Your Script in My Page: What Could Possibly Go Wrong?

Sebastian Lekies (@slekies) / Ben Stock (@kcotsneb)

Agenda

The Same-Origin Policy

Cross-Site Script Inclusion (XSSI)

Generalizing XSSI

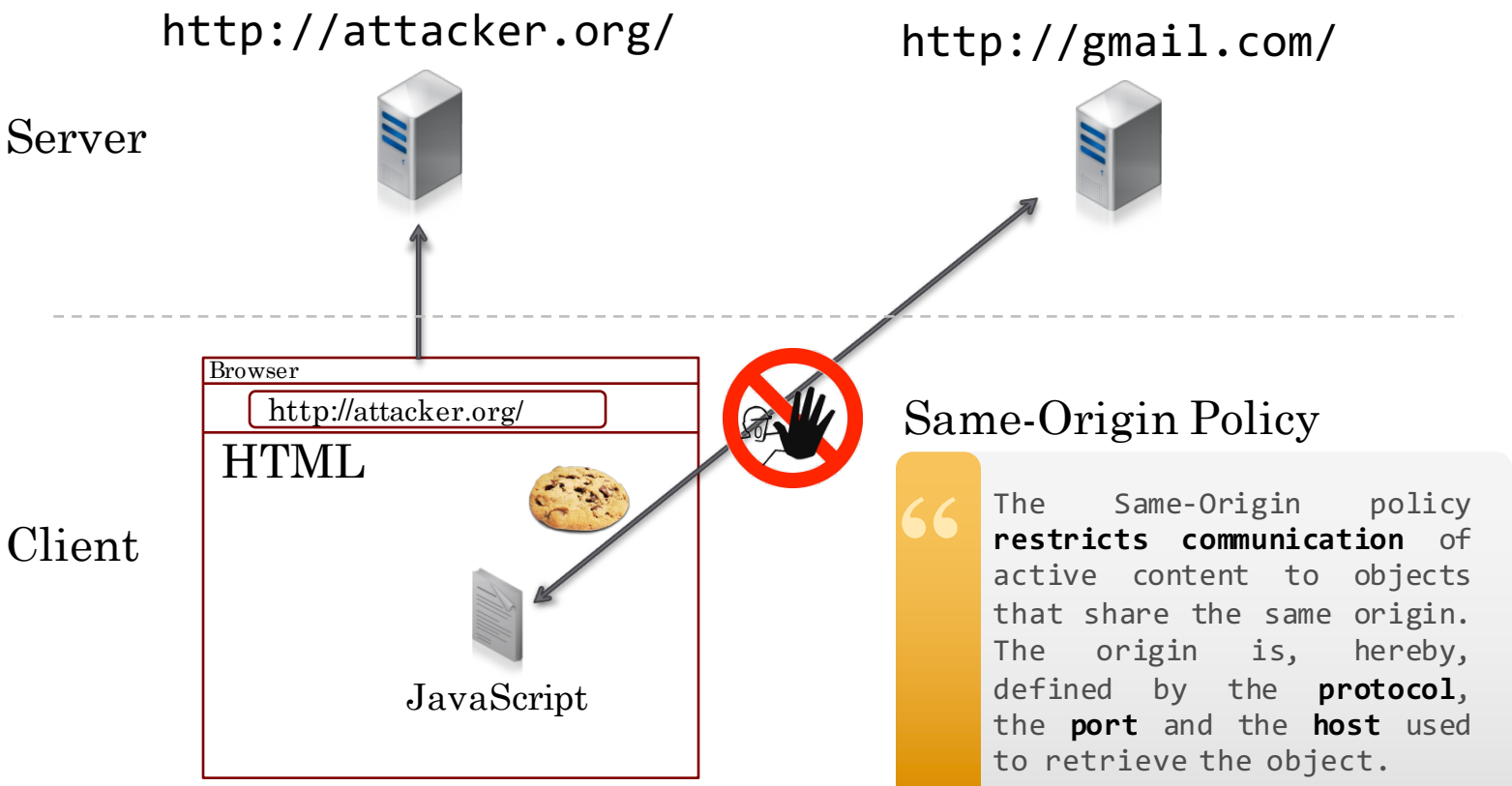
- Dynamic JavaScript files
- Leaking sensitive data from a JS file

Empirical Study

- Methodology
- Results

Conclusion

The Same-Origin Policy



Same-Origin Policy

> The Same-Origin policy **restricts communication** of active content to objects that share the same origin. The origin is, hereby, defined by the **protocol**, the **port** and the **host** used to retrieve the object.

The Same-Origin Policy for JavaScript

Inclusion of third-party scripts necessary

- Advertisement, jQuery, …

Same-Origin Policy relaxed for script inclusion

Included code inherits origin of including site

- both work on same global scope

JSON aka JavaScript Hijacking (2006)

https://attacker.org

Browser

http://attacker.org/

```
<script>
  // Override Array constructor.
  function Array() {
    // Steal data here.
  }
</script>

<script src="//gmail.com/contact.json”></script>
```

https://gmail.com

contacts.json

```
[
  [
    "ct“,
    “John Doe",
    "foo@gmail.com“
  ],
  [
    "ct“,
    “Jane Doe",
    "bar@gmail.com“
  ]
]
```

Cross-Site Script Inclusion

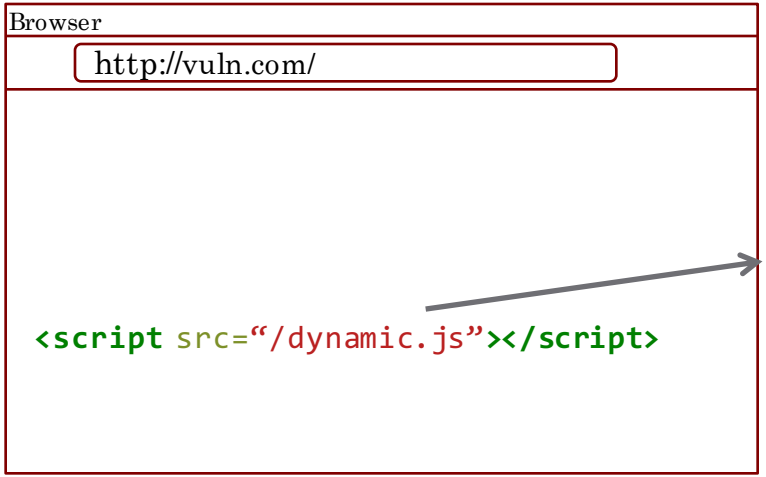
Previous attacks enabled by browser quirks

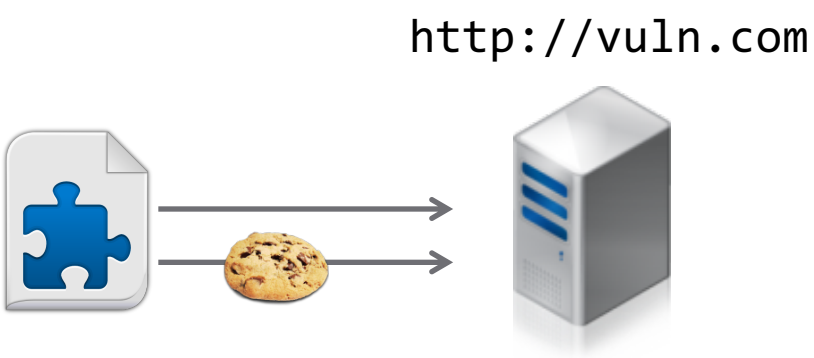
Idea: find other ways to leak private data

- Are there dynamic JavaScript files?
- If so, do these files contain user data?
- Can this data be leaked in a similar way?

Methodology

Detection of dynamic JavaScript files





Methodology

Registered accounts with 150 popular sites

We investigated each site by…

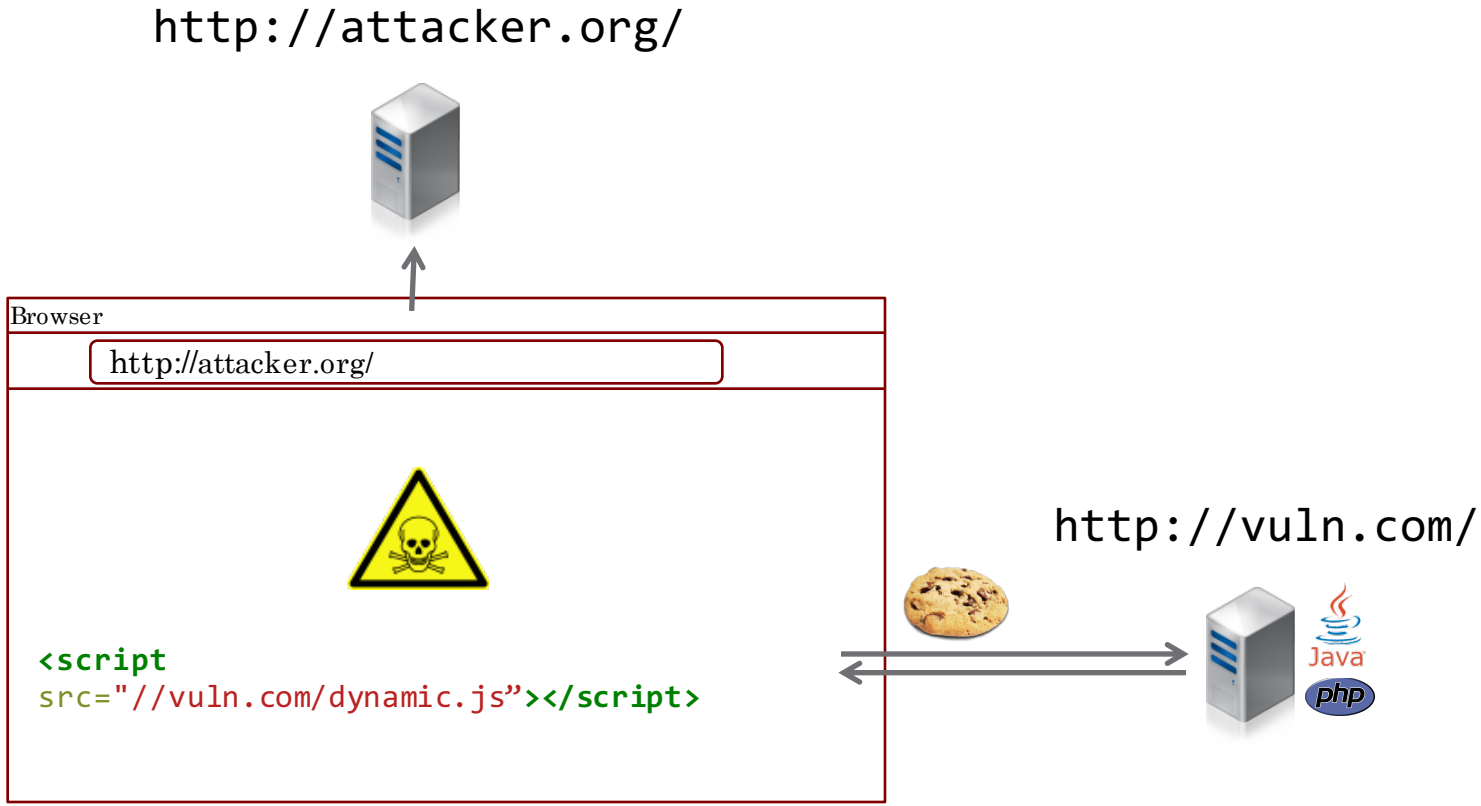
- …seeding the accounts with personalized data
- …thoroughly interacting with the site with our extension
- …manually investigating the dynamic scripts

Empirical Study

Are there JavaScript files that contain user data?

	No. of Domains
Total data set	150
Dynamic scripts based on cookies	49
Contained unique identifiers	34
Contained other personal data	15
Contained CSRF or auth tokens	7

Cross-Site Script Inclusion



Cross-Site Script Inclusion

Leaking data stored in global variables

dynamic.js

```
// local variable at top level
var first_name = "John";

// variable missing the "var" keyword
last_name = "Doe";

// global variable
window.user_email = "john@doe.com";
```

attacker.js

```
console.log(first_name);  // John
console.log(last_name);   // Doe
console.log(user_email);  // john@doe.com
```

More examples: http://sebastian-lekies.de/leak/

Cross-Site Script Inclusion

Leaking data via global functions

dynamic.js

```
function example() {
  var email = "john@doe.com";

  window.MyLibrary.doSomething(email);
};

example();
```

attacker.js

```
window.MyLibrary = {};
window.MyLibrary.doSomething = function(email) {
  console.log(email);
};
```

More examples: http://sebastian-lekies.de/leak/

Empirical Study - Analysis

Can data within JavaScript files be leaked across origin?

	No. of Domains	Exploitable
Dynamic scripts based on cookies	49	40
Contained unique identifiers	34	28
Contained other personal data	15	11
Contained CSRF or auth tokens	7	4

DEMO

~~a.k.a. we are feeling lucky~~

~~Perfect timing for an unplanned downtime~~

More perfect timing for a reboot

Empirical Study - Case Studies

Reading emails subjects and senders

- An email provider previewed the last 5 emails on their main page
- Subject, sender, date and msgId was provided through a dynamic script

XSSI -> CSRF -> XSS -> Facebook post

- A news site hosted a script containing the CSRF token
- The CSRF token enabled us to send profile change requests
- In the profile page there was a XSS
- A Facebook auth token was stored inside a cookie

Taking over an account at a file hoster

- Utilized an Ajax driven Web UI
- An authentication token was required for these XHRs
- The token was provided inside a script file

Preventing XSSI Vulnerabilities

Our attacks are not based on browser-quirks

- Hence, they cannot be fixed on a browser level
- It is very difficult to craft a dynamic script not prone to the attack

Prevent script files from being included by a third-party

- Solution 1: Strict referrer checking (error-prone)
- Solution 2: Use secret tokens

Separate JavaScript code from sensitive data

- Create static JS files and load data dynamically at run time
- The data service can be protected via the SOP

XSSI and Content Security Policy

Recap: CSP is a mechanism for preventing XSS

- …by white listing trusted JavaScript
- …requires all inline scripts to be externalized into script includes

Dynamic inline scripts are not prone to XSSI

- Externalizing the script makes it vulnerable to XSSI
- Do not blindly move script to external files

CSP might make XSSI more wide-spread

Conclusion

We investigated the security of dynamic JavaScript files

- Dynamic generation of JS is wide-spread
- Many dynamic JS files include information based on a user's session
- Data contained inside script files can be accessed across origins

We conducted a study on 150 popular sites

- One third of these sites use dynamic scripts
- 80% of these sites were vulnerable to XSSI
- Consequences range from privacy issues up to full account compromise

Introducing CSP will likely make the problem worse

Questions?

Sebastian Lekies
@slekies

Ben Stock
@kcotsneb