How to upload arbitrary file contents cross-domain

How to upload arbitrary file contents cross-domain - Author not stated, blog.kotowicz.net.

- Published: date not stated
- Original: <http://blog.kotowicz.net/2011/04/how-to-upload-arbitrary-file-contents.html>
- Preserved from: http://blog.kotowicz.net/2011/04/how-to-upload-arbitrary-file-contents.html (live) on 2026-08-09
- Licence: unknown

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

Update: *Since publishing details of this technique it has been used to exploit CRSFable file upload forms on Facebook , Flickr, Imgur, minus.com, Tumblr.com and others. It seems that many file upload forms lack anti CSRF tokens. *

HTML5, together with its sister specifications (XMLHTTPRequest level 2, File API etc.) has a really interesting property when it comes to security. Websites that are coded securely get new tools allowing them to be even more secure. Yet poorly coded websites might be prone to new flavours of attack. **It makes good even better, and bad even worse.**

The best example of this would be the Cross Origin Resource Sharing (CORS) specification commonly known as Cross Domain AJAX. Back in the days, AJAX request could not be sent cross domain - now, in all current browsers, they can. Does it affect security? Sure it does - even Faceboo k got hacked with it. While the specification was designed with security in mind, fully opt-in, introducing new headers and preflight mode, there are sites in the Internet that suddenly got vulnerable once surfers upgraded their browsers.

Continuing the fun with file upload issues in current browers, today I'd like to show you how to upload a file:

- from victim's browser
- with arbitrary filename
- with arbitrary content
- without user interaction
- **.. to another domain.**

This last point is crucial. Before CORS, the attacker might forge a file upload request with Javascript, but he could only upload it to the same domain (so XSS in a victim site was required) -

now he can do it from anywhere. *Disclaimer: *Don't expect a miracle that will exploit every website dealing with file upload. Described method is not likely to be exploited in the wild as it requires the victim application to be vulnerable in a specific way. It's rather a proof of concept of how bad application turns into terrible when HTML5 arrives.**Update: **now it's much more closer to miracle

How does file upload work?

HTTP file uploads are simply POST requests with multipart/form-data content type. Exemplary HTTP request looks like this:

```
POST /crossdomain-upload/vuln/recv.php HTTP/1.1
Host: victim.blog.security.localhost
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:6.0a1) Gecko/20110422 Firefox/6.0a1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Connection: keep-alive
Referer: http://victim.blog.security.localhost/crossdomain-upload/vuln/upload.php
Cookie: PHPSESSID=bdb46b013d71a85f3514c6c6031c4654
Content-Type: multipart/form-data; boundary=---------------------------153930214414330723532119279484
Content-Length: 238

-----------------------------153930214414330723532119279484
Content-Disposition: form-data; name="contents"; filename="hello.txt"
Content-Type: text/plain

Hello, world!
-----------------------------153930214414330723532119279484--
```

In multipart/form-data, every form field (including files) is represented as a MIME part and parts are separated by a arbitrary boundary. Browsers **after user chooses** *the file* populate the filename field with local file name (line 15) and copy file contents into MIME part (line 18).

Server processes this POST request, unpacks the MIME data, extracts files and, for example in PHP, stores the files in temporary directory, filling out $_POST and $_FILES arrays for the application. Simple and elegant.

File upload in Javascript

But what if you'd like to generate a file in the browser without forcing user to choose one from his HDD? This way, we could have control over file name, file contents AND the user wouldn't be bothered with the 'choose file' dialog.In fact, it would be invisible. Is it possible?

Sure! It was for years. Just make a XMLHttpRequest POST request, forming a multipart MIME in Javascript manually.

```
function fileUpload(url, fileData, fileName) {
  var fileSize = fileData.length,
    boundary = "xxxxxxxxx",
    xhr = new XMLHttpRequest();

  xhr.open("POST", url, true);
      // simulate a file MIME POST request.
  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;
}
```

Voila!

Same domain, XSS needed

How can this be used? Let's imagine that we have a website serving up a repository of files shared among a working group of users. Users upload files, others download them to work together on some secret nuclear project. The site follows common development practices, so it is vulnerable to XSS (it must be, it's the top feature!)

Now it is possible to inject a payload that will, on behalf of a user (the cookies will be included) upload any content we want, with any filename. It can be a trojan EXE file, PHP file to be used in LFI attack or malicious PDF file. Even uploading TXT files is considered harmful due to content sniffing. User needs only to visit a given URL.

But - **we need to be on the same domain**, hence requirement for XSS.

Having fun cross origin

ekhm... **no, we don't**. We have CORS. We can post to other domains, so we don't need any XSS, we can host the payload anywhere! The POST request with our forged MIME parts will happily leave victim browser and travel through Internet to its destination, decorated with CORS headers. The application will happily ignore them as it doesn't *speak* CORS.

But wait? Wasn't CORS designed *with security in mind*? Sure it was, there are a few caveats:

- you won't get access to the server response (unless the victim site sends Access-Control-Allow-Origin and other headers)
- cookies, HTTP authentication and custom headers won't be transferred, so application must accept file uploads without authenticating them **- Update: **Now there's a simple way to include credentials

These caveats make the described technique unlikely-to-be-used, but still, there probably are websites skipping authentication when processing file uploads. **For them, HTML5 turns bad into terrible.**

Demo

As always, I've prepared a demo:

http://**victim**.kotowicz.net/crossdomain-upload/vuln/

This is a **victim** site that allows you to upload a file you might need later into your personal web space. You need to be logged in to upload the file, of course, but the authentication is not always checked (a small *"mistake"* by me). You can only see your own files, because the filenames are prepended with your login.

http://**attacker**.kotowicz.net/crossdomain-upload/evil/upload.html

This is the **attacker** page - while being on a different domain, he can simulate file upload with any file name and any file contents to victim page, and the file will be processed. Note that I, knowing your skills, I'm not trusting you with my server ;) - **for security reasons file contents are erased and .txt extension is used for all files.**

This demo works in **Firefox 4+** and **Chrome**. It might be easily modified to work in IE, other browsers are not tested yet.

The source files for this demo are on GitHub.

Final words

Other ways to CSRF file upload were discovered in 2008 by @kuza55 and @pdp. Kuza55 method exploited a browser bug that converted standard form field into a file field (lack of proper escaping by browser engine). PDP used Flash to construct MIME message.

What is worth mentioning - method described in this blog post is "future-proof", it's unlikely to get fixed by browser vendors. It exploits a **CORS specification quirk** that allows sending POST requests without preflight.Originally the specification allowed only GET/HEAD requests to be sent this way, but it allowed POST in 2008. Why? FAQ answers:

>

Why is `POST` treated identically to `GET`? Cross-site `POST` requests have long been possible using the HTML `form` element. Cross-site `POST` requests with arbitrary an `Content-Type` header set have been possible for a long time in Flash.

It's true, one could always send POST request by doing form.submit() with form action being on different domain. However, one **could not send files** this way without user interaction (unless using bug @kuza55 discovered). Now we're able to send files, but cookies are missing, **The rules change** - **and existing applications should adjust**. Most of them won't - and this is exactly why html5 buzzword will be a game changer for web app security in the coming years.
