Ajax (in)security

Billy Hoffman (bhoffman@spidynamics.com)

SPI Labs Security Researcher
Overview

• Architecture of web applications
• What is Ajax?
• How does Ajax help?
• Four security issues with Ajax and Ajax applications
• Guidelines for secure Ajax development
Architecture of Web Applications
Traditional Web Application

**Browser receives input from user**

*Uses JavaScript for simple logic and optimizations*

*Sends HTTP request across the Internet*

*Server processes response*

  *Backend logic evaluates input (PHP, ASP, JSP, etc)*

  *Possibly access other tiers (database, etc)*

*Resource is returned to user*
Problems with Traditional Web Apps

classic web application model (synchronous)

client
user activity

data transmission
data transmission
data transmission

server
system processing
system processing

http://adaptivepath.com/
Case Study: MapQuest
Reducing the Long Wait

• These long pauses are very noticeable
• Regular applications don't with the user this way
• Reducing the delay between input and response is key
  – Request is a fixed size
  – Response is a fixed size
  – Network speed, latency is fixed
  – Server processes relatively fixed
• Trick the user with better application feedback
• This is what Ajax does
What is Ajax?

Asynchronous JavaScript And XML

JavaScript takes on a larger role

Send HTTP request

Provides immediate feedback to user

Application continues to respond to user events, interaction

Eventually processes response from server and manipulates the DOM to present results
Providing a Rich User Experience

Ajax web application model (asynchronous)

client
browser UI

user activity
input
display
input
display
input
display
input
display
input

Ajax engine
client-side processing
data transmission

server
server-side processing
data transmission

http://adaptivepath.com/
Case Study: Google Maps
More information on Ajax

- Use XMLHttpRequest Object
- Sends any HTTP method
  - Simple: GET, POST, HEAD
  - WebDav: COPY, DELETE
- Limited to where JavaScript came from (hostname, port)
- Fetch any kind of resource
  - XML, HTML, plain text
  - Images, Flash, media content
  - Script (Atlas Framework)

```
function loadXMLDoc(url)
{
    req = new XMLHttpRequest();
    req.onreadystatechange = processReqChange;
    req.open("GET", url, true);
    req.send(null);
}

function processReqChange()
{
    if (req.readyState == 4) {
        if (req.status == 200) {
            //process response...
        } else {
            //handle error...
        }
    }
}
```
Security Issues Surrounding Ajax

- Ajax applications have larger attack surface than traditional applications
- Implementation issues of Ajax make it insecure
  - Repudiation of requests
  - Implementation of “bridging”
- “Buzz” around Ajax is leading to inappropriate use
Attack Surface of Ajax Applications
Architecture of Traditional Web Application

- Application exists almost exclusively on server (except validation)
- Very similar to dumb terminal applications
- Commands from user sent to server
  - User clicked “checkout” button
  - User submits post to forum
  - User navigates to page for product #24
- Application returns whole “new” page
- “What to do” not “How to do it”
Attacks Against Traditional Web Applications

- Attacks involve sending malformed commands
  - User submits post with HTML tags
  - User uploads bad image to corrupt database, exploit
  - User navigates to page for product -1
  - User requests resource without logging in
- These malicious commands hit edge cases in application design
Input Validation for Web Applications

• Ensure commands are correct
• Takes place on both client and server
  – Client validation = performance
  – Server validation = security
• Should be very easy
  – Datatypes of inputs are known
  – Valid ranges and values are known
  – Very limited number of inputs
  – Everything is defined by you
• So how secure are traditional web applications?
Security State of the Union

• Web application vulnerabilities are everywhere!
  – Gartner: “70% of attacks are through the application layer”
  – Majority of posts on mailing lists are web vulnerabilities

• Theory: Input validation is easy on traditional applications

• Reality: Web developers are doing a poor job validating user input
Architecture of an Ajax Application

- Ajax applications extend across client and server
- JavaScript traps user commands, makes \textit{function calls} to server
  - Return price for product #24
  - Return valid cities for given state
  - Return last valid address for user #78
  - Update users age in database
- “How to do it” is being sent over the Internet!
• Attacker is now inside your application!
• Knowledge increases
  – Function/variable names
  – Function parameters and return types
  – Datatypes and valid ranges
• Attack surface increases
  – No longer attacking limited inputs, hoping attacks get through
  – Can audit application's entire API!
• Exploit trust relationship of client and server
Attacks Against Ajax Applications

• 3 types of inputs to filter
  – Forms input
    • Example: form variables, either GET or POST
    • Validation is not implemented enough
  – File Formats
    • Example: Images, ZIP files
    • WMF exploit, numerous PNG/JPG bugs
  – Exposed functions and web services
    • Example: control logic, functions
    • Rarely have validation, if any for QA, unit testing
Function Input Validation in the Wild

- Real world application using Microsoft's “Atlas” framework
- Function input not validated

```csharp
[WebMethod]
public string updatePassword(string custID, string newPassword) {
    //connect to SQL Server (connection string defined earlier
    SqlConnection conn = new SqlConnection(strCn);

    //create query
    string q = "UPDATE tblUser SET Password = '" + newPassword + "' WHERE CustomerID='" + custID + "'");

    SqlCommand cmd = new SqlCommand(q, conn);
    cmd.ExecuteNonQuery();
    return "Password Updated to " + newPassword;
}
```
Repudiation of Requests
(or, “No, I really didn't make that stock trade!”)
Overview of Repudiation

• What is repudiation?
  – Being able to deny you did something
  – Very specific in cryptography
• Pretty important in the computer world today
• How does this apply to Ajax?
More Information on Ajax

- HTTP requests look identical
  - Headers
  - Statekeeping/Authentication tokens (cookies)
- Server cannot discern Ajax requests from browser requests!

```
GET / HTTP/1.1
Host: maps.google.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9) Gecko/20080101 Firefox/3.0.1
Accept: text/xml,application/xml,application/xhtml+xml
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=86e9ce4c2b9dd60:FF=4:LD=en:NR=10:TM=11
```

```
GET /mt?n=404&v=w2.7&x=472&y=794&zoom=6 HTTP/1.1
Host: mt2.google.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9) Gecko/20080101 Firefox/3.0.1
Accept: 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
Connection: keep-alive
Referer: http://maps.google.com
Cookie: PREF=ID=86e9ce4c2b9dd60:FF=4:LD=en:NR=10:TM=11
```
Ajax Design on Repudiation

- You can't prove you didn't cause an XMLHttpRequest to happen
- In other words
  - JavaScript can make requests for resources
  - Request is hidden
  - Happens while you are using the computer
  - Contain authentication information
  - JavaScript can read response, send derived requests
Cross Site Scripting (XSS)

• Cross Site Scripting is injecting script (Javascript or VBScript) into the page that is returned to the user’s browser

• This scripts gets executed by the users browser exposing them to a variety of threats
  – Cookie theft
    • Session hijacking
    • Information leakage
  – Keylogging
  – Screen scrapping
  – Malicious requests
How Cross Site Scripting Works

- Personalized greeting page functioning normally

http://example.com/hello.php?name=Billy

```html
<HTML>
  ...
  <h1>Hello there Billy!</h1>
  ...
  ...
</HTML>
```
How Cross Site Scripting Works

- Personalized greeting page with XSS

http://example.com/hello.php?name=<SCRIPT>badness...

<HTML>
  ...
  <H1>Hello there <SCRIPT>badness...
  ...
  ...
</HTML>
Ajax Empowers XSS

• XSS can now make malicious requests with a user's credentials
• No hard refresh
• Drastically increases damage, information theft XSS can do
• Screen Scrapping
  – Before Ajax
    • Passively scrap what user visits (venus fly trap)
  – After Ajax
    • Actively hunt for specific content (tiger)
    • XSS/Ajax crawler: Scrap for data and links, repeat
    • Can access resources/data not available to passive
Injecting and Propagating XSS

• Before Ajax
  – By hand by a person
  – Typically can only inject one part of a website
    • Reflection based XSS
    • Stored based XSS
  – Self-propagation
    • Use Image object, DOM tricks to send requests
    • Limited to GET, sometimes POST
    • Blind requests, cannot see response
Injecting and Propagating XSS

- After Ajax
  - XSS payload can now autonomously inject itself into pages
  - Easily re-inject same host with more XSS
  - Can do this all seamlessly (no hard refresh)
  - Can send multiple requests using complex HTTP methods to accomplish propagation
  - Sound like a virus doesn't it
  - It is
Analysis of MySpace.com Virus

- Web virus
- October 2005: Infected 5th largest domain on the Internet
- JavaScript with Ajax
- Attack vector: XSS exploit allowed <SCRIPT> into user’s profile
- Propagation:
  - Used Ajax to inject virus into the user profile of anyone who viewed an infected page
- Payload:
  - Used Ajax to force viewing user to add user “Samy” to their friends list
  - Used Ajax to append “Samy is my hero” to victim’s profile
Blacklisting Input is Hard!

- MySpace.com did a pretty good job filtering, but used blacklisting.
- Blocked things like `<SCRIPT>`, `innerHTML`, and special characters
- Samy used `eval` statement
  - Allows you to execute JavaScript statements stored inside a string
  - Stored his payload in the DOM, used `eval`
- `<DIV id="code expr="alert('xss')" style="background('javascript:eval(document.code.expr))">
- See [http://namb.la/popular/tech.html](http://namb.la/popular/tech.html) for all the details
MySpace Infection Method Explained

profile.myspace.com -> www.myspace.com

Switch Domains

GET Confirm page (hash)

Extract hash, POST payload
Analysis of MySpace.com Virus

• Awesome hack!
• No, I didn’t write it.
  – I did present about XSS + Ajax attacks at Toorcon 7 a month before the virus hit
• Excellent proof of concept about how using Ajax is a security risk even though it obeys the DOM security model
  – Web server cannot tell the difference between Ajax requests and web browser requests
• Shows how Ajax + JavaScript RegExs can handle complex login sequences spanning multiple pages
• MySpace lucked out as it could have been much worse
Design problems with Ajax

- Request made using XMLHttpRequest are identical to normal requests
- The server cannot tell the difference
- You cannot repudiate the traffic created by Ajax
- Ajax makes XSS scarier
  - XSS can self propagate
  - XSS can make complex requests
  - Consider the MySpace attack inside a stock trading web application...
Ajax Bridging
What is an Ajax Bridge?

• Hack to overcome Ajax limitation
• Provides a “bridge” for client-side script to access 3rd party
• “Web service to another web service”
• How it works
  – Ajax can only connect back to host it came from
  – To allow script to access 3rd part, host provides bridge
  – Bridge acts as a proxy, forwarding traffic between parties
• Support
  – Many frameworks include bridges (“Atlas,” PHP add ons)
  – Even more are custom (CGI gateways, etc)
Data over the Bridge

• What 3rd parties can the bridge connect to?
  – Web service on a host
    • SOAP/REST/custom
  – Arbitrary web resource on a host
    • RSS feeds
    • Raw content (HTML, Flash, etc)
• What can the bridge return?
  – JSON
  – XML
  – Raw resource/custom
Exploiting Mash-up Trust

- This type of aggregate of 3\textsuperscript{rd} party services is encouraged
  - Mash-up
- Any web service is fair game
- Any commercial use requires a more formal agreement
  - These formal agreements care some kind of trust
  - Aggregator will and will not do certain things
  - Bridge provide a hole around most attempts to enforce these rules
  - Bridge allows malicious users to exploit this trust relationship
Security Issues with Bridges

- Open proxy that only goes one place
- No built in security mechanisms
  - No throttling
  - No authentication
  - No authorization
  - No obfuscation
- Relies on other components for security
  - We have seen how well that works
- Multiple attack vectors through an Ajax bridge
Dumping Database Through a Bridge

• Sure users can get free key to access Amazon's API
• Key for book lover's aggregate site will have more privileges than standard keys
  – More simultaneous connections
  – Higher bandwidth limits
  – More unique queries
• Performance enhancements only help me!
  – Aggregate site using shared cache for queries?
  – Dump results even faster
Hacking 3\textsuperscript{rd} Parties Through Bridges

- Start sending XSS, SQL injection, etc web attacks
  - Maybe 3\textsuperscript{rd} party notices
    - Google: “Why is housingmaps.com” SQL injecting me?
    - Another layer to hide behind
  - Maybe auto-shunning IDS/IPS along the way notices
    - IPS: “This site is SQL injecting me!” [Blocks IP]
    - Wanted SQL injection, got DoS of aggregate site
  - Maybe 3\textsuperscript{rd} party doesn't notice at all
    - Large site lots of requests from affiliates
    - Performs less analysis. Attacks only work through bridge
Layers of Indirection

• Really 2 connections here
• Bridges can perform transforms on the data at any time
  – User to 3rd party
  – 3rd party to user
• Transforms are useful for regular users
  – Convert datatypes
  – Naming conventions
  – Nationality conversions
• Don't transforms prevent our attacks?
“I Once Was Blind, But Now I See...”

- Results of web attack against 3rd party proxied by bridge may get filter/munged/converted
  - Useful error messages will be lost
- Unfortunately, this is a mature area of study
- Blind SQL injection methodology
  - Send good request, bad request, take the difference
  - Can know identify if attack work
- Attackers can use blind SQL injection methodology for other attacks
  - LDAP, XSS, content injection, etc
Hype Surrounding Ajax
All About the Money

• Web 2.0 almost like another bubble
• Lots of money in the air
  – MySpace.com bought for $400 million
  – Writely.com purchased by Google
  – Del.icio.us bought by Yahoo $50 million
  – Facebook turned down $700 million, wants $2 billion
  – O'Reilly's $1500 Where, Web 2.0, other conferences
• Not judging right or wrong
• Blood in the water
Buzzwords in the Air

- Ajax
- Web 2.0
- Social Networking
- Folksonomy
- Blogging
- Collaborative Filtering
- Gadgets
- Reputation Systems
- Tagging
- Swarming
- Mash-ups
- RSS feeds
- Podcasts
- Perpetual beta
- The Long Tail
- Citizen Media
- Traceback
- Rich User Experience
- Permalink
- Cluster mapping
Affects of the Hype

- Move to client-side web application for wrong reasons
  - To appear trendy
    - Office/Windows Live? Come on now.
  - To fight off (perceived) competitors
  - New frontend fun use of time
    - Sexy to customers
    - Companies like it more than bug fixes, or security
  - Because they can
Affects of the Hype

• Driving Inexperienced people into the web applications
  – “You can make money in web apps”
  – Similar to tech boom in the 90s
  – Demand spikes
  – Encourages cut+paste application development
• Technical resources suffering
  – More tutorials aimed at beginners
    • Examples do not include best practices
    • Examples are often wrong on security topics
      – Blacklisting vs. whitelisting
Affects of the Hype

• Technical resources are suffering (continued)
  – Book quality suffers
    • Tech rot forces fast production
    • Contains mistakes
    • Oversimplify important topics
      – “Teach yourself ASP.NET in 24 hours”
      – “Teach yourself enough ASP.NET in 24 hours to create an insecure web application”
  • Topics beyond the books are unknown to most users
Security Ramifications of Hype

• Rush to “Web 2.0”-ize applications with no thought
• Some applications shouldn't be web applications, let alone Ajax
• API's being exposed that never should be
• More and more people are novices
  – Signal to noise ratio approaches zero
    • In forums looking for answers
    • In companies at project meetings
  – Security conscience people become rare
    • Holding up development?
End result of Ajax Hype

• Already way too many web application vulnerabilities
  – BugTraq is swamped with reports
  – php[whatever] exploit of the week
  – Can't browse the Internet without finding new vulnerabilities
• And this was with reasonably educated web developers
  – We are already seeing bad design choices
Designing For Ajax Applications
Design Checklist

• Should you Ajax enable a web application?
  – What is gained?
  – Is it necessary?
• Retrofitting an old application
  – Document current inputs
  – Ensure input validation all existing inputs
• Design Ajax application
  – Minimize the program logic you need to expose
  – Implement validation on all function input
  – Use clear standards, not hacks
Conclusions

- Ajax is a powerful suite of technologies
- Ajax can improve user experience on *appropriate* applications
- Ajax applications live on the client and server
  - Increases the attack surface
  - Exposes internal functions, control layer
  - Creates many more inputs to secure
- Ajax bridges can be extremely dangerous
  - Its input, so secure it
  - Additional security measures
- Hype around Ajax may be deserved, but hurting the industry
Questions?
Ajax (in)security

Billy Hoffman (bhoffman@spidynamics.com)

SPI Labs Security Researcher