



build | integrate | secure

Grow with Denim Group



Web 2.0 and Web Services Security

OWASP Houston

June 5th, 2007

Overview

- Introduction
- Definition of Web 2.0
- Basics of AJAX
- Attack Vectors for AJAX Applications
- AJAX and Application Security
- Conclusions

Introduction

- Dan Cornell
- Principal of Denim Group, Ltd.
- Software developer by training and background
 - *JEE*
 - *.NET*
 - *A little bit of everything else*
- Currently focused on security
 - *Application assessments and penetration tests*
 - *Training and mentoring*
 - *Look at both building and breaking web applications*

Definitions of Web 2.0

- Evangelical
- Technical

Definition of Web 2.0 (Evangelical)

“Web 2.0 is the business revolution in the computer industry caused by the move to the internet as platform, and an attempt to understand the rules for success on that new platform. Chief among those rules is this: Build applications that harness network effects to get better the more people use them.”

-Tim O'Reilly

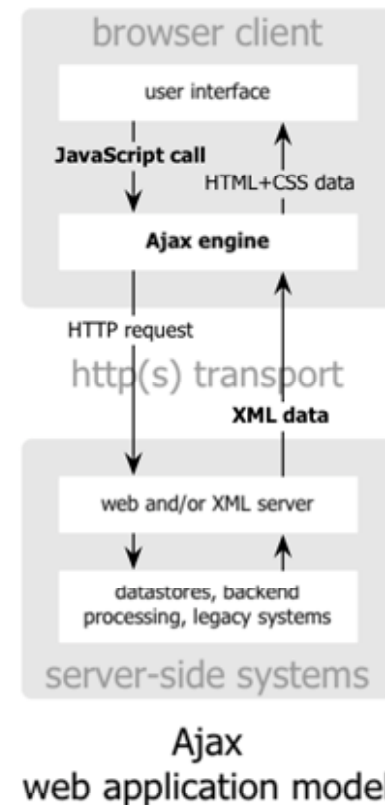
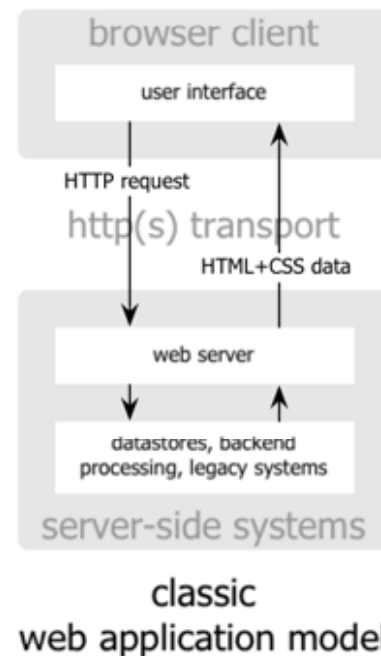
Definition of Web 2.0 (Technical)

- Collaborative Applications
- “New” Technologies
 - *AJAX*
 - *Flash/Flex*
- Rich Internet Applications
 - *UI is more interactive*
- Key to remember: HTTP requests are still just bits transmitted across the wire

Basics of AJAX

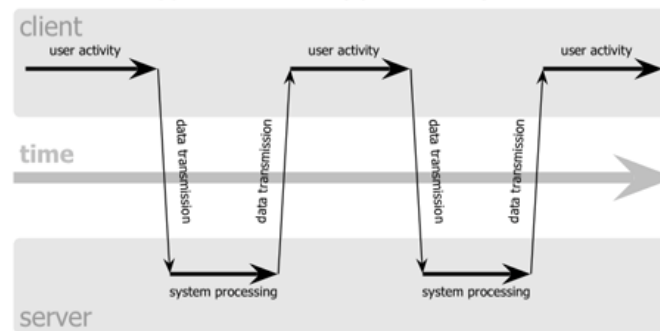
- Asynchronous JavaScript and XML
- Term coined by Jesse James Garrett
 - <http://www.adaptivepath.com/publications/essays/archives/000385.php>
 - *Illustrations from next two slides come from this URL*
- Relies on the XMLHttpRequest object accessible from JavaScript
 - *Security model only allows the browser to connect back to the original server*
- These techniques have been around for a while but have only recently seen wide adoption

AJAX Illustrated

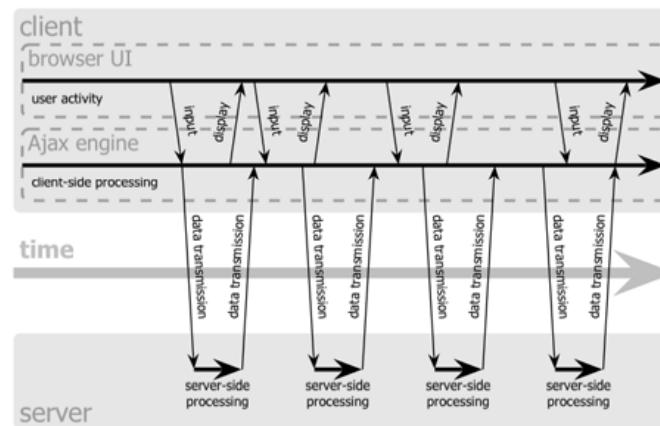


AJAX Illustrated

classic web application model (synchronous)



Ajax web application model (asynchronous)



What Does AJAX Look Like?

- Still bits going across the wire
- Still HTTP traffic
 - *Headers*
 - *GET parameters*
 - *POST data*
- Content is typically different than normal HTTP traffic
 - *XML*
 - *JSON*
 - *(versus HTTP POSTs and HTML)*

What Does AJAX Look Like?

```
POST http://localhost:2104/DenimGroup.Sprajax.Atlas.DemoSite/Services/BookService.aspx?mn=RetrieveBookTitleForId HTTP/1.1
Host: localhost:2104
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.2) Gecko/20070219 Firefox/2.0.0.2
Accept: application/x-shockwave-flash,text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
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
Proxy-Connection: keep-alive
Content-Type: application/json
Referer: http://localhost:2104/DenimGroup.Sprajax.Atlas.DemoSite/BookInformation.aspx
Content-length: 14
Cookie: ASP.NET_SessionId=cssl4pyz3irzi045uwel0n55; ASPSESSIONIDASSDSRTT=OBEPMGDAFMJNEMCOKLPEHCHE
Pragma: no-cache
Cache-Control: no-cache

{"bookId":"3"}
```

What Does AJAX Look Like?

```
HTTP/1.1 200 OK
Server: ASP.NET Development Server/8.0.0.0
Date: Wed, 21 Mar 2007 17:21:53 GMT
X-AspNet-Version: 2.0.50727
Cache-Control: private, max-age=0
Content-Type: application/json; charset=utf-8
Content-length: 15
Connection: Close

"Cryptonomicon"
```

Why is AJAX Security a Concern?

- Web 2.0 application have all the same problems as traditional web applications – and more!
- Threat models / risk profiles are not well understood
 - *Focus on what CAN be done not what SHOULD be done*
- Practitioners are focused on *what* they can do rather than *how* (or *if*) they should do it

Attack Vectors for AJAX Applications

- Front-door attacks on server-side resources
- Remote control attacks via target application
- Remote control attacks via 3rd party application

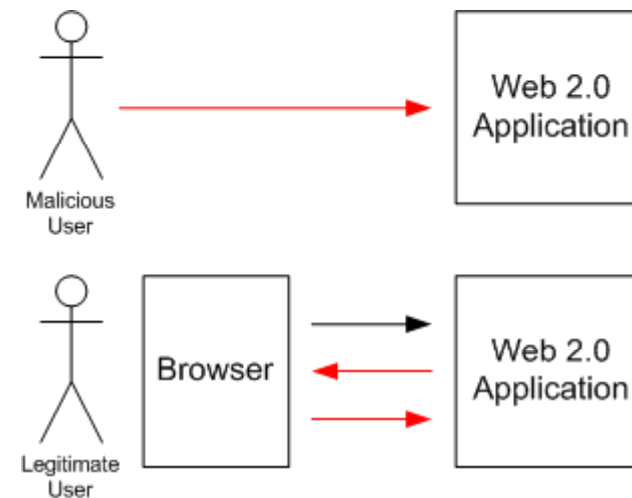
Attack Vectors for AJAX Applications

- Front-door attacks on server-side resources
 - *SQL injection attacks*
 - *AuthX, AuthZ attacks*



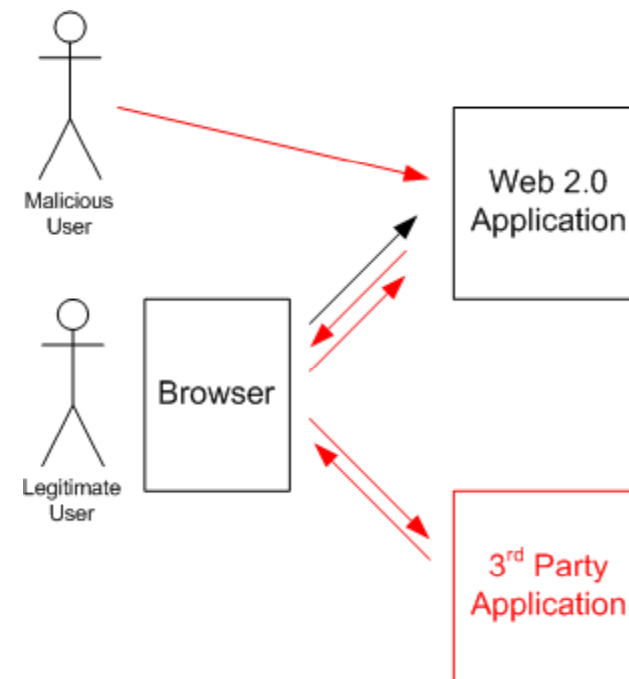
Attack Vectors for AJAX Applications

- Remote control attacks via target application
 - *Cross Site Scripting (XSS)*
 - *Cross Site Request Forgery (CSRF)*



Attack Vectors for AJAX Applications

- Remote control attacks via 3rd party application
 - *JavaScript manipulation*



Fundamentals of AJAX Security

- All the standard rules for web application security still apply
 - *Perform Threat Modeling / Risk Assessment*
 - *Validate input: Do not trust anything in the request*
 - Cookies, other HTTP Headers
 - Parameters
 - Prefer positive validation to negative validation
 - *Escape output*
 - HTML (HTMLEncode)
 - XML (XMLEncode)
 - URL parameters (URLEncode)
 - *Authentication and Authorization still matter*
 - Web services and other Web 2.0 endpoints must require authentication and should enforce authorization

Fundamentals of AJAX Security

- New things to think about
 - *Protect methods to access sensitive URLs*
 - Prefer POST requests over GET requests
 - There are lots of ways to make GET requests (<SCRIPT> and so on)
 - POST requests are typically only automated via XMLHttpRequest
 - Know what hosts serve what content
 - *Be careful what you send via JavaScript*
 - Sensitive data may be accessible off-domain

Some (In)Famous Vulnerabilities

- MySpace Worm
- GMail Hacks

MySpace Worm

- Based on an assembled JavaScript built up from several substrings
 - *Bypassed numerous negative filters against malicious tags, attributes, etc*
- Relied on the same functionality being served from multiple hostnames
 - *profile.myspace.com and www.myspace.com*
- Used a multi-step background combination of GETs and POSTs
 - *Get the list of friends*
 - *Parse out special POST tokens*
 - *Add samy as a new friend*
- For more info: <http://namb.la/popular/>
- Technical explanation: <http://namb.la/popular/tech.html>

MySpace Worm: Lessons Learned

- Better input validation – positive versus negative where possible
 - *Interpretable HTML is a “big place” so carving out a couple of tags or syntax bits will not remove the ability of attackers to find workarounds*
- Be careful about what functionality is available from where
 - *GETs versus POSTs*
 - *What applications are available from a given hostname?*
 - *Similar to checking to be sure that HTTPS content is not available via HTTP*

GMail Hacks

- Link to malicious site sent in GMail message
- User clicks on link and page renders
- Page contains a script reference to a Gmail URL that will render sensitive data encoded in JavaScript
 - *User's cookies are sent along to Gmail because user was just logged in to Gmail*
- Page also contains an override for the JavaScript Array() constructor
- Browser can then access off-domain JavaScript content and send along to a 3rd party site
- More information:
 - <http://jeremiahgrossman.blogspot.com/2006/01/advanced-web-attack-techniques-using.html>
- Easier to type: <http://tinyurl.com/2t99b8>

GMail Hacks: Lessons Learned

- Do not put sensitive data in JavaScript content in a form that can be automatically parsed
 - *What if it ends up in a SCRIPT tag?*
 - *Possible to override constructors in JavaScript language*
- URLs that render JavaScript with sensitive data should be hard to get to (for an attacker)
 - *POST rather than GET*
 - *Unpredictable URL*
- Check referrers
 - *Can be spoofed but can help prevent execution in some cases*

AJAX and Application Security

- Old Favorites
 - *Authentication and Authorization*
 - *SQL Injection*
 - *Cookie Tampering*
 - *Parameter Tampering*
- New Stuff
 - *Additional State on the Client*
 - *Cross Site Scripting*
 - *Cross Site Request Forgery (CSRF)*
 - *JavaScript Manipulation*

Old Favorites

- Authentication and Authorization
- SQL Injection
- Cookie Tampering
- Parameter Tampering

Authentication and Authorization

- XMLHttpRequests send along the same cookies as the browser
 - *Leverage the server-side session*
- Require authentication as with accessing any other web application
- Require authorization before allowing access to sensitive content or capabilities
- Continue to leverage platform Authentication and Authorization features

SQL Injection

- This works just like it does for normal web applications
- Creating SQL queries based on static text and unfiltered inputs

Cookie Tampering

- Cookies are passed along with AJAX XMLHttpRequests
 - *Can piggyback of pre-existing session login*
- Malicious users can still modify cookies when making direct requests

Parameter Tampering

- Parameters are passed along with the XMLHttpRequests via a variety of means
 - *GET*
 - *POST*
 - Each framework has its own encoding scheme

New Stuff

- Additional State on the Client
- Cross Site Scripting (XSS)
- Cross Site Request Forgery (CSRF)
- JavaScript Manipulation
- XML Security
- JSON Security

Additional State on the Client

- Bad in normal web applications
 - *Hidden FORM fields*
 - *Cookies*
- **TERRIBLE** in AJAX applications
 - *Although handling things on the client is kind of the point of AJAX*
 - *No excuse for sloppy server-side validation*

Cross Site Scripting

- Receiving HTML from 3rd party sites is just as dangerous to AJAX-enabled applications
- How much do you trust the folks you are talking to?
 - *Standard threat modeling applies*

Cross Site Request Forgery (CSRF)

- Similar to Cross Site Scripting (XSS)
- Inject HTML or JavaScript to make rogue requests
 - *XMLHttpRequest calls*
 - ``
- Risk mitigation:
 - *Limit URLs to only render for POST requests*
 - Helps prevent access via IMG tags and the like
 - *Add nonces to URLs*
 - *Require ViewState encryption/validation on ASP.NET sites*

JavaScript Manipulation

- AJAX applications often render our JSON or other valid JavaScript
- These can be potentially accessed via 3rd party sites
- Sensitive data can be retrieved and sent elsewhere
- Risk Mitigation
 - *Do not send sensitive data in JavaScript parsable format*
 - Might end up in a <SCRIPT> tag
 - *It is possible to defeat built-in browser protections for JavaScript security*
 - Redefine JavaScript Array() constructor
 - Redefine JavaScript object() constructor

XML Security

- If clients are sending XML documents be careful what you trust
- DOM
 - *Text on network wire is turned into objects in server memory*
 - *Potential for DoS attacks by consuming memory*
 - *<tag /><tag /><tag />...*
- SAX
 - *XML documents passed between systems can be injected just like SQL*
 - *Watch out for re-occurrence of tags*
 - *Application expects <tag1 /><tag2 />*
 - *Application sees <tag1 /><tag2 /><tag1 />*

JSON Security

- JSON can be made into JavaScript objects using the eval() command
- This is dangerous
- Use parseJSON() function instead

OWASP and Web 2.0 Security

- Open Web Application Security Project (OWASP)
 - <http://www.owasp.org/>
- OWASP AJAX Project
 - <http://www.owasp.org/index.php/Category:OWASP AJAX Security Project>
- OWASP Sprajax Project
 - <http://www.owasp.org/index.php/Category:OWASP Sprajax Project>
- OWASP Testing Guide
 - <http://www.owasp.org/index.php/Testing for AJAX: introduction>

Contact

Dan Cornell

dan@denimgroup.com

(210) 572-4400

Website: <http://www.denimgroup.com/>

Blog: <http://denimgroup.typepad.com/>