# Security



## JSON hijacking



Wed, 2010-01-27 20:20 — Heine

| | |
|---|---|
| Project: | Drupal core |
| Component: | General security hardening |
| Category: | bug report |
| Priority: | normal |
| Assigned: | Damien Tournoud |
| Status: | needs team response |

Jump to:
Most recent comment
Add new comment

**Description**

nuts and bolts: evil.com has JS that overrides the array / object constructor, tricks you into visiting a JSON returning URL on victim.org via <script src="http://victim.org/json". As the JSON data is instantiated as an array, and calls the overridden constructor, this allows evil.com access to the sensitive data.

We could patch Drupal.parseJSON & drupal_json to wrap the string in comments to prevent this.

Unfortunately, that will break scripts and modules that do not use parseJSON or drupal_json.

Thoughts?

Core

## Comments

### #1

Wed, 2010-01-27 20:52 — Heine

| | | |
|---|---|---|
| Project: | Core_legacy | » Drupal core |

Related issue for D7 on using jquery's json parser: http://drupal.org/node/673884

Coltrane & pwolanin also proposed the use of tokens on JSON retrieving URLs (atm #ahah callbacks do this already when form_get_cache is used).

delete edit reply

### #2

Wed, 2010-01-27 20:54 — greggles

The original whitepaper on this is http://www.fortify.com/servlet/downloads/public/JavaScript_Hijacking.pdf it's pretty dense, but quite good.

delete edit reply

### #3

Wed, 2010-01-27 21:37 — pwolanin

A suggested mitigation is to make ajax via POST by default, but this has a number of downsides in terms of ease of coding callbacks, performance, etc.

Also, the POST mitigation doesn't prevent other (new/unknown) CSRF attacks - a token provides generic defense.

delete edit reply

### #4

Wed, 2010-01-27 21:54 — greggles

From jresig:

> So looking at that PDF it's absolutely ancient (2007!). These days, as

> far as JSON-injection security is concerned, we're very well off. We now pre-analyze all incoming JSON to make sure that it is actually JSON and, if not, reject it outright. Additionally we use the native JSON parser provided by the browser (which should provide an additional layer of protection).
>
> I remember laughing at the concerns here back when they came out and it really hasn't changed much - if the server has been hacked and is sending malicious data then having it be sent via JSON is the least of your concerns - XSS attacks are much more problematic.

delete edit reply

#5 Wed, 2010-02-03 19:28 — Heine

The issue descibed is not about Drupal.ParseJSON getting invalid or 'dangerous' data, it's about external sites that can fetch and access JSON data via CSRF, leading to information disclosure.

delete edit reply

#6 Fri, 2010-02-12 08:56 — grendzy

I agree this is serious. It suspect it will be impossible to fix in D6 without breaking some contrib modules, though. For D7 there may be more options, but time is pretty short.

It's too bad the author of that paper used such a confusing name for this attack. :-(

Here's a quick list possible countermeasures:

- poison the json by prefixing with while(1);
- require json requests to use POST
- double-submit the session cookie
- use a unique token per request

Also from what I remember of the paper there are use cases where the "hijacking" is actually desirable. A public JSON resource available via GET can be used for mashups, and it can be cached by the browser.

So maybe we need a
function drupal_json($var, $xsrf_safe = FALSE) or something. In D7 drupal_json_output() could change the default to TRUE.

delete edit reply

#7 Wed, 2010-03-10 00:07 — grendzy

OK. I have been trying to steal data from https://security.drupal.org/taxonomy/autocomplete/2/c

The javascript console shows
Error: Invalid label

I think this is because for an associative array, drupal_to_js uses object notation and so the outer {} braces look like a code block to the javascript interpreter. Data structures with an array at the top level might still be vulnerable, though. Also I'm not sure if this behavior can be relied on in all browsers.

delete edit reply

#9 Wed, 2010-05-05 18:00 — drumm

This is a well-documented general vulnerability we won't fix in a stable version of Drupal. Can we make this a public security hardening issue?

delete edit reply

#11 Wed, 2010-08-11 20:47 — bjaspan

The fact that this is considered a security attack that app developers need to defend against pisses me off. From the original paper:

> JavaScript Hijacking allows an attacker to bypass the Same Origin Policy in the case that a Web application uses JavaScript to communicate

> sensitive information. The loophole in the Same Origin Policy is that it allows JavaScript from any website to be included and executed in the context of any other website.

To me, this is a bug in the web browsers. WTF are they doing letting one site override the array/object constructor used by another site's JS? Forget AJAX hijacking, this sounds like it is just a wide open complete failure of the JS security model. I can access and modify any object you operate on, 'cause they all have to be constructed first! Perhaps I'm missing something and this really is AJAX-specific, but I completely fail to see how this should be up to application developers or Drupal to fix.

In any case, I haven't had time to look at this and realistically am not going to have time to spearhead it.

delete edit reply

#12 Wed, 2010-09-22 18:10 — Heine

This has indeed been fixed in the spec and major browsers.

```
<script>
function Array() { alert('got you'); }
var foo = new Array('val', 'val2'); // Alert, override is used.
var foo = ['val', 'val2']; // No alert (this is relevant for JSON).
</script>
```

It is still possible to define setters for objects, but {'foo': 'bar'} cannot be interpreted, so we are safe there to.

Anyone have IE6?

delete edit reply

#13 Wed, 2010-09-22 18:53 — Heine

"It is still possible to define setters for objects, but {'foo': 'bar'} cannot be interpreted, so we are safe there to."

Well, ....

Eg.

```
<html>
<head>
<title>Titel</title>
<script>
Object.prototype.__defineSetter__('foo', function(o) { alert(o); });
</script>
<script src="http://l/core6/a.js"></script>
</head>
<body>
</body>
</html>
```

Where a.js (on another domain) is an array containing objects:

```
[{"foo": 'geheim'},{"foo": 'secret'}]
```

This **does** enable the caller to get the value of known members.

delete edit reply

#14 Wed, 2010-09-22 20:30 — Damien Tournoud

I can reproduce the setter method on v8, but not on SpiderMonkey.

delete edit reply

#16 Sat, 2011-08-06 11:16 — Damien Tournoud

Here is a quick review of the state of the art.

**This type of attack is well-known.** It is very well described by several blog posts (for example [1] and [2]), several books and whitepapers ([3]). As a direct consequence of that, I suggest we treat this issue publicly.

As far as solutions are concerned:

- There seems to be a consensus on the fact that none of the various JSON poisoning techniques (prefixing with while(); enclosing with (), adding control characters, returning objects instead of arrays) is a viable, future-proof solution;
- There also seems to be a consensus that this is a CSRF vulnerability that should be dealt with the anti-CSRF tools that already exists;
- A common technique was to only serve the JSON if the `X-Requested-With: XMLHttpRequest` header is present. It was the technique used in Rails (only for POST requests, Rails has no protection against GET requests) until it was proven a very bad idea [4];
- Both [2] and [3] mention the use of a standard CSRF token as a good solution, and also mention double-cookie confirmation technique (the idea of using the session cookie as a CSRF token)

Here would be my recommended course of action:

- We add a `<meta>` header to every page with a CSRF token;
- We build on the idea of a "Built-in support for security tokens in links and menu router" [5] and extend it to support passing the token in a `X-CSRF-Token` HTTP header; (note that we have to add `X-CSRF-Token` to the `Vary` header too, to make sure that proxy servers handle that correctly)
- We add a beforeSend()/ajaxPrefilter() handler to `drupal.js` so that all the existing Javascript code can continue to work the same way.
- Because this solution is generic and should not change any API, we should be able to backport to Drupal 6, but given that this issue doesn't seem to be dealt with properly very often, it doesn't seem critical to do so.

[1] http://www.thespanner.co.uk/2011/05/30/json-hijacking/
[2] http://blog.archive.jpsykes.com/47/practical-csrf-and-json-security/
[3] http://www.openajax.org/whitepapers/Ajax%20and%20Mashup%20Security.php#S...
[4] http://jasoncodes.com/posts/rails-csrf-vulnerability
[5] http://drupal.org/node/755584

delete edit reply

#17 Sat, 2011-08-06 21:37 — webchick

Given the choice between handling security issues here and in the public queue, I always lean towards the public queue because we can get more eyeballs and testing that way. So as long as the security team is comfortable with that approach (at least 3 people here seem to be), I'd say let's handle it in /issues/drupal.

Thanks for the detailed research here.

delete edit reply