AJAX Cheat Sheet

by binca via cheatography.com/44948/cs/13445/

Asynchronous JavaScript and XML

AJAX enables Web 2.0 using JavaScript and XML to provide dynamic application functions.

Results in "thick" client-like functionality.

JavaScript creates XMLHttpRequest objects that can request and receive asynchronously, updating the display as responses are received.

XMLHttpRequest objects are the heart of AJAX, enabling JavaScript to make requests in the background.

Begin using XMLHttpRequest object with **xmlhttp = new XMLHttpRequest()**

XMLHttpRequest Methods and Properties

open()	Specifies the properties of the request, but it does NOT initiate a connection.
send()	Creates the connection; specifies the function is called when the ready state changes.
readyState	Set with the state of the request
onreadystatechange	Sets which function should be called when ready state changes.
responseText	The response from the server is placed here.

readyState

0	The request is uninitiliazed
1	The request has been set up
2	The request has been sent
3	Waiting for a response
4	The response is complete

It provides information about the state of the server's response to a request.

Mashups

Popular feature of AJAX enabled sites is to combine 2 or more apps to provide more features.

Proxies are often used to circumvent same-origin policy because AJAX does not change it. Many apps use a built-in proxy.

Main issue is control of URLs to proxy. Proxies use GET or POST parameters to call backend site. If these are changed an attacker can proxy to a different site or instruct it to load malicious JavaScript. **Check strings** are often used to prevent this.

Attack Surface

AJAX does not add new attacks but it does increase the attack surface because there is more client-side code and business logic on the client.

Typical attacks are XSS, SQLi, and it lends itself toward CSRF because functionality is called directly by the client code.

AJAX Mapping

It is more difficult because many tools cannot parse and handle the client-side logic, especially dynamically generated links can cause issue

Manual work and tool verification is often necessary.

Burp and **ZAP**, with **AJAX Spider**, can often ably handle AJAX applications.

Exploitation

It is not more difficult but many tools require manual priming.

JavaScript Libraries/Frameworks

AJAX lends itself to complex frameworks; many common files are used within apps and functions are often included on all pages forsimplicity.

Non-AJAX applications can also make use of API files.

Most commonly JavaScript files are included across the application containing business logic and technical functionality. When features are present that we do not have access to it can help build malicious requests.

3rd party libraries/frameworks whether CDN-hosted or served locally can introduced their own vulnerabilities. Examples includes jQuery and MooTools.

Discovery often occurs during **mapping**, a spider should detect them (src attributes).

They can be parsed to find vulnerable or interesting functions, look for those that initiate or process **HTTP requests (XMLHttpRequest)**.

Exploitation takes many forms and are based on the framework. Possibilities include the ability to call functions without authentication, gather information for further exploitation, or use a known-flaw to exploit.

Data Attacks

Business logic residing on the client means the client receives more data, in some cases more than is required.

Developers rely on filtering on the client-side to display only what is necessary.

Data Formats: XML and JSON

There are 2 common formats that require parsing on client side: **XML** and **JSON**

XML is a tag-based format, that while common is heavier than other forms.

JSON, or JavaScript Object Notation, is a lightweight interchangeable format for both requests and responses, which client-side JavaScript loads into memory.

Typically, **JSON** uses eval() to invoke JavaScript compiler to parse text and project object structure. It is very fast but runs the risk of executing any JavaScript program present. It is safer to use a JSON parser.

On the **response** side an attacker will look for extraneous data that can be used.

On the **request** side an attacker will look for the opportunity to inject attacks, such as SQLi or XSS.

JSON format is an array of arrays.

Exploitation of JSON is usually either information disclosure or injection.

JSON Information Disclosure is the easiest to find and is typically found browsing via proxy. Some applications send complete record sets.

JSON Injection focuses on request with the goal of intercepting them and injecting attack strings, SQLi and XSS being the most common. The client or server code can be targeted. The results may not be visible on the page, so using an interception proxy is necessary.

Note: JSON is unforgiving of syntax errors but the errors are often verbose, which can provide guidance.