=Invalid+name
```

```
<div class="error">Invalid name</div>
```

```
http://example.com/?error=<script>alert(1)</script>
```

```
<div class="error"><script>alert(1)</script></div>
```

Stored/Persistent Cross Site Scripting

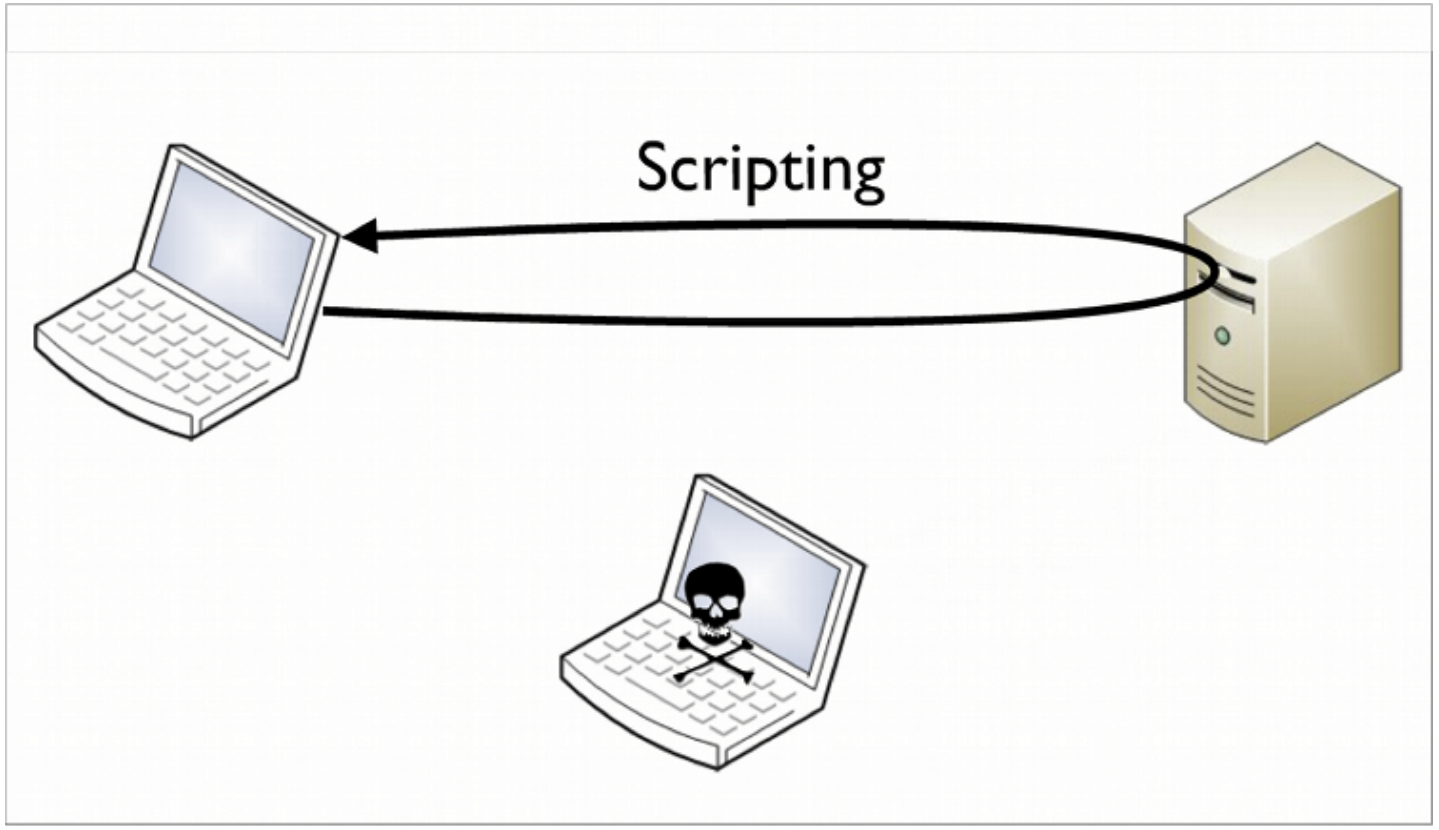


Image source: @johnwilander

DOM-based Cross Site Scripting

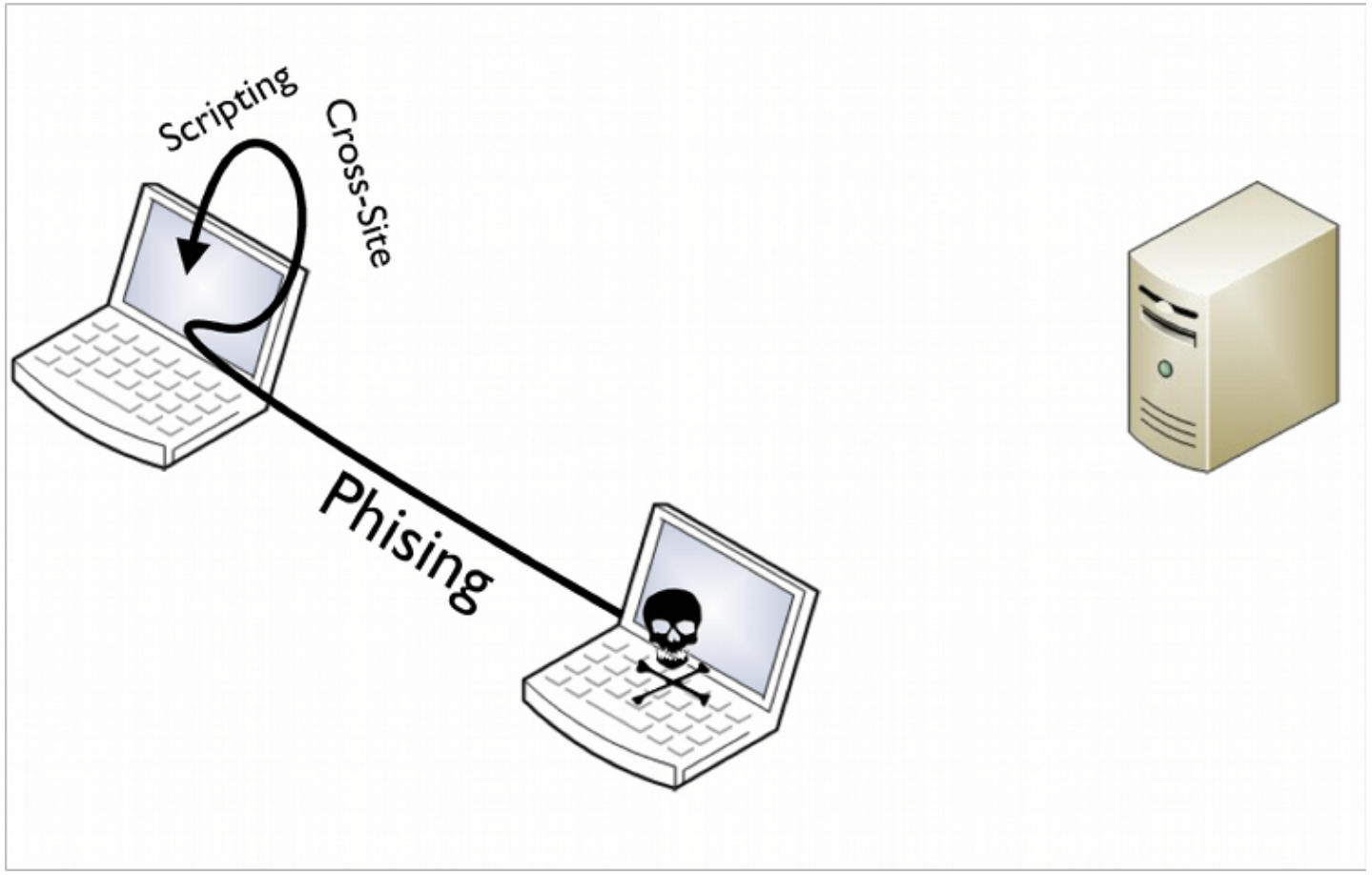


Image source: @johnwilander

DOM-based - XSS

- Occurs in javascript
- Not necessarily visible at the server

```
http://ex.fm/#!/explore/<script>alert("@vlycser");</script>
```

- Insecure handling of input in javascript

DOM-based - XSS - sources

```
document.URL
document.documentURI
document.URLUnencoded
document.baseURI
location
location.href
location.search
location.hash
location.pathname
window.cookie
window.referrer
window.name
++
```

Source: domxsswiki

Twitter september 2010

```
(function(g) {
  var a=location.href.split("#!")[1];
  if(a) {
    g.location=g.HBR=a;
  }})(window);
```

https://twitter.com/#!/owasp

↓

https://twitter.com/owasp

https://twitter.com/#!http://evil.com

↓

http://evil.com

Twitter september 2010

```
(function(g) {
  var a=location.href.split("#!")[1];
  if(a) {
    g.location=g.HBR=a;
  }})(window);
```

https://twitter.com/#!/owasp

↓

https://twitter.com/owasp

https://twitter.com/#!javascript:alert(42)

More: http://blog.mindedsecurity.com/2010/09/twitter-domxss-wrong-fix-and-something.html

Playtime

Find reflected and stored DOM-based XSS

XSS Contexts

- Context #1 - Between tags

```
<div>HERE</div>
```

- Context #2 - HTML tag attributes

```
<input type="text" value="HERE">
```

- Context #3 - Javascript strings

```
<script>
var a ='HERE';
...


<a onclick="return confirm('HERE')">...
```

XSS Contexts

- Context #4 - CSS

```
<style>
body { font-size: HERE; }
</style>
```

- Context #5 - URLs

```
<a href="/?value=HERE">
```

XSS Contexts - mitigation

Rule #1 - between tags	HTML escaping	Convert & to & Convert < to < Convert > to > Convert " to " Convert ' to ' Convert / to /
Rule #2 - HTML attributes	HTML attribute escaping	Escape all except alphanumeric characters using &#xHH;
Rule #3 - JavaScript	Strict JavaScript escaping	Escape all except alphanumeric characters using \uXXXX;
Rule #4 - CSS	CSS escaping and filtering	\XX or \XXXXXX

Read more:OWASP XSS Prevention Cheat Sheet

XSS Contexts - mitigation

Escape all except alphanumeric using %HH

Rule #5 - URLs	URL encoding
Rule #6 - User-provided HTML	Whitelist-based Policy Engine
Rule #0 - Other locations	Don't

Read more:OWASP XSS Prevention Cheat Sheet

DOM-based XSS in the wild

```
<script language="JavaScript" type="text/javascript">
  var qs = location.search.substring(1);
  var nv = qs.split('&');
  var url = new Object();
  for(i = 0; i < nv.length; i++)
  {
    eq = nv[i].indexOf('=');
    url[nv[i].substring(0,eq).toLowerCase()]
      = unescape(nv[i].substring(eq + 1));
  }
</script>
<script LANGUAGE="JavaScript">
  document.write('<SCRIPT LANGUAGE="JavaScript" ' +
   'SRC="'+url['source']+'?"\></SCRIPT\>');
</script>
```

Some unsafe JavaScript

- `eval("...user data...")`
- `setTimeout("...user data...", t)`
- `setInterval("...user data...", t)`
- `new Function("...user data...")`
- `document.write("...user data...")`
- `document.writeln("...user data...")`
- `element.innerHTML = "...user data..."`
- `Range.createContextualFragment("..user data...")`
- `HTMLButton.value = "..user data..."`
- `window.location = "user supplied URI"`
- `a.href = "user supplied URI"`
- `++`

Source: domxsswiki

Unsafe jQuery functions

```
$.after()
$.append()
$.appendTo()
$.before()
$.html()
$.insertAfter()
$.insertBefore()
$.prepend()
$.prependTo()
$.replaceAll()
$.replaceWith()
$.unwrap()
$.wrap()
$.wrapAll()
$.wrapInner()
$.prepend()
```

Source: http://twitpic.com/95n3ak

Safe jQuery functions

- `$.text()`
- `$.attr() - unless attr is JS event handler`

jQuery encoder

- $.encoder.canonicalize()
- $.encoder.encodeForCSS()
- $.encoder.encodeForHTML()
- $.encoder.encodeForHTMLAttribute()
- $.encoder.encodeForJavaScript()
- $.encoder.encodeForURL()

Source: https://github.com/chrisisbeef/jquery-encoder

HTML JavaScript Templates

- What kinds of coding/output-possibilites does it have?
- Does it escape input?
- What kinds of escaping?
- Is the escaping context based?

Underscore.js

- Tags:
  - <% %> - evaluate code
  - <%= %> - output
  - <%- %> - HTML-escaped output
- Escaping

```
_.escape = function(string) {
  return (''+string).replace(/&/g, '&amp;').
                     replace(/</g, '&lt;').
                     replace(/>/g, '&gt;').
                     replace(/"/g, '&quot;').
                     replace(/'/g, '&#x27;').
                     replace(/\//g,'&#x2F;');
};
```

This is all well and good as long as...

- ... you are not outputing inside javascript event handlers.

```
<button onclick="return confirm('Really delete <%- model.title %>')">Delete</button>
```

```
<button onclick="return confirm('Really delete &#x27;);alert(&x27;XSS')">Delete</button>
```

- ... you are not using quote-less attributes:

```
<img title=<%- model.title %>  ... >
```

```
<img title=monkey onmouseover=alert(/XSS/.source) ... >
```

- ... you are not outputting data inside `style` attributes or tags
- ... you are not outputting data inside `script` tags

For more info - see the OWASP XSS Prevention Cheat sheet

helmet.js - an experiment

- Code on github: https://github.com/eoftedal/helmet.js
- <% %> - evaluate code
- <%- %> - unescaped/raw output
- <%= %> - **contextually** escape output or refuse output
- Playground at: http://research.insecurelabs.org/helmet.js

helmet.js - bypass

```
<a
  href=<%=url%>
  title="Buy <%=number%> at <%=price%> = $<%=cost%>/month
  AND SAVE $$$">BUY NOW</a>
```

```
{"url":"","number":42,"price":"onmouseover","cost":"=alert(1)/"}
```

```
<a href="title="Buy" 42="" at="" onmouseover="$=alert(1)//month" and="" save="" $$$"="">BUY NOW
```

Contributor: @steike

#booster2013 #js

helmet.js - bypass

```
<svg>
<a xmlns:xlink="http://www.w3.org/1999/xlink" xlink:href="#">
<set attributeName="xlink:href" begin="0s" to="<%= url %>" /><circle r=40>
</a>
</svg>
```

```
{"url":"javascript:alert(1)"}
```

Contributor: @0x6D6172696F

#booster2013 #js

Content Security Policy

```
Content-Security-Policy: default-src 'self'; script-src 'self' *.googleapis.com
```

- Upcoming standard
- Fits well with single page web apps
- Server instructs browser through header (or meta tag in 1.1)
- By default disallows the unsafe versions of `eval/setTimeout/setInterval/new Function`
- By default disallows inline CSS and JavaScript
- Allows developers to specify which domains scripts, images, videos etc. can be loaded from
- Supported in Chrome and Firefox (rumored but not found in IE10)
- Test your browser: http://csptesting.herokuapp.com/

Content Security Policy - directives

- `default-src` - default
- `img-src` - images
- `object-src` - flash, java etc.
- `connect-src` - xhr, websockets etc.
- `media-src` - audio and video
- `frame-src` - iframes
- `font-src` - fonts
- `script-src` - scripts
- `style-src` - styles
- Origin: `https://*.google.com:443`, `www.bekk.no`
- `'none'`, `'self'`, `*`
- `'unsafe-inline'`, `'unsafe-eval'`

Content Security Policy - debug

- `report-uri` - browser sends violation reports to this url

JSON

```
{"csp-report": {
    "document-uri":"http://localhost:3000/",
    "referrer":"",
    "blocked-uri":"self",
    "violated-directive":"eval script base restriction",
    "source-file":"http://localhost:3000/assets/jquery.js?body=1",
    "script-sample":"call to eval() or related function blocked by CSP","line-number":565
}}
```

Playtime

Try previously found XSS with CSP on

XSS auditor

- XSS protection built into browser
- Supported by IE and Chrome
- Header:

```
X-XSS-Protection: 1; mode=block
```

- Compares tags in URI with response body
- First version in IE8 would block result if searching for script
- XSS auditors have actually lead to other security problems

Playtime

Try previously found XSS with CSP off and XSS auditor on

Content Sniffing

- Browser secondguesses Content-Type header
- Looks at reponse content, URI and also tag that initiated the request
- An attacker can trick the browser into guessing the wrong Content-Type
- Example: GIFAR
- Both a valid GIF and a JAR

Playtime

Trick server into interpreting json as HTML

Avoiding Content Sniffing

- Disable content sniffing:

```
X-Content-Type-Options: nosniff
```

- Have browser prompt for download:

```
Content-Disposition: attachment; filename=data.json
```

Playtime

Trick server into interpreting json as HTML

Cross Site Request Forgery (CSRF)

- User visits attacker's or hacked site
- Site initiates request towards a victim site where the user is logged in
  - For GET request use <img> tag
  - For POST request, use hidden form. Javascript to post form
- Data is changed/deleted or settings altered
- Request comes from a given user
- Examples:
  - Change DNS of home router
  - Post to twitter or Facebook
  - Request application on internal network

CSRF - Example

CSRF - Real-life examples

- @homakov posted a blog post with CSRF vulnerabilities in:
  - github
  - slideshare
  - vimeo
  - bitbucket
  - heroku
- Heroku bug:

```
<img src=xxx/update?site_name[name]=yyy>
```

- ... rename site from xxx.heroku.com to yyy.heroku.com

CSRF + JSON

```
<html>
  <form action="http://example.com" method="post" enctype="text/plain">
    <input name='{"a":1,"b":{"c":3}, "ignore_me":"' value='test"}' type='hidden'>
    <input type=submit>
  </form>
</html>
```

```
{"a":1,"b":{"c":3}, "ignore_me":"=test"}
```

http://blog.kotowicz.net/2011/04/how-to-upload-arbitrary-file-contents.html

CSRF + Cross Domain XHR

```
function fileUpload(url, fileData, fileName) {
 var fileSize = fileData.length;
 var boundary = "xxxxxxxxx";
 var xhr = new XMLHttpRequest();
 xhr.open("POST", url, true);
 xhr.setRequestHeader("Content-Type",
   "multipart/form-data, boundary="+boundary);
 xhr.setRequestHeader("Content-Length", fileSize);
 var body = "--" + boundary + "\r\n";
 body += 'Content-Disposition: form-data; name="contents"; filename="'
   + fileName + '"\r\n';
 body += "Content-Type: application/octet-stream\r\n\r\n";
 body += fileData + "\r\n";
 body += "--" + boundary + "--";
 xhr.send(body);
 return true;
}
```

http://blog.kotowicz.net/2011/04/how-to-upload-arbitrary-file-contents.html

Playtime

Trick victim into posting message

CSRF prevention

- Generate random token and put in session
- Send token to browser
- Never make changes on GET requests
- Every PUT/POST/DELETE request to JSON API includes token
- Token is checked on server, and reject if invalid
- → Attacker site does not know the token value

```
$("body").bind("ajaxSend", function(elm, xhr, s){
  if (s.type === "POST" || s.type === "DELETE" || s.type === "PUT") {
    xhr.setRequestHeader('X-CSRF-Token', authentication.csrf_token);
  }
});
```

Playtime

Enable CSRF protection, and retest CSRF attack

CS#RF

- Does a hash change make your app change data?
- Open document in edit mode:
  `http://conference.cfn/#talks/1/edit`
- Delete document:
  `http://conference.cfn/#talks/1/delete`
- Circumvents CSRF-protection - the app will actually send the token

CS#RF - why does it work?

1. Browser opens url
2. JS framework bootstraps (this allows bookmarking)
3. JS framework processes route
4. CSRF token is included in ajax request
5. PWN

Image:http://wiki.exim.org/NigelMetheringham/HowSecuritySystemsFail

#booster2013 #js

Playtime

Enable CSRF protection and try to find a CS#RF attack

CS#RF - protection

- A hash change should not cause changes on the server
- Bring up delete dialog on:
  `http://conference.cfn/#talks/1/delete`

Clickjacking

- Attacker sites brings up victim site in hidden iframe
- User visits attacker's site
- User clicks on attacker's site, but actually clicks inside hidden iframe

Clickjacking

Start game: 

Playtime

Slette melding via clickjacking

Clickjacking prevention

- Prevent page from being showed in iframe:

```
X-Frame-Options: DENY
```

- Optionally: `SAMEORIGIN` or `ALLOW-FROM`

Playtime

Enable clickjacking protection and retest

Stealing JSON data

Avoiding

- Prevent JSON response from being showed in iframe:

```
X-Frame-Options: DENY
```

- Have browser prompt for download when accessing JSON data directly:

```
Content-Disposition: attachment; filename=data.json
```

Playtime

Test security headers for JSON responses

Hello JS!

```
$=[$=[]][(!$+'')[-~-~-~$]+({}+$)[+!'']+($$=(!''+$)[+!''])+(_=(!+$+$)[+$])],$()[(!$+$)
        [+!'']+(!$+'')[-~-~$]+(!''+'')[-~-~-~$]+$$+_](+!'')
```

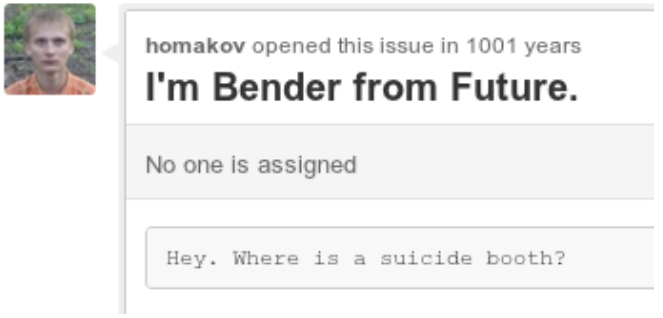
```
alert(1)
```

Promiscuous services

- Is your service showing too much?
- Does it allow others to touch its privates?

Promiscuous services - Mass assignment

- Change fields not available through UI
- Send JSON request with unexpected fields



Playtime

Post message as different user

Promiscuous services - Fixing

- Limit exposed fields in reponse
- Ignore unwanted fields in request

Misplaced data and Insecure storage

- Leave sensitive data out of URLs
  - Password
  - Session ids
  - OAuth access tokens
- Users can see what you send to the client
- Don't store sensitive data on the client side (e.g. local storage)
- Example: Session data in signed cookie to avoid server side session

Rails 3 - Example

Rails 3 uses digitally signed cookies as the default store for sessions. Digitally signed cookies cannot be easily tampered with, but....

users can read the data that is being saved.

Source: andylindeman.com

Some final thoughts on frameworks

- If you use a framework...
- Keep it up to date!
- Security flaws may be discovered at a later time

- Yahoo! 0-Day - 13. jan 2013
- sessvars.js
- Security update May 17, **2008** - Sanitizer added to prevent eval() of malicious code

Security summarized

- Security belongs on the server side
- Only exception is XSS protection
- Secure your JSON services:
  - Access control
  - Mass assignment
  - CSRF
- Remember common web app security mechanisms like clickjacking protection