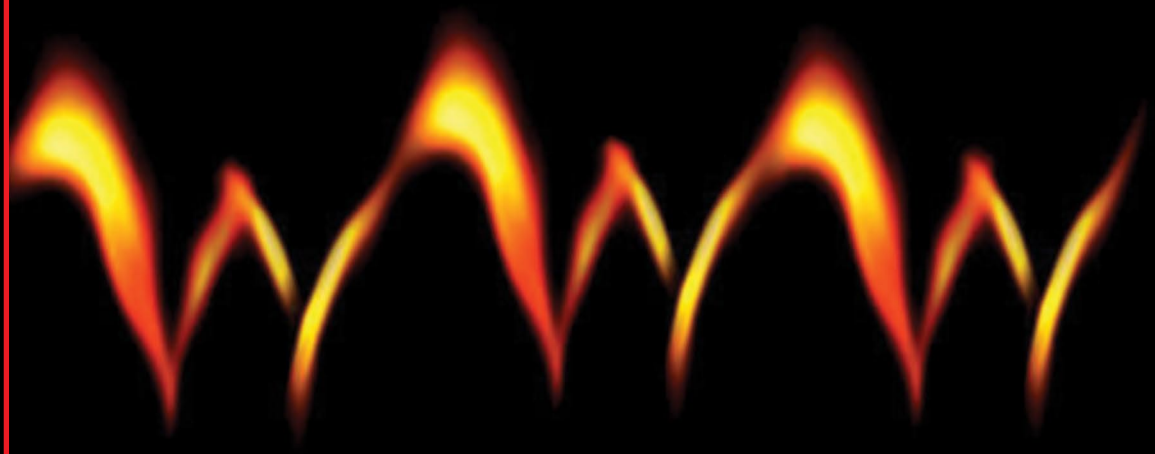
Secure Coding.
Practical steps to defend your web apps.

SANS

Working Papers in Application Security



Cross Site Request Forgery: What Attackers Don't Want You to Know

A Study of Browser Implementations and Security Mechanisms for XMLHttpRequest and XDomainRequest

Written by Jason Lam, SANS Internet Storm Center and Johannes B. Ullrich, PhD SANS Technology Institute

Volume 2.1

SANS

Working Papers in Application Security

Cross Site Request Forgery: What Attackers Don't Want You to Know

Written by
Jason Lam and
Johannes B. Ullrich, PhD

This article is part of a series dedicated to application security and secure coding practices. In coming months, the SANS Institute will release additional articles like this that cover other aspects of application security and secure coding. However, please don't wait until the end of the series to start following these tips. Make a commitment today to ensuring that your applications and code are more secure.

www.sans.org/info/39149

XMLHttpRequest is the backbone of Web 2.0 applications. It is a powerful JavaScript function that allows the flexible creation of HTTP requests. Lately, with Internet Explorer 8, XDomainRequest was released, which extends and refines the creation of HTTP requests in JavaScript. Both functions had a defined impact on the development of Web standards. However, both functions are also frequently cited for their usefulness in attack tools. We will investigate the evolution of these functions and how these functions evolved to mitigate the harm done. We found that security requirements put forward by the standard are not implemented consistently across different browsers. Developers need to be aware of these inconsistencies to protect applications from cross site request forgery.

1. Introduction

With the advent of Web 2.0, XMLHttpRequest (XHR) has become a prominent JavaScript function (Oreilly, 2007). It has existed for some time, but has not been used much until the era of asynchronous JavaScript and XML (AJAX) allowed the function to show its full potential.

XHR, as defined by the World Wide Web Consortium (W3C), included one important security feature from the start (W3C Web APIs Working Group, 2006). This feature, referred to as the "same origin policy", prevents XHR from creating requests against other Web sites than its origin. The origin is the Web site from which the JavaScript was loaded.

The pressure to allow cross domain access is constantly increasing with Web 2.0. Mash-ups, a concept that allows for data from various sites to be consolidated into a single view, is one of the goals of Web 2.0. These requirements conflict with the same origin policy as the one initially formulated for XHR. In response, the XHR definition has been extended by a "Level 2" standard, allowing cross domain access if other requirements are met to protect the user (W3C Web Applications Working Group, 2008). While W3C reacted to the push for a cross domain capability, Microsoft came forward with XDomainRequest (XDR) (Microsoft Corp.), a function similar to XHR Level 2.

To complicate the situation, all these standards are in draft form at this point and rapidly changing. XHR Level 2 is only implemented in beta versions of Firefox and Safari. XDR has recently been released as part of Internet Explorer 8, but much of the documentation found online is still referring to implementations only found in beta versions of Internet Explorer 8.

The risk of a function like XHR is Cross Site Request Forging (XSRF). If unchecked, XSRF forging can be used by a malicious site to send a request on the user's behalf. This request will originate from the user's browsers and include all authentication information the user would have submitted with a request by clicking on a link or submitting a form (Stamos & Lackey, 2006). This attack has been implemented against Web based router admin interfaces and online shopping sites (Kassner, 2008).

XSRF can be done without the use of JavaScript or XHR using standard HTML features like IMG tags (Watkins, 2001). With XHR, its full potential could be released if the same origin policy is not applied correctly or if headers like the "Referer"[1] header are used to hide the real origin of the request (Klein, 2005).

[1] Note that "Referer" was spelled with one "r" on purpose. The HTTP RFC requires this spelling for the "Referer" header.

SANS
Working Papers in
Application Security

Cross Site Request Forgery: What Attackers Don't Want You to Know

A Study of Browser Implementations and Security Mechanisms for XMLHttpRequest and XDomainRequest

Written by
Jason Lam and
Johannes B. Ullrich, PhD

2. XMLHttpRequest

XMLHttpRequest Basics

Functionally, XHR lets the browser perform Hyper Text Transport Protocol (HTTP) requests without user interaction. For example, a page with JavaScript can leverage XHR to request information from another page on the fly. This technique is frequently used by AJAX in order to retrieve information and insert it into an already loaded Web site. For example, Google Maps will use this feature to download additional map images whenever the user pans or zooms the map (Google Inc.).

The XHR object defines six methods: open, setRequestHeader, send, abort, getAllResponseHeaders, and getResponseHeader (Web API Working Group, 2007).

"open" is used to define the method used, the URL and set a username and password. In addition, the developer may indicate if the request should be executed asynchronously or not. The default is asynchronously. However, it is important to realize that the "open" method will just configure the request, not send it.

The "send" method is used to send the request after it has been completely configured.

A simple request will only require the "open" and "send" method. In order to specify any special headers, the "setRequestHeader" method can be used. It is optional, but may be useful in some cases for authentication or to pass additional parameters via custom headers.

The "abort" method can be used to abort a request after it has been sent. Typically this is used to time out requests.

Lastly, the "getAllResponseHeaders" and "getResponseHeader" methods are used to retrieve all headers or particular headers from the response sent back by the server.

If the request succeeds, the response body will be available via the "responseText" or "responseXML" attribute. The status code returned by the server is available via the "status" attribute.

Whenever the status of a request changes, the event listener "onreadystatechange" is called. This function can then be used to retrieve data, or catch errors.

A typical code snippet to implement XHR is shown in Listing 1.

```
<script>
  var element = document.getElementById('myplace');
  xmlhttp=new XMLHttpRequest() ;
  xmlhttp.open("GET", "http://someurl",true);
  xmlhttp.onreadystatechange =
     setcontent(xmlhttp,element_id);
  xmlhttp.send(null);

function setcontent(xmlhttp,element_id) {
  var element = document.getElementById(element_id);
  if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
      element.innerHTML = xmlhttp.responseText;
  }
}
</script>

<html><div id="myplace"></div></html>
```

Listing 1: Sample XHR Code

SANS

Working Papers in Application Security

Cross Site Request Forgery: What Attackers Don't Want You to Know

A Study of Browser Implementations and Security Mechanisms for XMLHttpRequest and XDomainRequest

Written by
Jason Lam and
Johannes B. Ullrich, PhD

XHR Security Concerns

The ability to create HTTP requests does not appear to be all that dangerous at first. Any HTML page may trigger HTTP GET requests by including IMG tags and other tags that cause downloading additional data. POST request can be crafted using traditional JavaScript by adding an HTML form and using the JavaScript "submit" method (Finch, 2008). The way in which XHR differentiates itself is its ability to set arbitrary headers. No other HTML or JavaScript features allows for this. The threat is more sophisticated cross-site-request forging (XSRF) attacks. A cross site request forging attack triggers a request in a browser against a Web site without any intentional user interaction. This is particularly useful if the user is logged in to the target Website. A malicious script may assume the user's privileges and trigger actions using the user's stored credentials. For example session cookies will be sent with the request just like as with any user-initiated request.

Without JavaScript, only GET requests may be sent. With JavaScript, but without XHR, forms may be submitted via GET and POST, but headers cannot be manipulated.

Same Origin Policy

The ability to issue requests is particularly dangerous if it crosses from an untrusted Website to a trusted one. In order to prevent this scenario, XHR implements the same origin policy. Javascript can only be used to trigger requests against the Web site from which it was loaded. The "origin", which identifies the Website, includes the protocol (http or https), the host name and the port number. If implemented correctly, the same origin policy prevents the majority of XSRF attacks (Bhola & Steiner).

In order to launch an XSRF attack, the attacker has to create javascript that is either delivered by the target Website, or via proxies, appears to be delivered by the target Website. One major avenue to inject javascript is cross site scripting. Cross site scripting is a very common vulnerability and should be considered as a serious problem if exploitable for XSRF. Figure 1 shows the victim first establishing a session with Bank.com and is assigned a token in the form of a cookie. Any subsequent request from the user to Bank.com will automatically return the cookie to Bank.com.

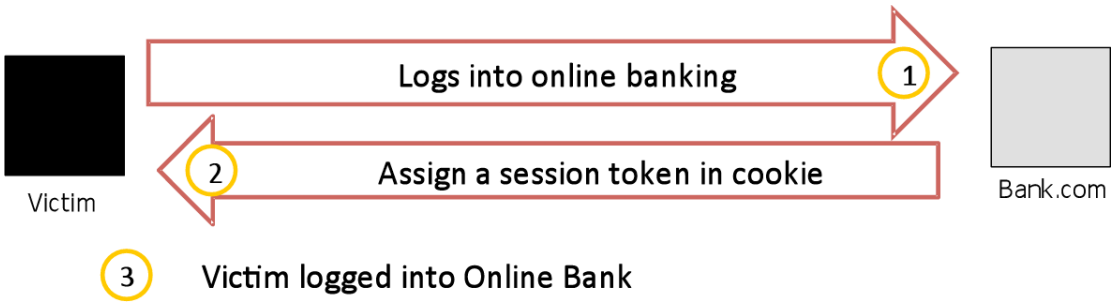


Figure 1. Login process prior to CSRF attack {Dave: please add an "s" to "fail" next to circled 3.}

SANS
Working Papers in Application Security

Cross Site Request Forgery: What Attackers Don't Want You to Know

A Study of Browser Implementations and Security Mechanisms for XMLHttpRequest and XDomainRequest

Written by
Jason Lam and
Johannes B. Ullrich, PhD

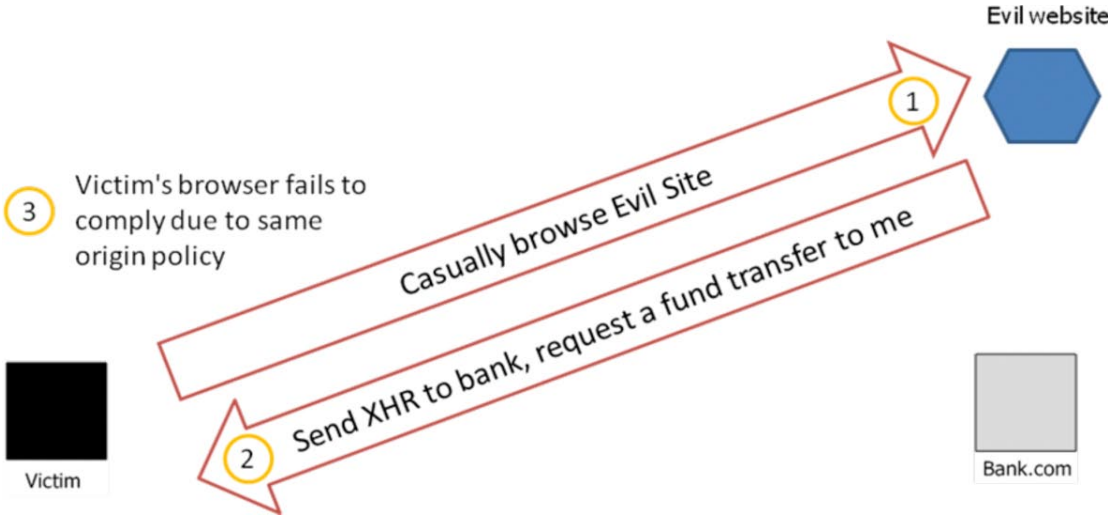


Figure 2. Same Origin Policy stopping the attack

Figure 2 demonstrates the process that an attacker may take to defraud the victim. After the user logs in at Bank.com and acquires the cookie, the attacker has to lure the user to a Web site controlled by the attacker. When the user's browser loads the attacker's page, Javascript created by the attacker executes and instructs the victim's browser to issue an XHR request to Bank.com. The request may for example prompt a money transfer to the attacker. Luckily, the Javascript engine in the victim's browser follows the same origin policy and disallows the XHR to another site, effectively stopping the attack. Had the XHR been issued, Bank.com would process the fund transfer not knowing that the request was issued without the victim's consent, and money would be stolen from the victim.

The same origin policy can be bypassed using DNS rebinding attacks and proxies. (Jackson, Barth, Bortz, Shao, & Boneh). In a DNS rebinding attack, the script is first loaded from an attacking host, using the attacking host's hostname. Next, the DNS resolution for the attacking host's hostname is changed to point to the trusted server's IP address. A request to the untrusted hostname will now be directed to the trusted host. This attack, sometimes also referred to as DNS pinning, is avoided in modern browsers by ignoring the TTL associated with the DNS record. This is not compliant with Internet standards but a choice justified by the security implications (Matthies, 2007). Proxies on the other hand will forward requests from the untrusted to the trusted host. The proxy can also be implemented on the client, and requests are then directed at localhost to be distributed to the respective targets (Stuttard, 2007).

XSRF attacks may still be possible using other languages like Flash, Actionscript, and Java. But that topic is beyond the scope of this paper.

Evolution of Security Features

XHR was first formally defined by the W3C in April of 2006 (W3C Web APIs Working Group, 2006). Since then, the standard has been refined multiple times, most recently in October 2007 (Web API Working Group, 2007). Many of the refinements limit the functionality of XHR for security reasons. Currently, the XHR specification is considered a draft and future updates are possible. The application programming interface (API) defined by XHR allows the browsers scripting engine to issue HTTP or HTTPS requests without direct user interaction, like clicking on a hyper link. The server may respond with content other than XML. Many times, an HTML snippet or an image is returned instead of XML data. Browsers may extend the functionality of XMLHttpRequest beyond HTTP and HTTPS by adding support for other protocols like FTP.

SANS

Working Papers in Application Security

Cross Site Request Forgery: What Attackers Don't Want You to Know

A Study of Browser Implementations and Security Mechanisms for XMLHttpRequest and XDomainRequest

Written by
Jason Lam and
Johannes B. Ullrich, PhD

Table 1 summarizes how various versions of the XHR specification restricted the functionality of the command. Each version removed either certain headers or methods. As an example, the "Host" header could be used in conjunction with DNS pinning to bypass the same origin policy. The CONNECT, TRACE and TRACK methods are usually considered dangerous. They can be used to use Web servers as relays, or for cross site tracing attacks.

Standard Version	Methods Restricted	Headers Restricted
Sept 30, 2008 (this is XMLRequest Level2)	Same as prior version	Access-Control-Request-Headers, Access-Control-Request-Method, Authorization, Connection, Cookie, Cookie2, Host, Origin, User-Agent
April 15, 2008 (this version includes a security section)	CONNECT TRACE TRACK	Same as prior version.
October 26, 2007	Same as prior version	Same as prior version
June 18, 2007	Same as prior version	Same as prior versions and * Connection * Content-Transfer-Encoding * Expect * Via
February 27, 2007	Same as prior version	Same headers restricted as in prior version. However, instead of ignoring them, an SECURITY_ERR is raised.
September 27, 2006	None listed, but a general remark that the function should return a SECURITY_ERR if method is not implemented for security reasons	Following headers are ignored: Accept-Charset, Accept-Encoding, Content-Length, Expect, Date, Host, Keep-Alive, Referer, TE, Trailer, Transfer-Encoding or Upgrade
June 19, 2006	This version has a very generic security section. It mentions the "Host" header as one that should be restricted to enforce same-origin. An editor note states that restrictions on methods need to be addressed.	
April 5, 2006	No mention of restricted methods and headers. However, it is pointed out that the user agent needs to set the Host header appropriately.	

Table 1: Comparison of security features in different versions of the W3C XHR draft.

SANS

Working Papers in Application Security

Cross Site Request Forgery: What Attackers Don't Want You to Know

A Study of Browser Implementations and Security Mechanisms for XMLHttpRequest and XDomainRequest

Written by
Jason Lam and
Johannes B. Ullrich, PhD

XHR Level 2

In 2008, XHR was enhanced by the creation of a Level 2 standard (World Wide Web Consortion (W3C), 2008). Level 2 extends the April 15, 2008 version of XHR. It adds the ability to use cross domain requests and defines restrictions to do so securely.

XHR is now no longer bound by the same origin policy; this is essential to the mash-up of content in the Web 2.0 world. In the past, the developers had to resort to various tricks to get around this limitation. With the new level 2 XHR, developers can directly use XHR to collect data from various sources of different origin while maintaining a reasonably high-security standard and limiting the possibilities for XSRF.

The W3C's access control standard is the basis of the security mechanism within XHR Level 2. A resource can have an access control header included to let a browser know whether cross domain access should be allowed. A browser must also make an appropriate access control decision based on information provided by the resource. Through a set of controls defined by the W3C standard, the cross domain request is tightly controlled.

Browser Testing

Over time, browsers evolved along with the changing XHR draft. In order to validate which features are supported and which are not, a test page (Jason Lam, 2009) was set up to validate which browser versions are complying with which version of the standard. This test page attempts to set various headers and verifies whether or not an error has occurred. The test page will carefully examine errors and check if the header was actually sent to the Web server. The results are displayed to the user immediately after accessing the site.

In order to simplify the analysis of the results and achieve better reliability, the synchronous version of XHR was used. If successful, the response from the server will return the headers sent to the server. For XHR Level 2, the asynchronous version had to be used. The synchronous option is not available for XHR Level 2.

Only current browsers have been tested: Internet Explorer 8, Firefox 3 and Firefox 3.1 Beta. Firefox 3.1 Beta was included because its release is imminent and it does make some substantial changes to the workings of XHR. By the time this paper is released, Firefox 3.1 may be renamed to Firefox 3.5.

SANS
Working Papers in Application Security

Cross Site Request Forgery: What Attackers Don't Want You to Know

A Study of Browser Implementations and Security Mechanisms for XMLHttpRequest and XDomainRequest

Written by
Jason Lam and
Johannes B. Ullrich, PhD

Table 2 summarizes the results:

	IE 7	IE 8	Opera	Safari	Firefox
XHR Support	Yes	Yes	Yes	Yes	Yes
Set Custom Header	Yes	Yes	Yes	Yes	Yes
Set cookie	Yes	Yes	Yes	Yes	Yes
Set Accept-Charset	Yes	Yes	No	No	No
Set Accept-Encoding	No	No	No	No	No
Set Content-Length	No	No	No	No	No
Set Expect	No	No	No	No	No
Set Date	Yes	Yes	No	No	No
Set Host	No	No	No	No	No
Set Keep-Alive	No	No	No	No	No
Set Referer	No	No	No	No	No
Set TE	Yes	Yes	No	No	No
Set Transfer-Encoding	No	No	No	No	No
Get Upgrade	Yes	Yes	No	No	No
Get Connection	Yes	Yes	No	No	No
Get Content-Transfer-Encoding	Yes	Yes	No	No	No
Get Via	Yes	Yes	No	No	No
Opera version: 9.64, Firefox version: 3.0.8, Safari version: 4 build 528.16					

Table 2: Summary of browser XHR capability (Yes=supported)

3. XDomainRequest

Initially proposed by Microsoft, and so far only implemented in Internet Explorer 8, XDomainRequest (XDR) is removing the same-origin policy from XHR. In return, to improve security, XDR is more limited in what headers it can set, and how the origin information is applied to access control. Currently, there is no open standard defining XDR.

At first sight, XDR looks very similar to XHR and its current specification mimics XHR Level 2. Like XHR Level 2, it can only be used asynchronously. The open method can only be used with the GET and POST method and only HTTP and HTTPS URLs are permitted. No method is defined to set headers. Similar to XHR, a callback function is defined to process the response.

Each request created by XDR includes an "Origin" that may not be changed via JavaScript. The value of the origin header is the origin of the script. For example, for a script loaded from http://example.com/test.js, the origin header value would be http://example.com. Any response to a request has to include an Access-Control-Allow-Origin header. If this header is not included at all, JavaScript will exit with an error. The value for the return header may be set to a wildcard "*" if all origins are allowed.

Cross Site Request Forgery: What Attackers Don't Want You to Know

A Study of Browser Implementations and Security Mechanisms for XMLHttpRequest and XDomainRequest

Written by
Jason Lam and
Johannes B. Ullrich, PhD

```
xdr = new XDomainRequest();
xdr.onload=function()
{
    alert(xdr.responseText);
}
xdr.open("GET", "http://www.anotheresite.com/myfile.txt");
xdr.send();
```

Listing 2: XDomainRequest Example code.

XDR follows the W3C cross origin resource sharing standard, which is the same standard followed by XHR Level 2. The standard specifies the headers used to communicate the access control policy on resources. Having this common standard ensures that the resource on the server only need to serve one type of access control headers.

4. W3C Cross-Origin Resource Sharing

There are two modes of operation for cross domain requests. For cross domain requests that are relatively straightforward and safe, the browser directly sends the request to the server, and the data together with the access policy is returned by the server.

In making cross domain requests, some unsafe methods of the HTTP requests can cause the server status to be changed. To avoid any unauthorized access via cross domain request, a preflight request strategy is implemented. Instead of sending the actual request to the server, an OPTION method is sent to the server to ask for the resource's access policy. The preflight request is made to ensure the server is in agreement with the request. The result from the preflight is stored and cached in browser cache. This is done prior to the actual cross domain request.

Below is a comparison of simple and preflighted requests.

	Simple	Preflighted
Criteria	GET or POST No custom header	Other than GET or POST Custom header
Process	Request sent out directly	Pre-flight request sent to server to get resource's policy using OPTION method Result cached Send actual request out
Risks	Generally lower	Much higher risk due to extra header or unsafe HTTP methods

SANS

Working Papers in Application Security

Cross Site Request Forgery: What Attackers Don't Want You to Know

A Study of Browser Implementations and Security Mechanisms for XMLHttpRequest and XDomainRequest

Written by
Jason Lam and
Johannes B. Ullrich, PhD

The W3C access control standard introduces three new request headers. Details are documented in the following table.

Request Header	Function
Origin	Let server know the origin of the request Origin does not contain the sensitive path information (eg. URL querystring
Access-Control-Request-Method	Used in preflight request. Let the server know the actual access method in the intended request (eg. PUT, DELETE)
Access-Control-Request-Headers	Used in preflight request. Let the server know the headers that are used by intended request

W3C standard also specifies five new header responses. Details are documented below.

Request Header	Function
Access-Control-Allow-Origin	Server specify to the client the origin of which the resource can be shared with. For example, value of sans.org in this header will allow the client script to share the resource if the script is running from sans.org
Access-Control-Max-Age	Specify the length of time (in seconds) that the preflight result can be cached.
Access-Control-Allow-Credentials	XMLHTTPRequest can be made to send request to cross domain site with cookie (or HTTP authentication header) by adjusting the "withCredentials" boolean state. Allow-Credentials response is a response to the "withCredentials" request. If the request was sent with credentials and the response does not contain a true value in the Allow-Credentials response, the browser must reject the response and not let the script have access to the response.
Access-Control-Allow-Methods	Only exist in response to preflight request. Specify the methods which can be requested.
Access-Control-Allow-Headers	Only exist in response to preflight request. Specify the headers which can be requested.

SANS

Working Papers in Application Security

Cross Site Request Forgery: What Attackers Don't Want You to Know

A Study of Browser Implementations and Security Mechanisms for XMLHttpRequest and XDomainRequest

Written by
Jason Lam and
Johannes B. Ullrich, PhD

5. Conclusion

The inception of XHR created the opportunity for AJAX technologies. But it benefited various attackers by allowing them to create a new generation of client side attacks. Through the updates to the W3C drafts, the standards of XHR have become more limited over time, eventually making the life of attackers more difficult and stopping some of the attack techniques in their tracks.

The new and upcoming cross domain request ability in Level 2 XHR and in XDR generates very interesting opportunities for both AJAX technologies and hacking communities. This new generation of technologies has security built into them from the start, and the access control component is built by industry consensus. There are no doubts that the security research community and hackers will leverage this newly gained cross domain function in their future arsenal, but they will first have to get past the various controls put in place by the W3C standard. Web developers need to understand these technologies to protect their applications from the ill side effects.

SANS

Working Papers in Application Security

Cross Site Request Forgery: What Attackers Don't Want You to Know

A Study of Browser Implementations and Security Mechanisms for XMLHttpRequest and XDomainRequest

Written by
Jason Lam and
Johannes B. Ullrich, PhD

6. Works Cited

Bhola, S., & Steiner, M. (n.d.). CSRF Protection. Retrieved from OpenAjax alliance: http://www.openajax.org/member/wiki/CSRF_Protection

Finch, P. (2008, 06 02). Using Javascript to POST data between pages. Retrieved from Mental Jetsam: http://mentaljetsam.wordpress.com/2008/06/02/using-javascript-to-post-data-between-pages

Google Inc. (n.d.). Google Maps API Concepts. Retrieved 4-10, 2009, from Google Code: http://code.google.com/apis/maps/documentation

Jackson, C., Barth, A., Bortz, A., Shao, W., & Boneh, D. (n.d.). Protecting Browsers from DNS Rebinding Attacks. Retrieved from Protecting Browsers from DNS Rebinding Attacks: http://crypto.stanford.edu/dns/dns-rebinding.pdf

Jason Lam, J. B. (2009, 4 5). XMLHttpRequest Test Page. Retrieved 4 5, 2009, from SANS Internet Storm Center: http://isc.sans.org/xhrtest.html

Kassner, M. (2008, 12 8). CSRF attacks: Home DSL routers are vulnerable. Retrieved from TechRepublic: http://blogs.techrepublic.com.com/networking/?p=756

Klein, A. (2005, 9). Exploiting the XmlHttpRequest Object in IE - referrer spoofing, and a lot more... Retrieved 4 10, 2009, from cgisecurity: http://www.cgisecurity.com/lib/XmlHTTPRequest.shtml

Matthies, C. (2007, 07 01). DNS Pinning Explained. Retrieved from christian's weblog: http://christ1an.blogspot.com/2007/07/dns-pinning-explained.html

Microsoft Corp. (n.d.). XDomainRequestObject. Retrieved 4 5, 2009, from Microsoft Developer Network: http://msdn.microsoft.com/en-us/library/cc288060(VS.85).aspx

Oreilly, T. (2007). What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software. Communications & Strategies (1), 17.

Stamos, A., & Lackey, Z. (2006). Attacking AJAX Web Applications. Blackhat Japan. iSecPartners.

Stuttard, D. (2007). DNS Pinning and Web Proxies. Retrieved from NGSSoftware: http://www.ngssoftware.com/research/papers/DnsPinningAndWebProxies.pdf

W3C Web APIs Working Group. (2006, April 5). The XMLHttpRequest Object (05 April 2006). (A. van Kesteren, & D. Jackson, Editors) Retrieved April 1, 2009, from World Wide Web Consortium: http://www.w3.org/TR/2006/WD-XMLHttpRequest-20060405

W3C Web Applications Working Group. (2008, 9-30). XMLHttpRequest Level 2. (A. van Kesteren, Ed.) Retrieved 4-10, 2009, from World Wide Web Consortium (W3C): http://www.w3.org/TR/XMLHttpRequest2

Watkins, P. (2001, 06 13). Cross-Site Request Forgeries. Retrieved from http://www.bright-shadows.net/tutorials/csrf.txt

Web API Working Group. (2007, October 26). The XMLHTTP Request Object (26 October 2007). (A. van Kasteren, Editor) Retrieved April 1, 2009, from World Wide Web Consortium: http://www.w3.org/TR/2007/WD-XMLHttpRequest-20071026

World Wide Web Consortium (W3C). (2008, 9 30). XMLHttpRequest Level 2. (A. v. Kesteren, Editor) Retrieved 4-5, 2009, from World Wide Web Consortium: http://www.w3.org/TR/XMLHttpRequest2

About the Authors

Jason Lam is a senior security analyst at a major financial institute in Canada. He is also an author and instructor for SANS Institute where he writes courses on pentesting and defending Web applications. In his ever diminishing free time, he helps with the SANS Internet Storm Center as an incident handler.

Johannes Ullrich, PhD is chief technology officer of the Internet Storm Center.

Last Updated: December 18th, 2017

Upcoming SANS App Sec Training

SANS Security East 2018	New Orleans, LA	Jan 08, 2018 - Jan 13, 2018	Live Event
SANS Amsterdam January 2018	Amsterdam, Netherlands	Jan 15, 2018 - Jan 20, 2018	Live Event
Community SANS San Jose DEV534	San Jose, CA	Jan 29, 2018 - Jan 30, 2018	Community SANS
Community SANS Indianapolis DEV534	Indianapolis, IN	Feb 05, 2018 - Feb 06, 2018	Community SANS
Cloud Security Summit & Training 2018	San Diego, CA	Feb 19, 2018 - Feb 26, 2018	Live Event
Communtiy SANS Seattle DEV534	Seattle, WA	Feb 26, 2018 - Feb 27, 2018	Community SANS
SANS San Francisco Spring 2018	San Francisco, CA	Mar 12, 2018 - Mar 17, 2018	Live Event
San Francisco Spring 2018 - DEV522: Defending Web Applications Security Essentials	San Francisco, CA	Mar 12, 2018 - Apr 17, 2018	vLive
SANS Northern VA Spring - Tysons 2018	McLean, VA	Mar 17, 2018 - Mar 24, 2018	Live Event
SANS 2018	Orlando, FL	Apr 03, 2018 - Apr 10, 2018	Live Event
RSA Conference 2018	San Francisco, CA	Apr 11, 2018 - Apr 16, 2018	Live Event
SANS Baltimore Spring 2018	Baltimore, MD	Apr 21, 2018 - Apr 28, 2018	Live Event
Community SANS New York DEV522	New York, NY	Apr 23, 2018 - Apr 28, 2018	Community SANS
SANS Riyadh April 2018	Riyadh, Saudi Arabia	Apr 28, 2018 - May 03, 2018	Live Event
SANS Security West 2018	San Diego, CA	May 11, 2018 - May 18, 2018	Live Event
SANS Northern VA Reston Spring 2018	Reston, VA	May 20, 2018 - May 25, 2018	Live Event
SANSFIRE 2018	Washington, DC	Jul 14, 2018 - Jul 21, 2018	Live Event
SANS OnDemand	Online	Anytime	Self Paced
SANS SelfStudy	Books & MP3s Only	Anytime	Self Paced