



Development Issues within AJAX Applications: How to Divert Threats

Lars Ewe
CTO
Cenzic
lars@cenzic.com

OWASP

July, 2009

Copyright © The OWASP Foundation
Permission is granted to copy, distribute and/or modify this document
under the terms of the OWASP License.

The OWASP Foundation
<http://www.owasp.org>

Agenda

- What is AJAX?
- AJAX and Web App Security
- AJAX and Test Automation
- Vulnerability Examples:
XSS, CSRF & JavaScript Hijacking
- AJAX Best Security Practices
- Demo
- Q & A



What is AJAX?

- Aynchronous JavaScript And XML
- AJAX allows for a new generation of more dynamic, more interactive, faster Web 2.0 applications
- AJAX leverages existing technologies, such as Dynamic HTML (DHTML), Cascading Style Sheets (CSS), Document Object Model (DOM), JavaScript Object Notation (JSON), etc., and the (a)synchronous XMLHttpRequest (XHR)
- Not just a set of technologies, but a new Web application development approach and methodology

What is AJAX? (contd.)

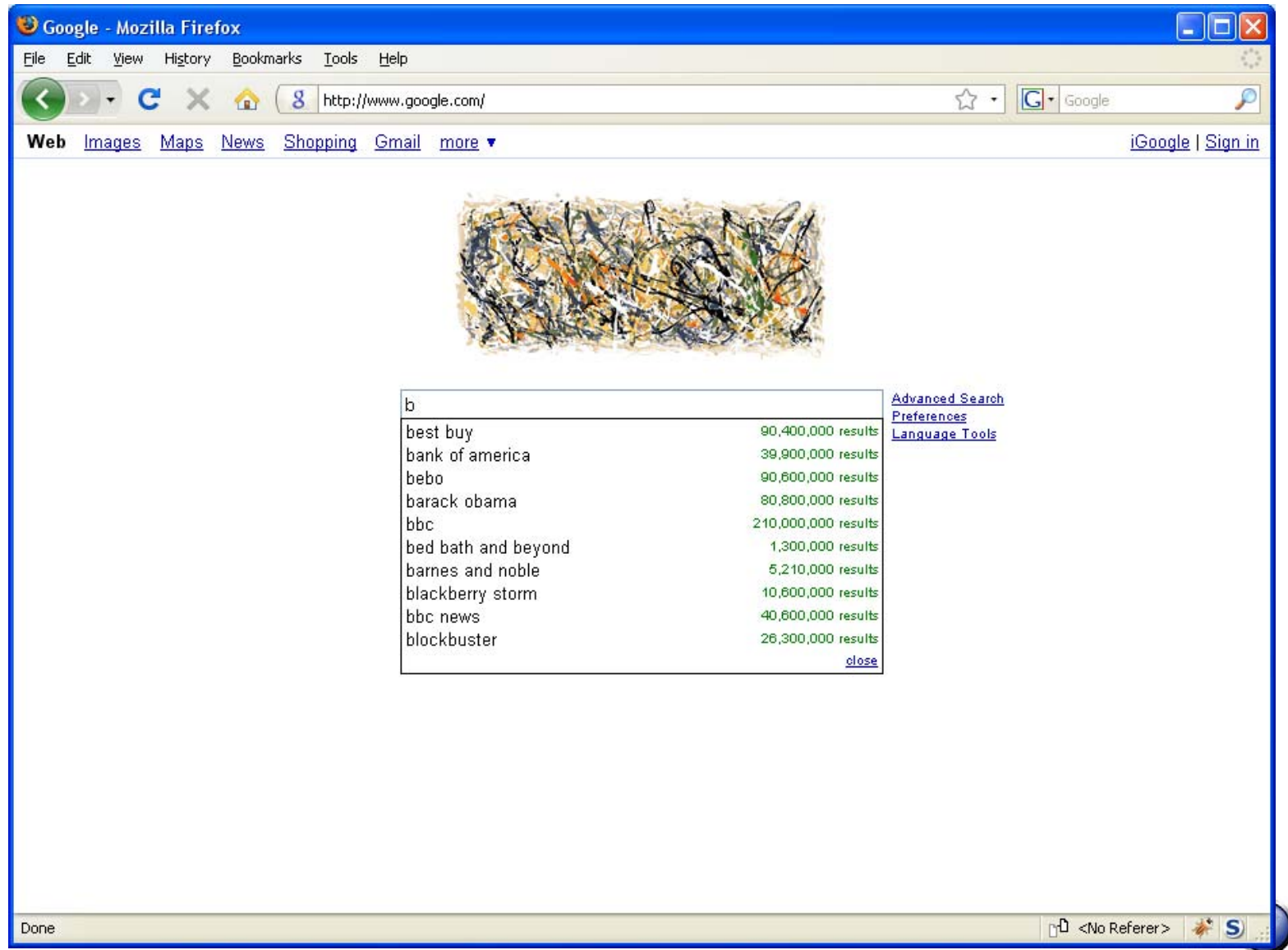
- XHR allows for (a)synchronous server requests without the need for a full page reload
- XHR “downstream” payload can be
 - XML, JSON, HTML/JS snippets, plain text, serialized data, basically pretty much anything...
- Responses often get further processed using JavaScript and result in dynamic web page content changes through DOM modifications

AJAX Code Example

```
xhr = new XMLHttpRequest();  
xhr.open("GET", AJAX_call?foo=bar, true);  
xhr.onreadystatechange = processResponse;  
xhr.send(null);
```

```
function processResponse () {  
    if (xhr.readyState == 4) {  
        if (request.status == 200) {  
            response =  
                xhr.responseText;  
            .....  
        }  
    }  
}
```

AJAX Example #1

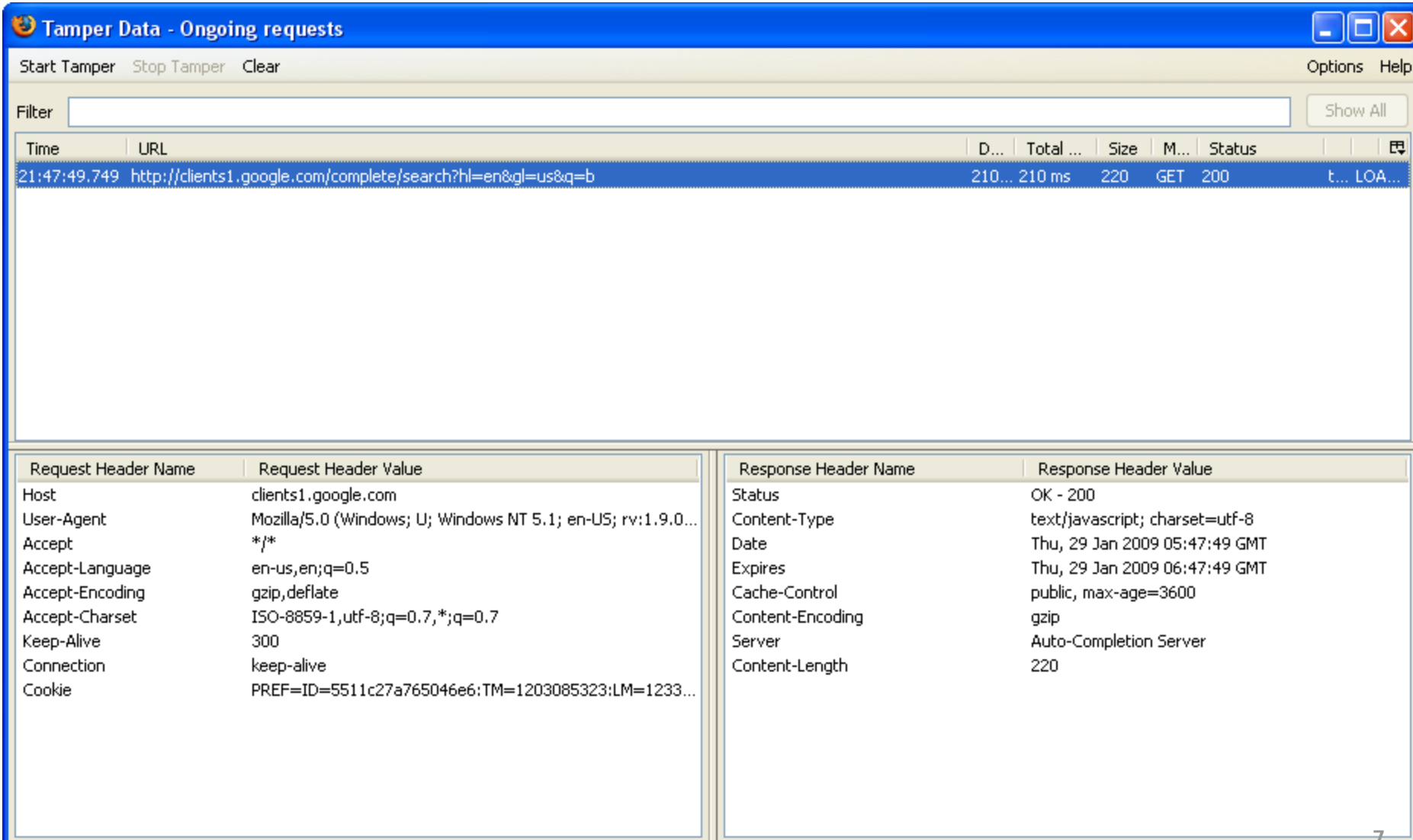


The screenshot shows a Mozilla Firefox browser window with the Google homepage. The address bar shows the URL `http://www.google.com/`. The search bar contains the letter 'b', and a dropdown menu is open, displaying a list of search suggestions. The suggestions are as follows:

Suggestion	Results
best buy	90,400,000 results
bank of america	39,900,000 results
bebo	90,600,000 results
barack obama	80,800,000 results
bbc	210,000,000 results
bed bath and beyond	1,300,000 results
barnes and noble	5,210,000 results
blackberry storm	10,600,000 results
bbc news	40,600,000 results
blockbuster	26,300,000 results

Navigation links at the top include [Web](#), [Images](#), [Maps](#), [News](#), [Shopping](#), [Gmail](#), and [more](#). On the right side, there are links for [iGoogle](#) and [Sign in](#). The status bar at the bottom shows 'Done' and '<No Referer>'.

AJAX Example #1



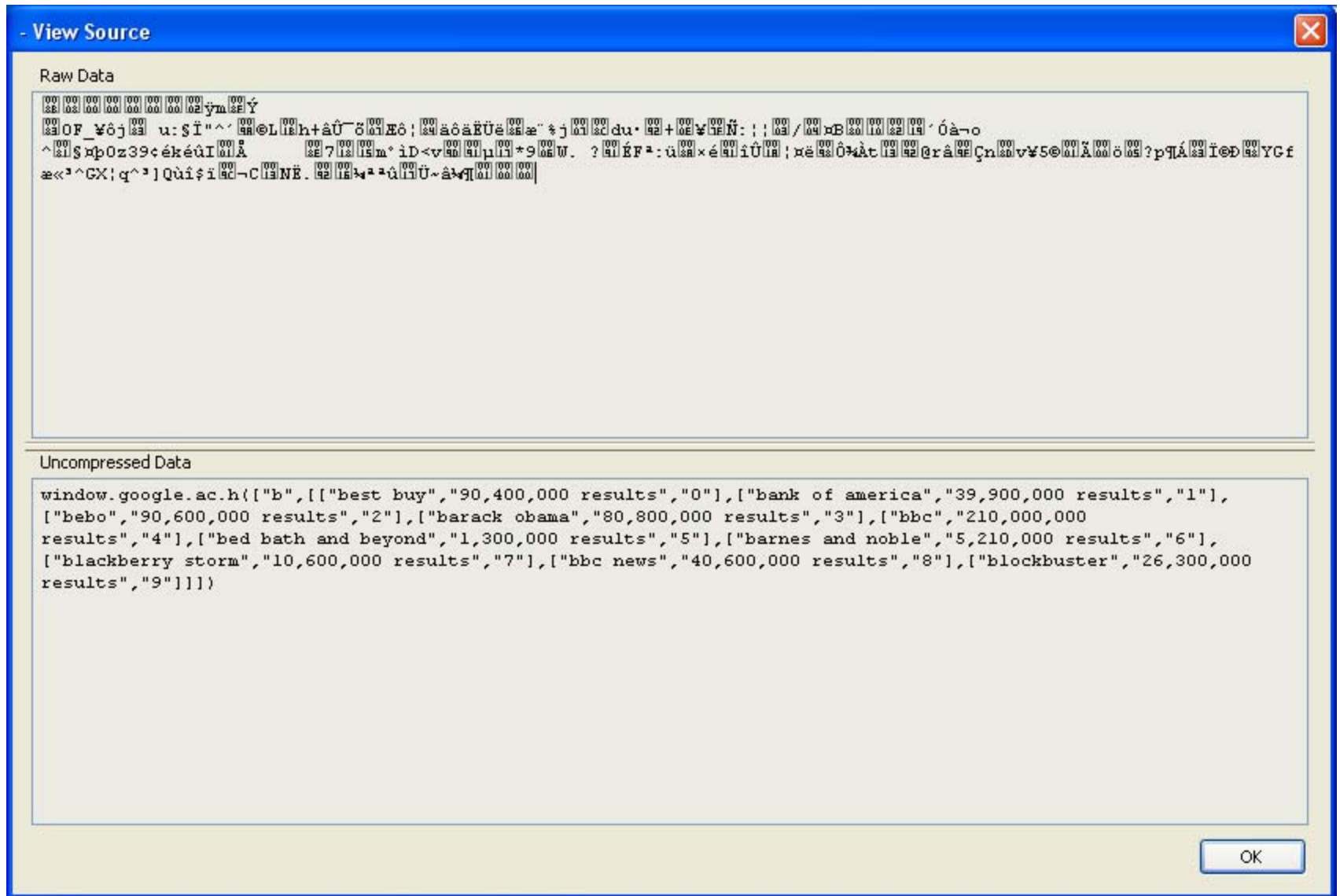
The screenshot shows the 'Tamper Data - Ongoing requests' window. At the top, there are buttons for 'Start Tamper', 'Stop Tamper', and 'Clear', along with 'Options' and 'Help'. A 'Filter' input field is present. Below this is a table of ongoing requests with columns for Time, URL, D..., Total..., Size, M..., and Status. One request is listed at 21:47:49.749 to http://clients1.google.com/complete/search?hl=en&gl=us&q=b. The bottom section is divided into two tables: 'Request Header Name' and 'Request Header Value' on the left, and 'Response Header Name' and 'Response Header Value' on the right.

Time	URL	D...	Total ...	Size	M...	Status
21:47:49.749	http://clients1.google.com/complete/search?hl=en&gl=us&q=b	210...	210 ms	220	GET	200

Request Header Name	Request Header Value
Host	clients1.google.com
User-Agent	Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.0...
Accept	*/*
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
Keep-Alive	300
Connection	keep-alive
Cookie	PREF=ID=5511c27a765046e6:TM=1203085323:LM=1233...

Response Header Name	Response Header Value
Status	OK - 200
Content-Type	text/javascript; charset=utf-8
Date	Thu, 29 Jan 2009 05:47:49 GMT
Expires	Thu, 29 Jan 2009 06:47:49 GMT
Cache-Control	public, max-age=3600
Content-Encoding	gzip
Server	Auto-Completion Server
Content-Length	220

AJAX Example #1



The screenshot shows a browser window titled "- View Source" with a close button in the top right corner. The window is divided into two sections: "Raw Data" and "Uncompressed Data".

Raw Data

```
OF_Ψój u: $İ"^^ @Ll|h+âŮ-Œó; äóäRÜëæ" *j du• +ΨN: ; /xB l' Óà-o  
^ $xp0z39< ékéúI 7m* iD<v qm+9W. ? ÉF²: ú×é iŮ; æé Ó%àt @ràÇn v$5@ Ä ö? pTÁ I@ YGf  
æ<³ ^CX; q^³ J Qùí $ i c-C INÈ. %² = ú Ů ~ â V
```

Uncompressed Data

```
window.google.ac.h(["b", [{"best buy", "90,400,000 results", "0"}, {"bank of america", "39,900,000 results", "1"}, {"bebo", "90,600,000 results", "2"}, {"barack obama", "80,800,000 results", "3"}, {"bbc", "210,000,000 results", "4"}, {"bed bath and beyond", "1,300,000 results", "5"}, {"barnes and noble", "5,210,000 results", "6"}, {"blackberry storm", "10,600,000 results", "7"}, {"bbc news", "40,600,000 results", "8"}, {"blockbuster", "26,300,000 results", "9"}]])
```

An "OK" button is located at the bottom right of the window.

AJAX Example #2

The screenshot shows a Mozilla Firefox browser window with the address bar displaying `http://demo.script.aculo.us/ajax/autocompleter_customized`. The page features the *script.aculo.us* logo and navigation links for "back to site" and "demos". A central orange box contains the text: "This site demonstrates the **Ruby on Rails** AJAX helpers, which use script.aculo.us effects, drag-and-drop and autocomplete."

The main content area is titled "ajax auto completion demo" and shows a "To:" field with the letter "a" entered. A dropdown menu lists several email addresses, with "Ada Noel" selected. The list includes:

- Ada Noel (ada@noel.fake)
- Adlai Cathy (adlai@cathy.fake)
- Adrian Audrey (adrian@audrey.fake)
- Adrian Clyde (adrian@clyde.fake)
- Adrian Ramneek (adrian@ramneek.fake)
- Adrienne Amos (adrienne@amos.fake)
- Adrienne Conrad (adrienne@conrad.fake)
- Agatha Lesley (agatha@lesley.fake)

Below the dropdown is a "View source" link. To the right, a "rails demos" section lists various features: "script.aculo.us helpers", "Autocompleting text fields (basic)", "Autocompleting text fields (customized)", "Shopping cart", "Sortable elements", "New AJAX features", "Error handling", and "Update element helper".

At the bottom right, there is a "RAILS" logo and a paragraph of text: "Rails is a full-stack, open-source web framework in Ruby for writing real-world applications with joy and less code than most frameworks spend doing XML sit-ups. That quite sums it up. So, if you're still working late hours writing definitions of what's in your database because the framework you use works against and not for you -- do yourself a favour, and check out [Ruby on Rails](#)."

The footer of the browser window shows copyright information: "© 2005-2007 thomas fuchs license mir.aculo.us". The status bar at the bottom indicates "Done" and "<No Referer>".

AJAX Example #2

The screenshot shows the 'Tamper Data - Ongoing requests' window. At the top, there are buttons for 'Start Tamper', 'Stop Tamper', and 'Clear', along with 'Options' and 'Help'. A 'Filter' input field is present. Below is a table of ongoing requests:

Time	URL	M...	D...	Total D...	Size	...	Load Flags
13:09:21.218	http://demo.script.aculo.us/ajax/auto_complete_for_message_to	POST	326...	326 ms	-1	2...	t... LOAD_BYPASS_C...
13:09:21.547	http://demo.script.aculo.us/demos/images/contacts/5.jpg	GET	139...	139 ms	-1	4...	t... LOAD_FROM_CA...
13:09:21.548	http://demo.script.aculo.us/demos/images/contacts/8.jpg	GET	205...	205 ms	-1	4...	t... LOAD_FROM_CA...
13:09:21.549	http://demo.script.aculo.us/demos/images/contacts/3.jpg	GET	210...	210 ms	-1	4...	t... LOAD_FROM_CA...
13:09:21.550	http://demo.script.aculo.us/demos/images/contacts/1.jpg	GET	213...	213 ms	-1	4...	t... LOAD_FROM_CA...

Below the request list are two panels for request and response headers:

Request Header Name	Request Header Value	Response Header Name	Response Header Value
Host	demo.script.aculo.us	Status	OK - 200
User-Agent	Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1...	Server	nginx/0.5.33
Accept	text/html,application/xhtml+xml,application/xml;q=0.9...	Date	Thu, 29 Jan 2009 21:09:23 GMT
Accept-Language	en-us,en;q=0.5	Content-Type	text/html; charset=utf-8
Accept-Encoding	gzip,deflate	Transfer-Encoding	chunked
Accept-Charset	ISO-8859-1,utf-8;q=0.7,*;q=0.7	Connection	keep-alive
Keep-Alive	300	Status	200 OK
Connection	keep-alive	X-Runtime	0.21905
X-Requested-With	XMLHttpRequest	Etag	"ad2a0b746d83f962da315e12acaaaadd"
X-Prototype-Version	1.3.0	Cache-Control	private, max-age=0, must-revalidate
Content-Type	application/x-www-form-urlencoded; charset=UTF-8	Content-Encoding	gzip
Content-Length	20		
Cookie	__scriptaculous-demos_session=BAh7BiIKZmxhc2hJQzo...		
Pragma	no-cache		
Cache-Control	no-cache		
POSTDATA	message%5Bto%5D=a&_ =		

AJAX Deployment Statistics

- Cenzic CTS (SaaS): ~30% of recently tested applications use AJAX
- >50% AJAX developer growth year-over-year – Evans Data, 2007
- ~3.5 million AJAX developers worldwide – Evans Data, 2007
- 60% of new application projects will use Rich Internet Application (RIA) technologies such as AJAX within the next three years – Gartner, 2007

AJAX and the Same Origin Policy

- Same origin policy is a key browser security mechanism
 - To prevent any cross-domain data leakage, etc.
 - With JavaScript it doesn't allow JavaScript from domain A to access content / data from domain B
- In the case of XHR, the same origin policy does not allow for any cross-domain XHR requests
 - Developers often don't like this at all!

Common Cross Domain Workarounds

Cross-domain access is often still implemented by various means, such as

- Open / Application (server-based) proxies
- Flash & Java Applets (depending on `crossdomain.xml`)
 - E.g. `FlashXMLHttpRequest` by Julien Couvreur
- RESTful web service with JS callback and JSON response
 - E.g. `JSONscriptRequest` by Jason Levitt

AJAX Frameworks

- AJAX frameworks are often categorized as either “Client” or “Proxy/Server” framework
- “Proxy/Server” frameworks sometimes result in unintended method / functionality exposure
- Beware of any kind of “Debugging mode”
- Remember: Attackers can easily “fingerprint” AJAX frameworks
- Beware of JavaScript Hijacking
 - Don't use HTTP GET for “upstream”
 - Prefix “downstream” JavaScript with `while(1);`

AJAX and Web App Security

- AJAX potentially increases the attack surface
 - More “hidden” calls mean more potential security holes
- AJAX developers sometimes pay less attention to security, due to it’s “hidden” nature
 - Basically the old mistake of security by obscurity
- AJAX developers sometimes tend to rely on client side validation
 - An approach that is just as flawed with or without AJAX

AJAX and Web App Security (contd.)

- Mash-up calls / functionality are often less secure by design
 - 3rd party APIs (e.g. feeds, blogs, search APIs, etc.) are often designed with ease of use, not security in mind
 - Mash-ups often lack clear security boundaries (who validates, who filters, who encodes / decodes, etc.)
 - Mash-ups often result in untrusted cross-domain access workarounds
- AJAX sometimes promotes dynamic code (JavaScript) execution of untrusted response data

The Bottom Line...

AJAX **adds to the problem** of well-known Web application vulnerabilities, such as XSS, CSRF, etc.



AJAX and Test Automation

- Spidering is more complex than just processing ANCHOR HREF's; various events need to be simulated (e.g. mouseover, keydown, keyup, onclick, onfocus, onblur, etc.)
- Timer events and dynamic DOM changes need to be observed
- Use of non-standard data formats for both requests and responses make injection and detection hard to automate
- Page changes after XHR requests can sometimes be delayed
- In short, you need to have browser like behavior (JS engine, DOM & event management, etc.)

Cross-Site Scripting (XSS)

- **What is it?:** The Web Application is used to store, transport, and deliver malicious active content to an unsuspecting user.
- **Root Cause:** Failure to proactively reject or scrub malicious characters from input vectors.
- **Impact:** Persistent XSS is stored and executed at a later time, by a user. Allows cookie theft, credential theft, data confidentiality, integrity, and availability risks. Browser Hijacking and Unauthorized Access to Web Application is possible.
- **Solution:** A global as well as form and field specific policy for handling untrusted content. Use whitelists, blacklists, and regular expressions to ensure input data conforms to the required character set, size, and syntax.

Cross-Site Request Forgery (CSRF)

- **What is it?:** Basic Web application session management behavior is exploited to make legitimate user requests without the user's knowledge or consent.
- **Root Cause:** Basic (cookie-based) session id management that is vulnerable to exploitation.
- **Impact:** Attackers can make legitimate Web requests from the victim's browser without the victim's knowledge or consent, allowing legitimate transactions in the user's name. This can result in a broad variety of possible exploits.
- **Solution:** Enhance session management by using non-predictable "nonce" or other unique one-time tokens in addition to common session identifiers, as well as the validation of HTTP Referrer headers.

JavaScript Hijacking

- **What is it?:** An attack vector specific to JavaScript messages. Confidential data contained in JavaScript messages is being accessed by the attacker despite the browser's same origin policy.
- **Root Cause:** The `<script>` tag circumvents the browser's same origin policy. In some cases the attacker can set up an environment that lets him observe the execution of certain aspects of the JavaScript message. Examples: Override/implement native Object constructors (e.g. Array) or callback function. This can result in access to the data loaded by the `<script>` tag.
- **Impact:** Data confidentiality, integrity, and availability with the ability to access any confidential data transferred by JavaScript.
- **Solution:** Implement CSRF defense mechanisms; prevent the direct execution of the JavaScript message. Wrap your JavaScript with non-executable pre- and suffixes that get stripped off prior to execution of the sanitized JavaScript message. Example: Prefix your JavaScript with `while(1);`

JavaScript Hijacking

Example #1: Override Array Constructor

Attacker code (override Array constructor)

```
<script type="text/javascript">  
function Array(){  
/* Put hack to access Array elements here */  
}  
</script>
```

AJAX Call

```
<script src="http://AJAX_call?foo=bar"  
type="text/javascript"></script>
```

Example AJAX response

```
[ "foo1", "bar1" ], [ "foo2", "bar2" ]
```

JavaScript Hijacking

Example #2: Implement Callback

Attacker code (implement callback)

```
<script type="text/javascript">
function callback(foo){
/* Put hack to access callback data here */
}
</script>
```

AJAX Call

```
<script src="http://AJAX_call?foo=bar"
type="text/javascript"></script>
```

Example AJAX response

```
callback(["foo", "bar"]);
```


Preventing JavaScript Hijacking

A simple code example

```
var object;  
var xhr = new XMLHttpRequest();  
xhr.open("GET", "/object.json", true);  
xhr.onreadystatechange = function () {  
    if (xhr.readyState == 4) {  
        var txt = xhr.responseText;  
        if (txt.substr(0,9) == "while(1);") {  
            txt = txt.substring(10);  
            Object = eval("(" + txt + ")");  
        }  
    }  
};  
xhr.send(null);
```

Remember, the attacker cannot sanitize the JavaScript, since they are relying on the `<script>` tag

AJAX Best Security Practices

Pretty much all the usual Web app security best practices apply:

- Analyze and know your security boundaries and attack surfaces
- Beware of reliance on client-side security measures
 - Always implement strong server side input & parameter validation (black & whitelisting)
 - Test against a robust set of evasion rules
 - Remember: The client can never be trusted!
- Assume the worst case scenario for all 3rd party interactions
 - 3rd parties can inherently not be trusted!

AJAX Best Security Practices (contd.)

- Be extremely careful when circumventing same origin policy
- Avoid / limit the use of dynamic code / `eval ()`
- Beware of JavaScript Hijacking (prefix JavaScript with `while (1) ;`)
- Implement anti-CSRF defenses
- Escape special characters before sending them to the browser (e.g. `<` to `<` ;)
- Leverage HTTPS for sensitive data, use `HTTPOnly` & `Secure` cookie flags
- Use parameterized SQL for any DB queries
- Also see owasp.org and OWASP dev guide

XSS AJAX & JavaScript Hijacking Demo

