JSON RPC

Cross Site Scripting and Client Side Web Services

Steffen Meschkat
Google

2006-12-28

Browser Side Web Service Access — Motivation

AJAX

- moves application functionality to the browser,

SOA

- makes library functionality available remotely,

Both together

- browser side code uses remote services.

But wait

- remember cross site scripting security?

Cross Site Script Access — Definitions

Same Origin Policy

- Script can access content from the **same origin** of the page it runs in.

And the Origin is ...

- determined by the location of the HTML of the page, not by the location of the script,
- the HTTP **Server**, i.e. the triple of protocol, hostname, port,
- also called the **Site**, though this is a vague term,
- in special situations, the **Domain**; cf. `document.domain` property; `domain` and `secure` cookie fields.

Cross Site Script Access Restrictions

Script in a page **cannot** access data *from* another site:

- no windows, `IFRAME`s, `XMLHttpRequest`s, cookies.

Script **can** send requests *to* another site:

- dynamically load `IFRAME`, post `FORM`, load `IMG`,
- but can't read the response (*almost* not).

Reason

- No restriction on static web functionality.
- Coexists well with domain-bound authorization.
- Assumes there is nothing to hide in the same page.

This is why script injection attacks are effective.

Cross Site SCRIPT Elements

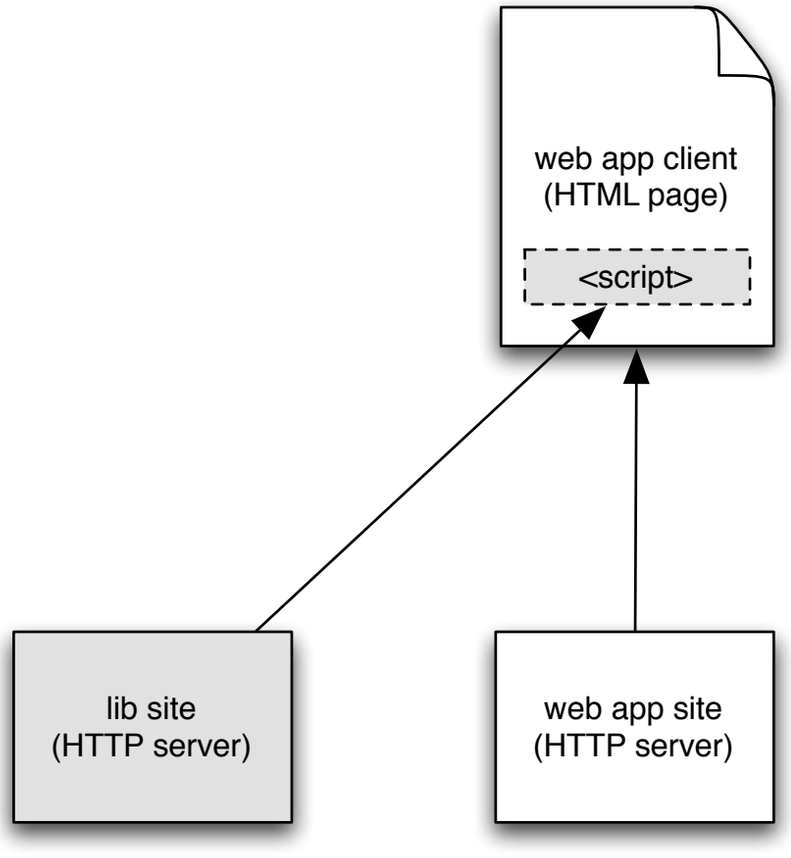
Arbitrary scripts can be part of an HTML page

- Even from foreign sites, just like IMG elements.
- Even when dynamically created by script; they are loaded and executed.

Why isn't that forbidden?

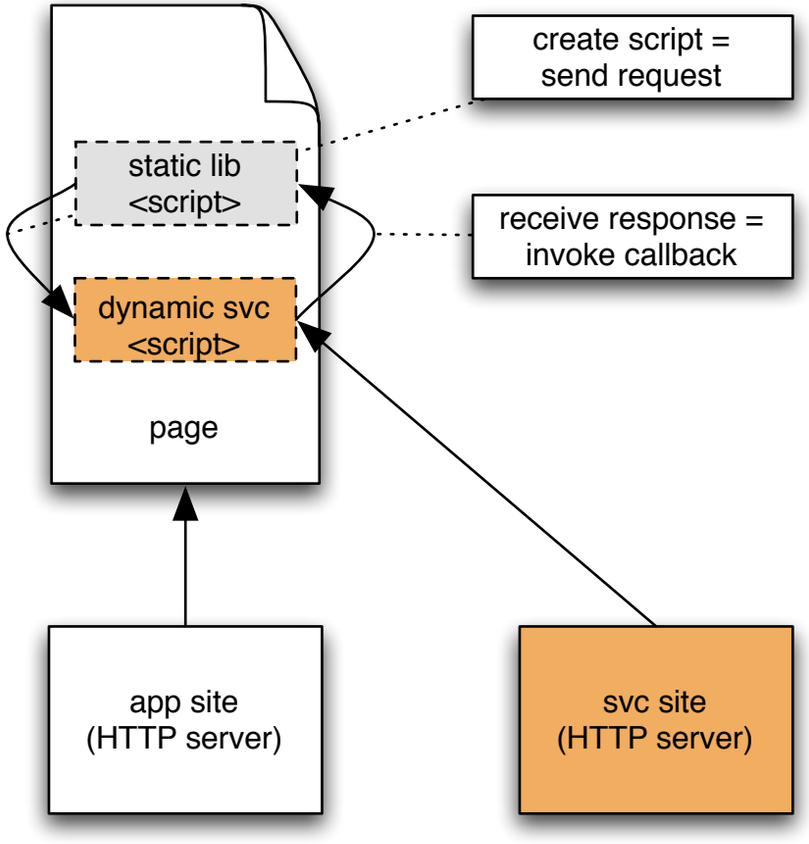
- Script usually contains no data, only code,
- code isn't as sensitive as data; e.g., not personalized,
- can only be *executed*, not *read* (not directly),
- only script is loaded into a SCRIPT element, not HTML.

Cross Site Libraries



- Page from application site
- loads script library from library site
- and uses functionality provided by the library.
- E.g. Google Maps API.

Cross Site Services



Sequence:

1. Page registers callback to receive result.
2. Page creates `SCRIPT` element.
3. Page encodes request and callback in script URL.
4. Server executes service request.
5. Server creates callback invocation with result data as argument.
6. Page loads and executes script, invoking the callback.

Remarks

Cross Site Service

- also known as *JSON with Padding* (JSONP),
- but any js expression can be used, not just JSON.

Cross Site Library

- often complementary to a cross site service,
- e.g. for formatting, convenience methods, caching.

Example: Google Maps API — Geocoding Service

What is it?

- Part of the **Google Maps API**.
- Launched earlier this year.
- http://www.google.com/apis/maps/documentation/

Client Code

```
var map = new GMap2;
var geocoder = new GClientGeocoder;
geocoder.getLatLng("berlin", function(latlng) {
  map.setCenter(latlng);
});
```

Example: Google Maps API — Geocoding Service

Request URL

```
http://maps.google.com/maps/geo?q=berlin
&output=json&callback=cbstore.cb1&key=xxx
```

Response Body

```
cbstore.cb1({
  name: "berlin",
  Status: { code: 200, request: "geocode" },
  Placemark: {
    address: "Berlin, Germany",
    Point: {
      coordinates: [ 13.41156, 52.523533, 0 ]
    }
  }
});
```

Implementation

Parameters and Abbreviations

```
var cbstore = {};
var counter = 0;
var base = 'http://www.google.com/maps/geo';
var encode = encodeURIComponent;
```

We disregard naming requirements for the sake of the example.

Implementation

```
function send(name, value, callback) {
  var id = 'cb' + ++counter;
  var timeout = setTimeout(function() {
    cleanup(id, s);
    callback(null);
  }, 2000);
  cbstore[id] = function(data) {
    cancelTimeout(timeout);
    cleanup(id, s);
    callback(data);
  };
  var s = document.createElement('script');
  document.body.appendChild(s);
  s.src = base + '?callback=cbstore.' + id +
     '&' + encode(name) + '=' + encode(value);
}
```

Implementation

Cleanup helper function

```
function cleanup(id, s) {
  delete cbstore[id];
  setTimeout(function() {
    document.body.removeChild(s);
  }, 0);
}
```

Implementation

Glue code between service library function and API

```
GClientGeocoder.prototype.getLatLng =
    function(address, callback) {
  send('q', address, function(data) {
    if (!data || data.Status.code != '200') {
      callback(null);
    } else {
      var c = data.Placemark.Point.coordinates;
      callback(new GLatLng(c[1], c[0]));
    }
  });
};
```

End of example.

Implementation Details

Concurrency

- Unique callback per request.
- Global id counter and callback map.

Reentrancy

- Assignment to `src` the last step in `send()`.

Robustness

- Invoke `callback()` last, so client code can't break library.
- Avoid suicidal script elements.

Sanity

- Invoke `callback()` with error indication after timeout.
- Clean up resources after use.

Authentication and Authorization

Needed for access to personalized data

- Access to personalized services is authorized for combination of *application and user* to access personal data.
- This fact is shadowed in personalized *applications*, because the application is trusted by default.

Cookie based authorization doesn't work

- because it's based on *where* the request is sent *to*,
- not *who* is sending it.

Must use request parameter based authorization.

- Authorization token must be established based on application, service, and user credentials.

API Standardization

Probably failed: JSONRequest object

- Security by coincidence: POST request, JSON response.
- Unnecessarily ties transport authorization to data format.
- `JSONRequest` : `SCRIPT` = `XMLHttpRequest` : `IFRAME`
- Relies on application level authorization.

Promising approaches

- HTTP response header field that authorizes cross site access.
- access file like `robots.txt`
- just go ahead using `SCRIPT`

Summary: Cross Site Access

Restrictions

- Writing *to* foreign sites is unrestricted.
- Reading *from* foreign sites *is* restricted, but not impossible.

Channels

- `IFRAME`, `A`, `FORM`, `IMG` can send cross site requests; responses are shown to the user but not accessible by script.
- `XMLHttpRequest` can't even send cross site *requests*, maybe because there is no way to display the response.
- `SCRIPT` elements can load and *execute* scripts in response to cross site requests.

Consequence

- Dynamic script elements enable client side support for service oriented application architecture.