Hacking Browser's DOM Exploiting Ajax and RIA

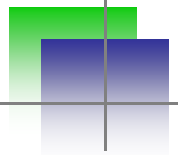
Shreeraj Shah

Blueinfy

Who Am I?

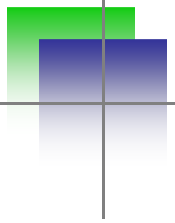
http://shreeraj.blogspot.com
shreeraj@blueinfy.com
http://www.blueinfy.com

- **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

- Attacks and Trends
  - Cases, Client Side and Patterns
- DOM and Application Architecture
  - Layout, Browsers, DOM and DOM's Attack Surface
- DOM based Attacks
  - DOM based XSS, Widget Hacking, Feeds and Mashup injections, Reverse Engineering, Logic leakage, CSRF with XML/AMF/JSON etc.
- Defense and Countermeasures
- Conclusion & Questions



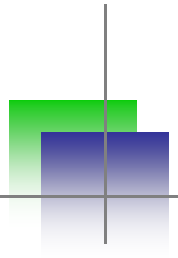
Attacks and Trends

Real Life Cases

- Reviewed – Banks, Portal, Telecom etc.
- Complex usage of DOM both by developers and libraries
- Vulnerabilities detected
  - XSS with DOM
  - Widgets and Mashup injections from DOM
  - Logic bypass
  - Other …

Multiple **DOM-Based XSS** in Dojo Toolkit SDK - msg#00133 - bugtraq ...
24 Jun 2010 ... More information on **DOM-based XSS** can be found at ..... **yahoo**-IBOVWKEEIL4GQU5DM3GOZUJ62U June 22, 2010, 7:19 pm UTC ...
osdir.com/ml/bugtraq.security/2010-03/msg00133.html - Cached

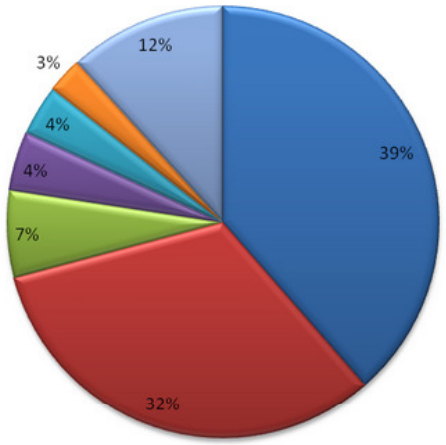
Full Disclosure: Yahoomail **Dom Based XSS** Vulnerability
13 Jun 2010 ... Title: **Yahoo** mail **Dom Based Cross Site Scripting** Auth
pratulag[at]**yahoo**[dot]com> Date: 13/06/2010 Indian Hacker Service: ...
seclists.org/fulldisclosure/2010/Jun/289



Client Side Attacks

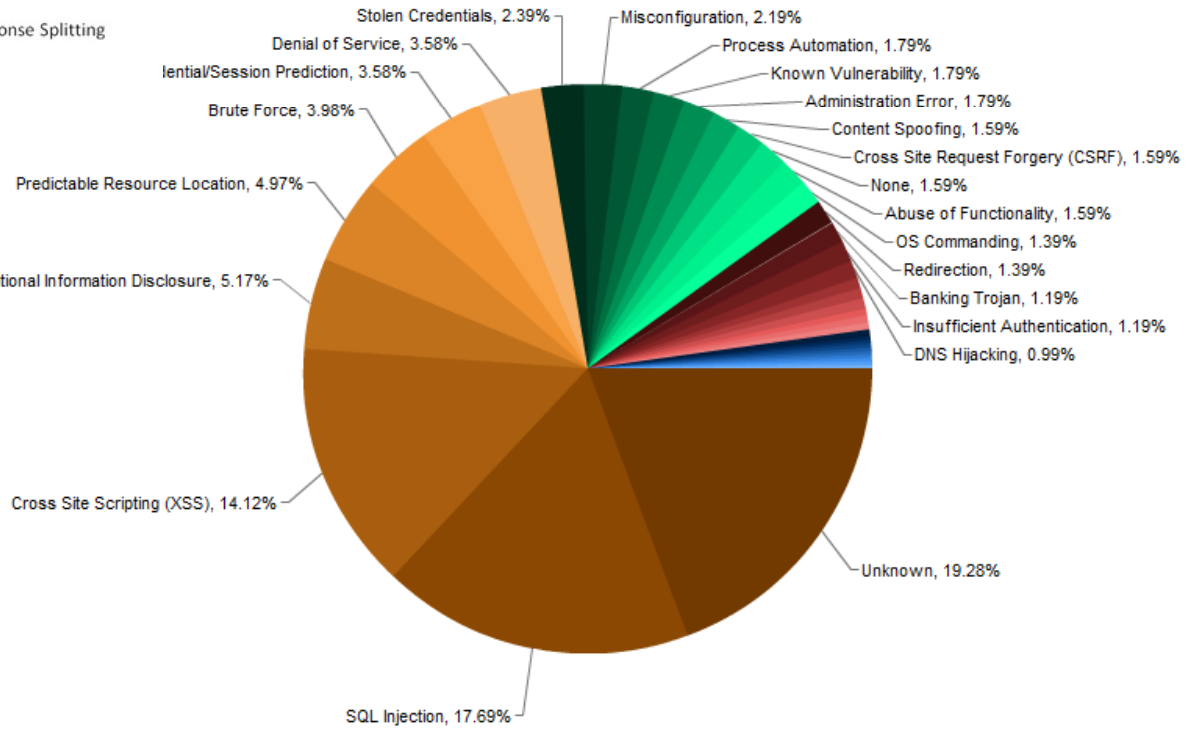
- Malware and Attacks are centered around browser
- DOM is an active part of Browser and popular attack point
- XSS is one of the major threats to applications
- CSRF and some other client side attacks are on the rise.
- Web 2.0 exposing attack surface – Widgets, Mashups etc.

Attacks & Exploits



- Cross-Site Scripting
- Information leakage
- SQL Injection
- Insufficient Transport Layer Protection
- Fingerprinting
- HTTP Response Splitting
- Other

Client side attacks & DOM hacks



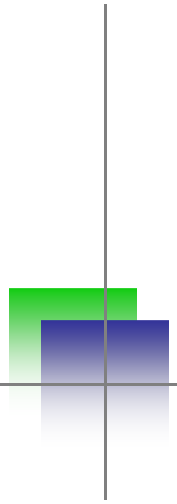
Source - WASC

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

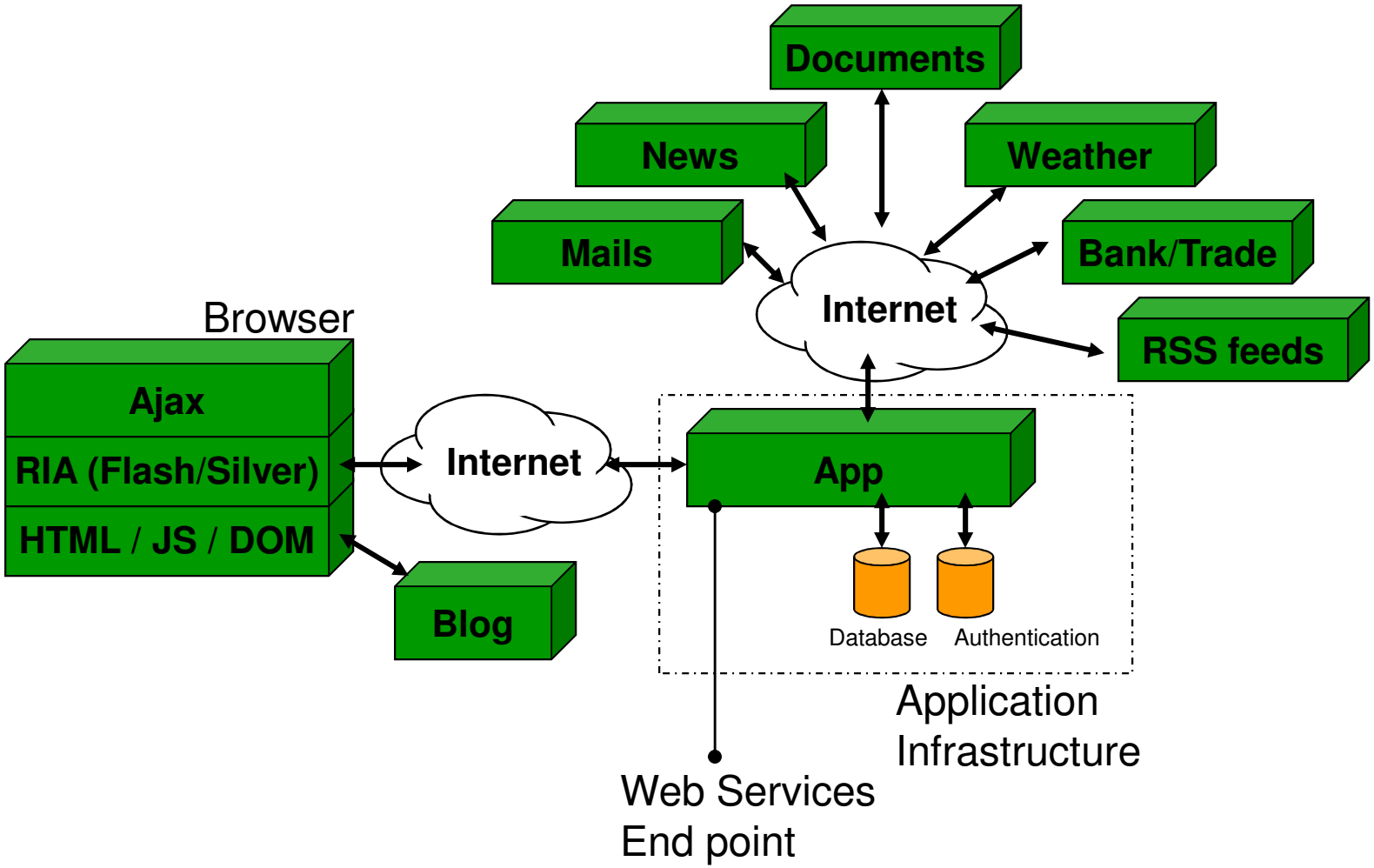
OWASP Top 10 – 2007 (Previous)	OWASP Top 10 – 2010 (New)
A2 – Injection Flaws	A1 – Injection
A1 – Cross Site Scripting (XSS)	A2 – Cross Site Scripting (XSS)
A7 – Broken Authentication and Session Management	A3 – Broken Authentication and Session Management
A4 – Insecure Direct Object Reference	A4 – Insecure 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

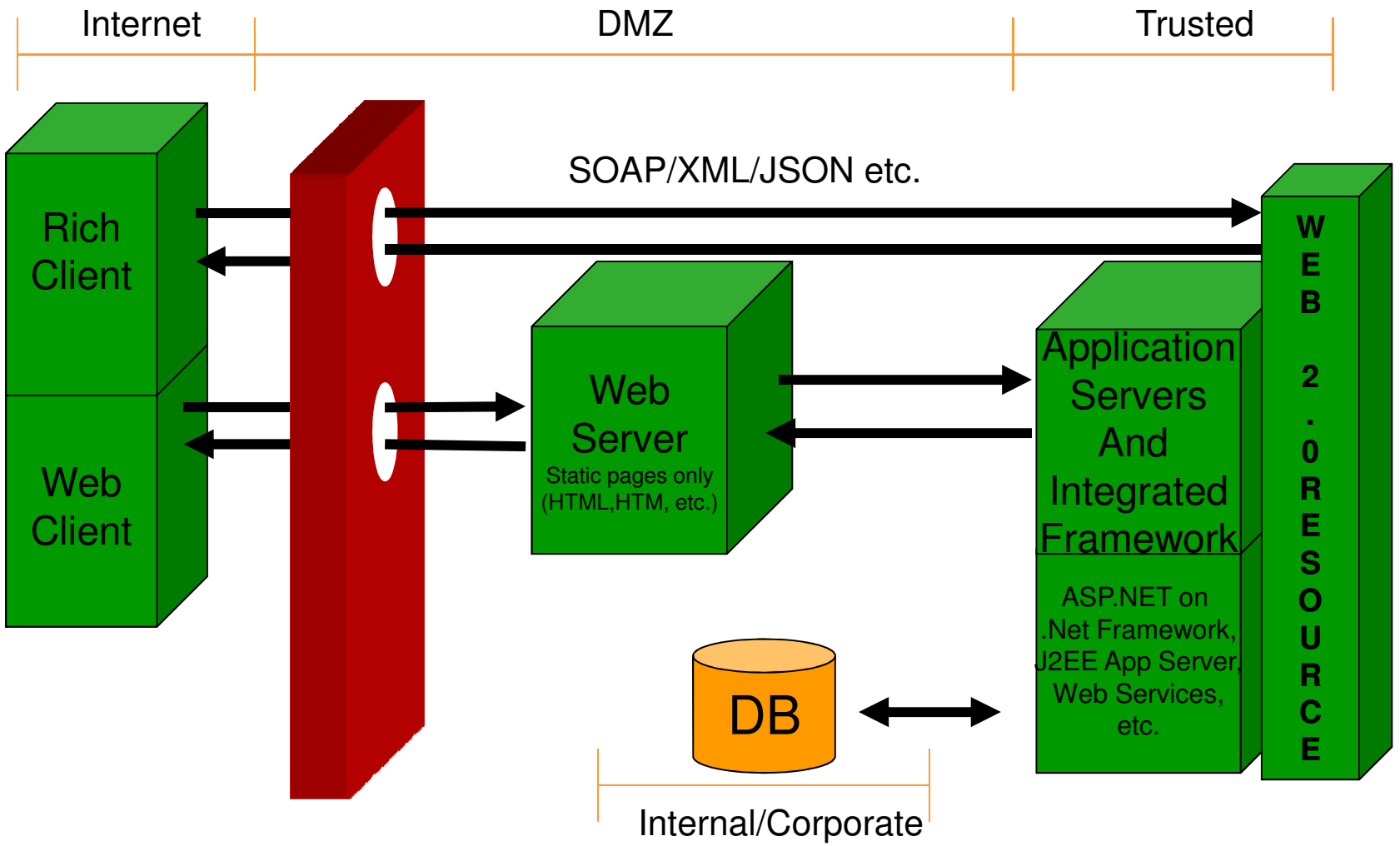


Architecture and DOM

Web 2.0 & DOM usage



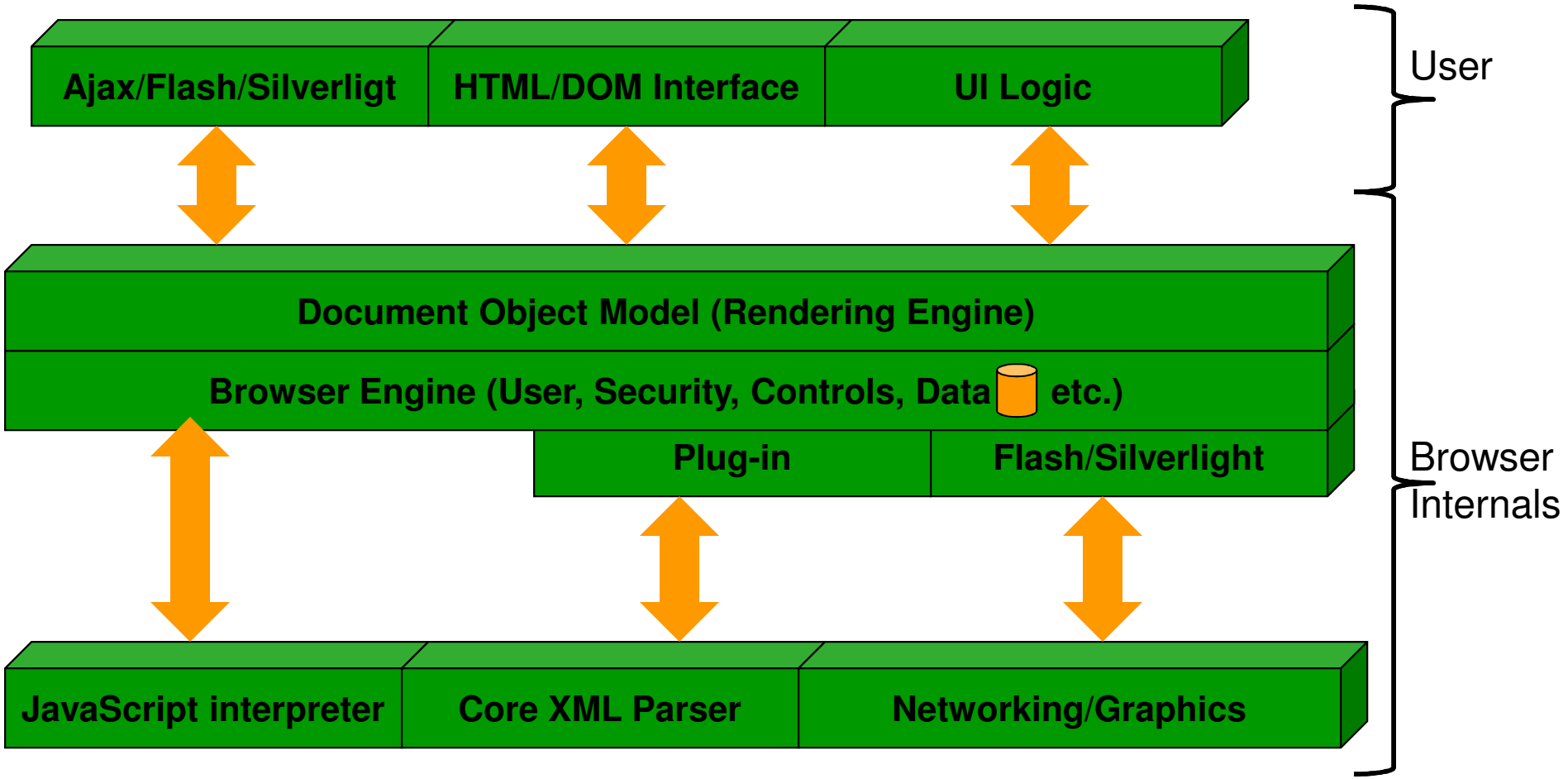
Application Layout



Demos

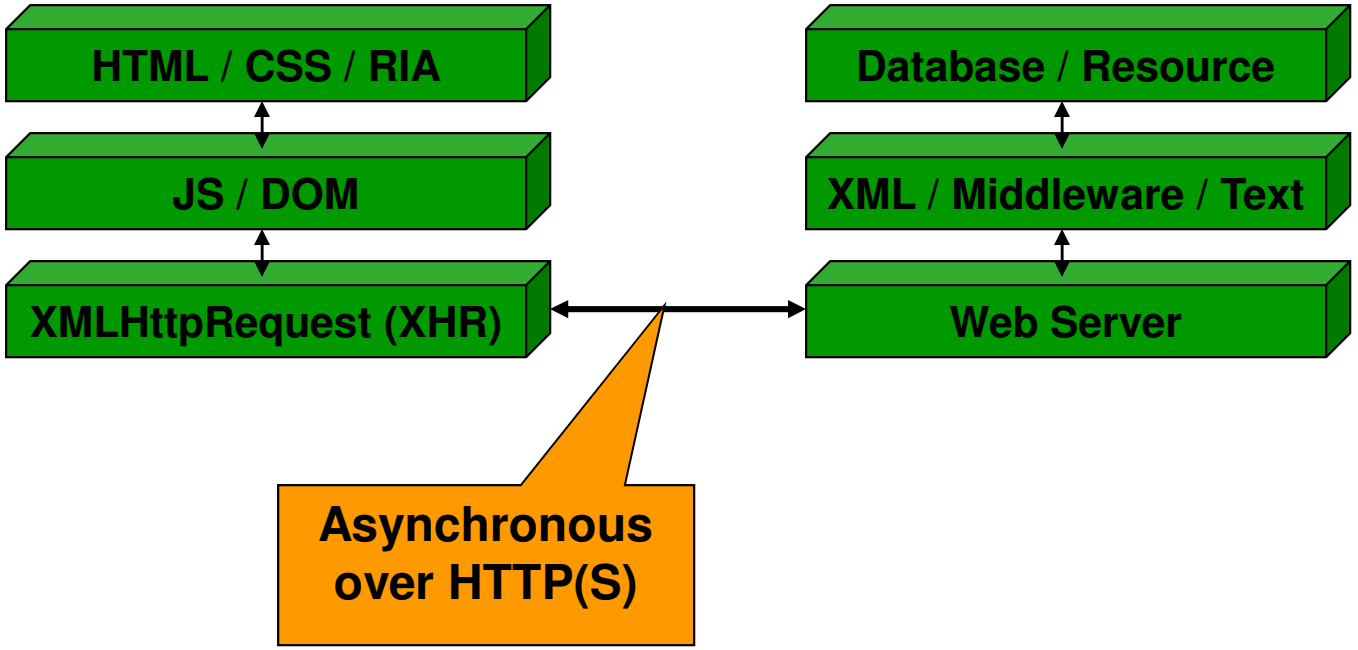
- Web 2.0 Application Demo ★
- Identifying backend resources hidden in the DOM or JavaScripts ★
- Quick look at Java based 2.0 applications – DWR/Struts ★

Browser/Application View

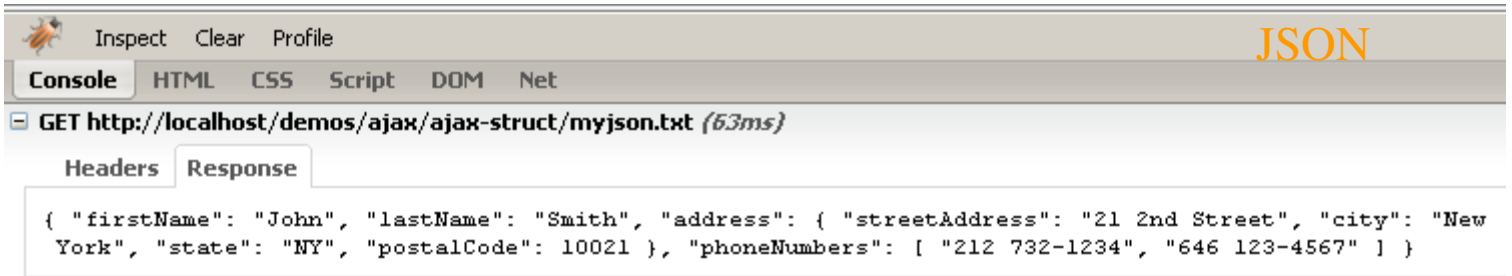


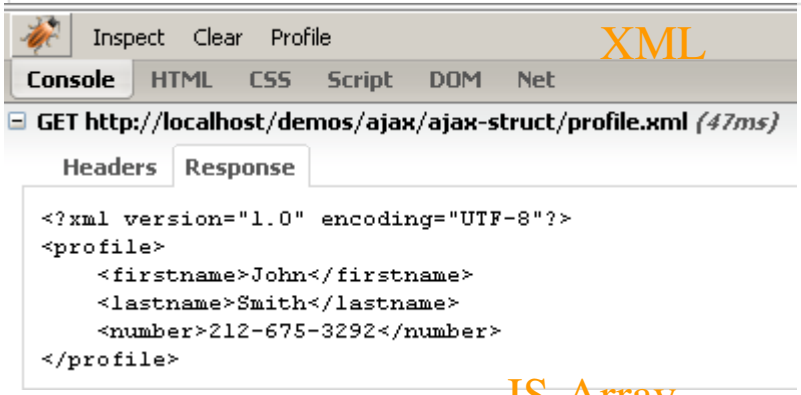
DOM Calls

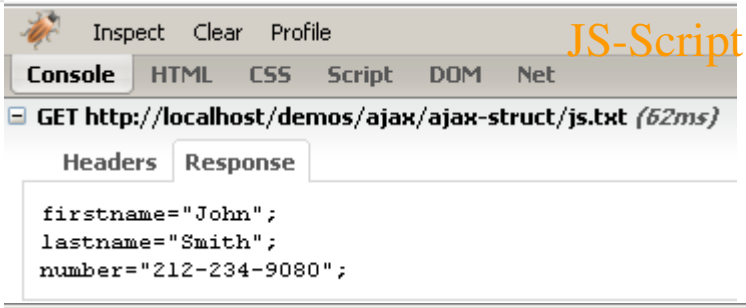
- Ajax/Flash/Silverlight – Async Calls

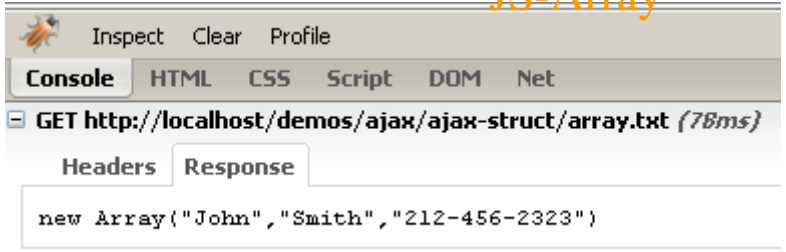


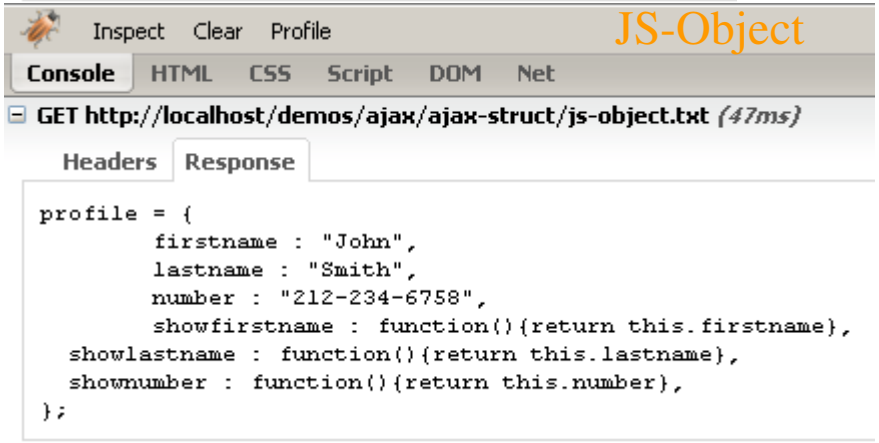
DOM Calls


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";

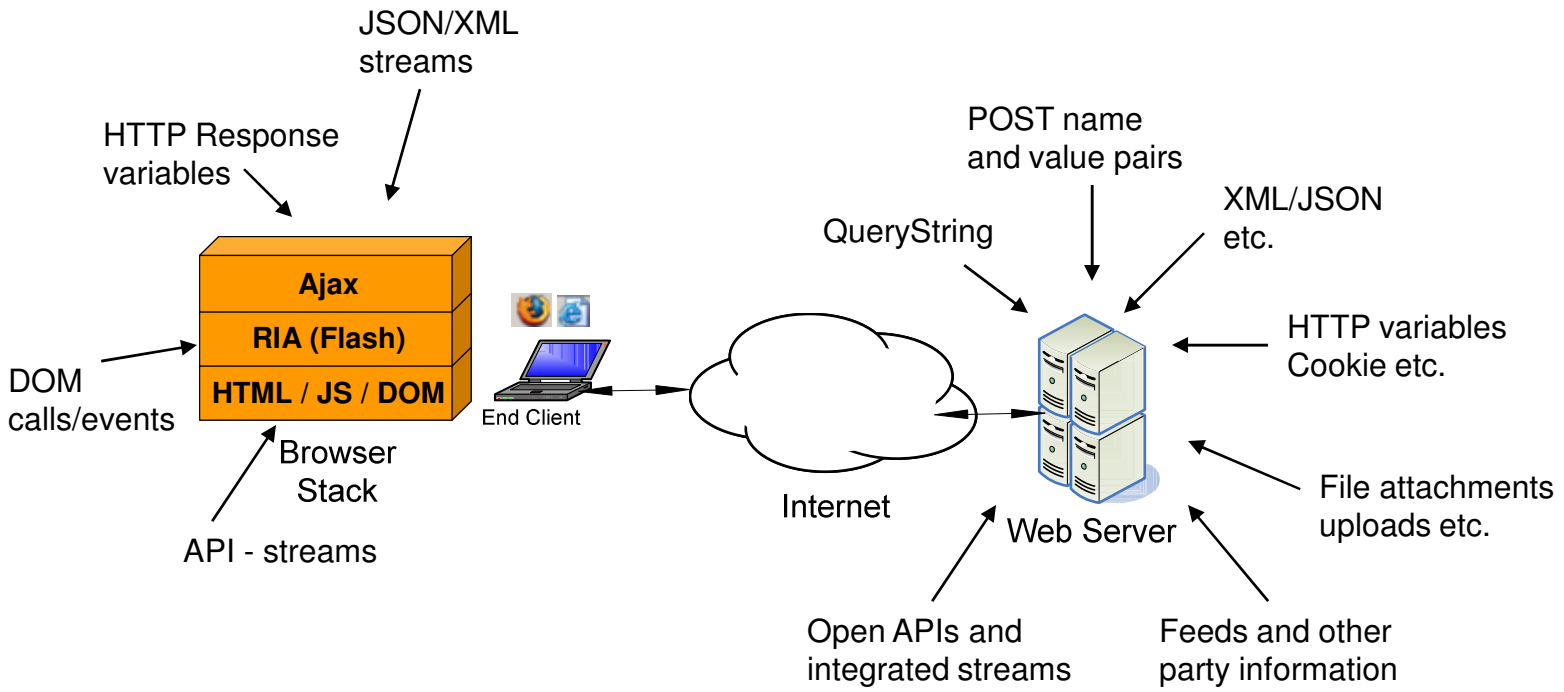

JS-Array
Inspect Clear Profile
Console HTML CSS Script DOM Net
GET http://localhost/demos/ajax/ajax-struct/array.txt (78ms)
Headers Response
new Array("John","Smith","212-456-2323")

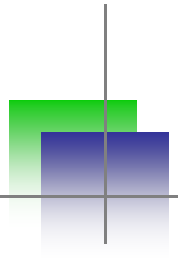

Inspect Clear Profile
Console HTML CSS Script DOM Net
JS-Object
GET http://localhost/demos/ajax/ajax-struct/js-object.txt (47ms)
Headers Response
profile = {
firstname : "John",
lastname : "Smith",
number : "212-234-6758",
showfirstname : function(){return this.firstname},
showlastname : function(){return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

Demos

- Challenge for automation – DOM fetch and harvesting
  - Can’t crawl and extract sites
  - DOM drivers required
  - DOMScan – Loading the DOM and extracting links

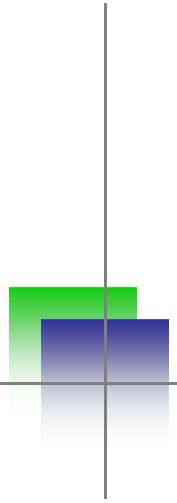
Attack Surface



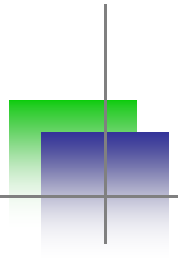


DOM Hacking

- DOM based XSS
- DOM based request/response/variable stealing
- Flash and DOM access – Cross Technology access
- Widgets hacking with DOM
- Feeds and Mashup – DOM manipulations
- CSRF with JSON/XML/AMF (SOP bypass/Proxy channel)
- DOM reverse engineering

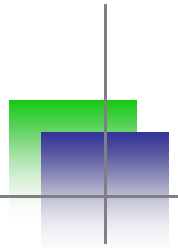


DOM based XSS



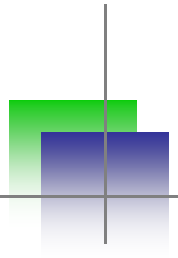
DOM based XSS

- 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



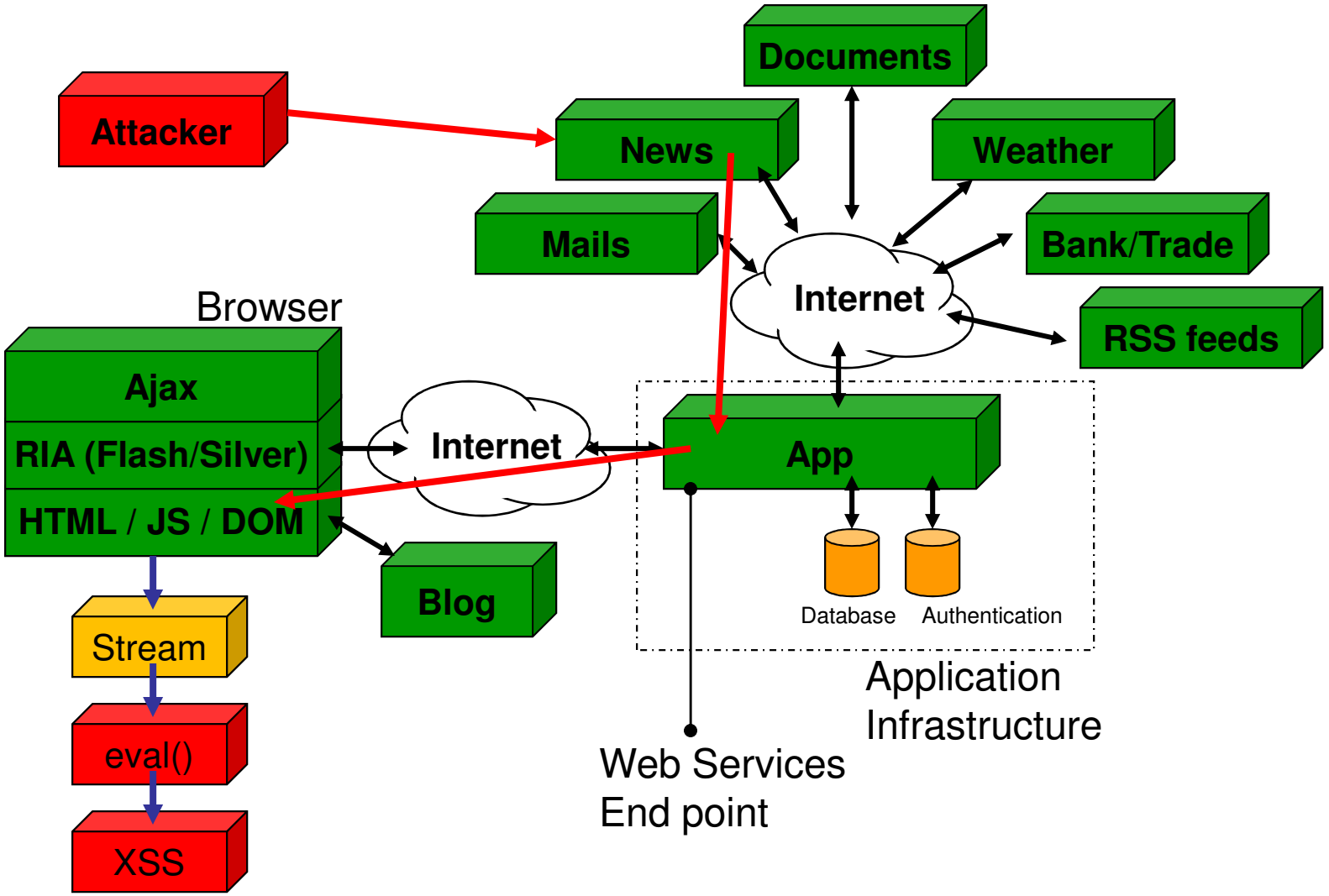
1. DOM based URL parsing

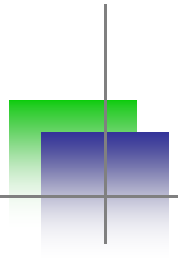
- Ajax applications are already loaded and developers may be using static function to pass arguments from URL
- For example
  - hu = window.location.search.substring(1);
  - Above parameter is going to following ajax function
    - eval('getProduct('+ koko.toString()+')');
  - DOM based XSS

Demo

- Scanning with DOMScan 
- Injecting payload in the call

2. Third Party Streaming

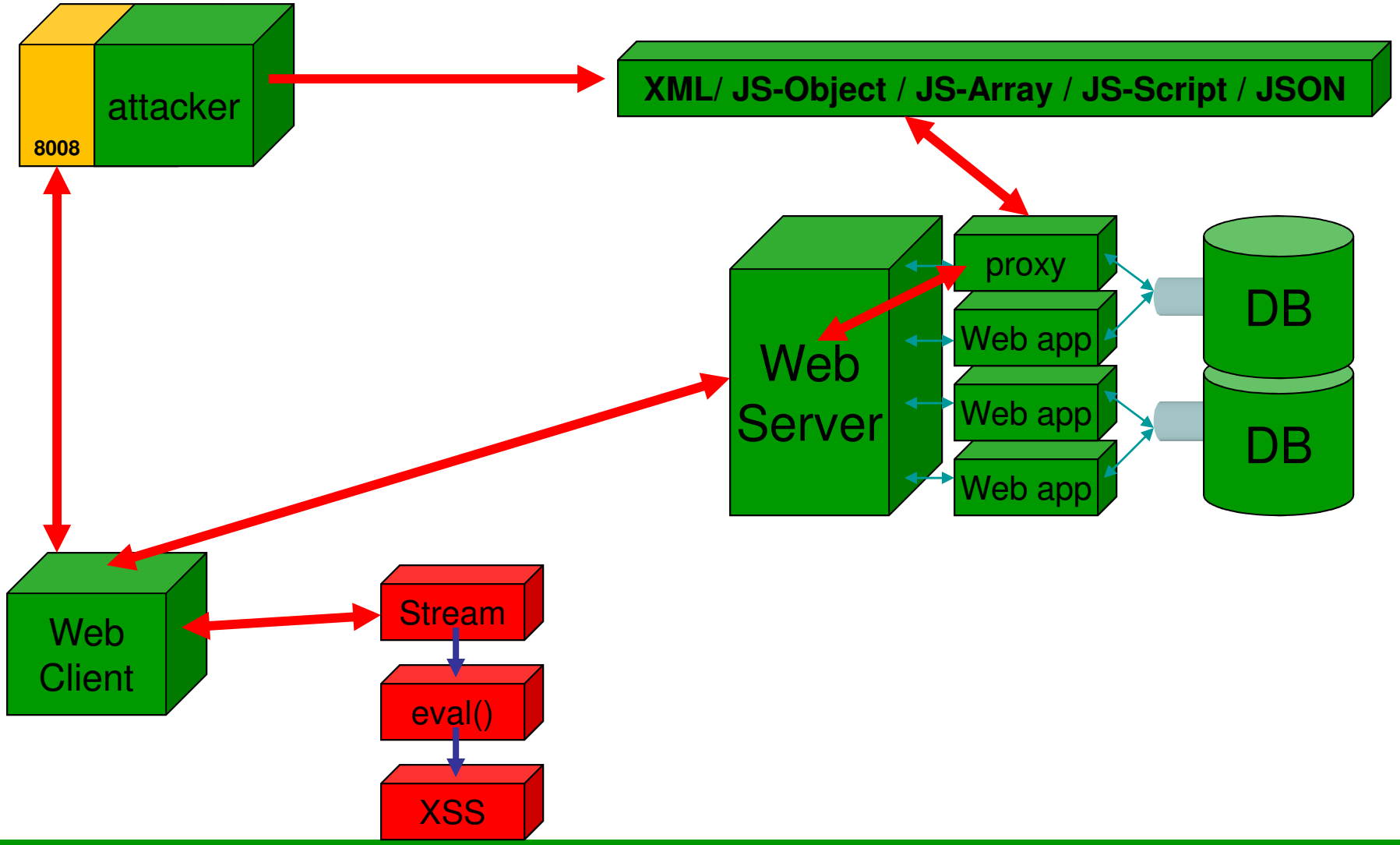




Stream processing

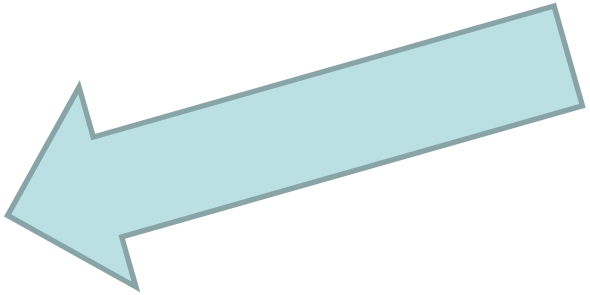
```
if (http.readyState == 4) {
        var response = http.responseText;
        var p = eval("(" + response + ")");
       document.open();
       document.write(p.firstName+"<br>");
       document.write(p.lastName+"<br>");
       document.write(p.phoneNumbers[0]);
       document.close();
```

Polluting Streams



Exploiting DOM calls

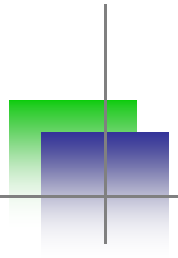
document.write(…)
document.writeln(…)
document.body.innerHtml=…
document.forms[0].action=…
document.attachEvent(…)
document.create…(…)
document.execCommand(…)
document.body. …
window.attachEvent(…)
document.location=…
document.location.hostname=…
document.location.replace(…)
document.location.assign(…)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



Example of vulnerable Calls

Demo

- Sample call demo 
- DOMScan to identify vulnerability 

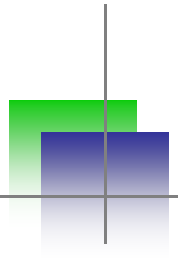


3. Direct Ajax Call

- 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…

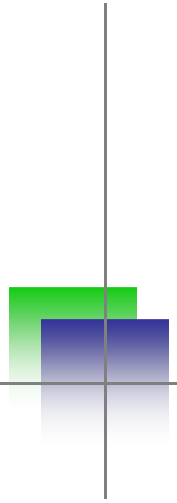
Demo

- DWR/JSON call – bypassing and direct stream access 

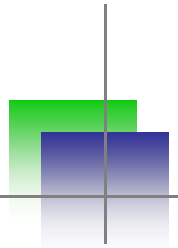


Nutshell - DOM based XSS

- It is very common now a days
- Other instances or possible areas
  - Callbacks directed to DOM
  - HTML 5 and some other added tags and attributes like autofocus, formaction, onforminput etc.
  - Third party JavaScript processing
  - innerHtml calls
  - Many different ways it is possible
- Watch out in your applications



Accessing from DOM

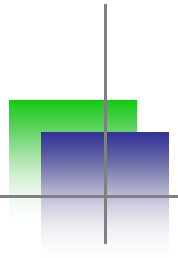


Action in DOM

- 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?

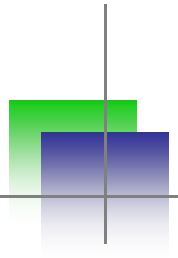
```
1  function getLogin()
2  {
3
4  gb = gb+1;
5  var user = document.frmlogin.txtuser.value;
6  var pwd = document.frmlogin.txtpwd.value;
7  var xmlhttp=false;
8   try  {   xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
9
10  }
11  catch (e)
12  {    try
13     {    xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");    }
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

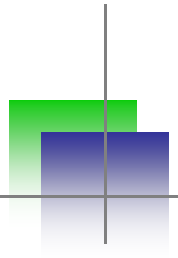
- Here is the line of code

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



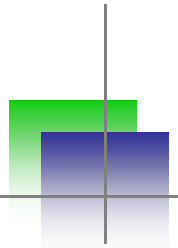
DOM stealing

- 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

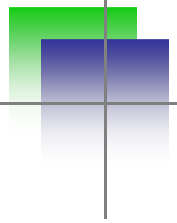


Accessing Flash Data

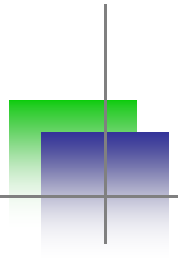
- Flash or Silverlight running in the browser
- It is sharing same DOM
- DOM based XSS can retrieve variables from the flash object
- In some cases depending on the scope one can craft an attack to retrieve these values
- If these files are using set of parameters then possible to exploit.

Demo

- Simple decompilation ★
- Cross Technology Access and exploiting XSS for fetching flash variables ★
- Flash loading Flash through DOM ★



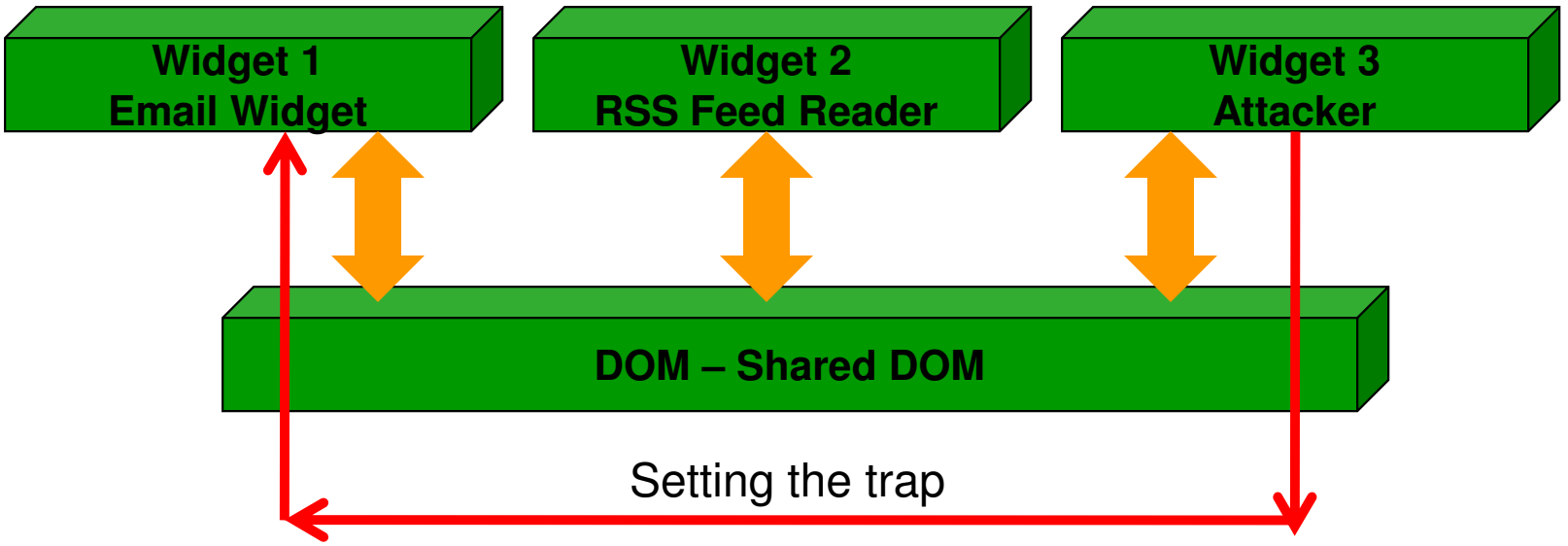
Widget Hacking

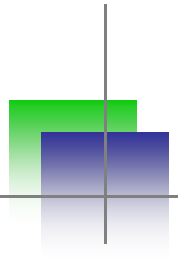


Widgets

- 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



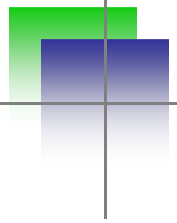


DOM traps

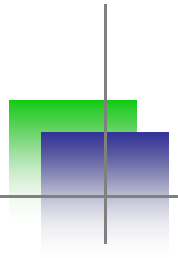
- It is possible to access DOM events, variables, logic etc.
- Sandbox is required at the architecture layer to protect cross widget access
- Segregating DOM by iframe may help
- Flash based widget is having its own issues as well
- Code analysis of widgets before allowing them to load

Demo

- Cross Widget Spying
- Using DOMScan to review Widget Architecture and Access Mechanism



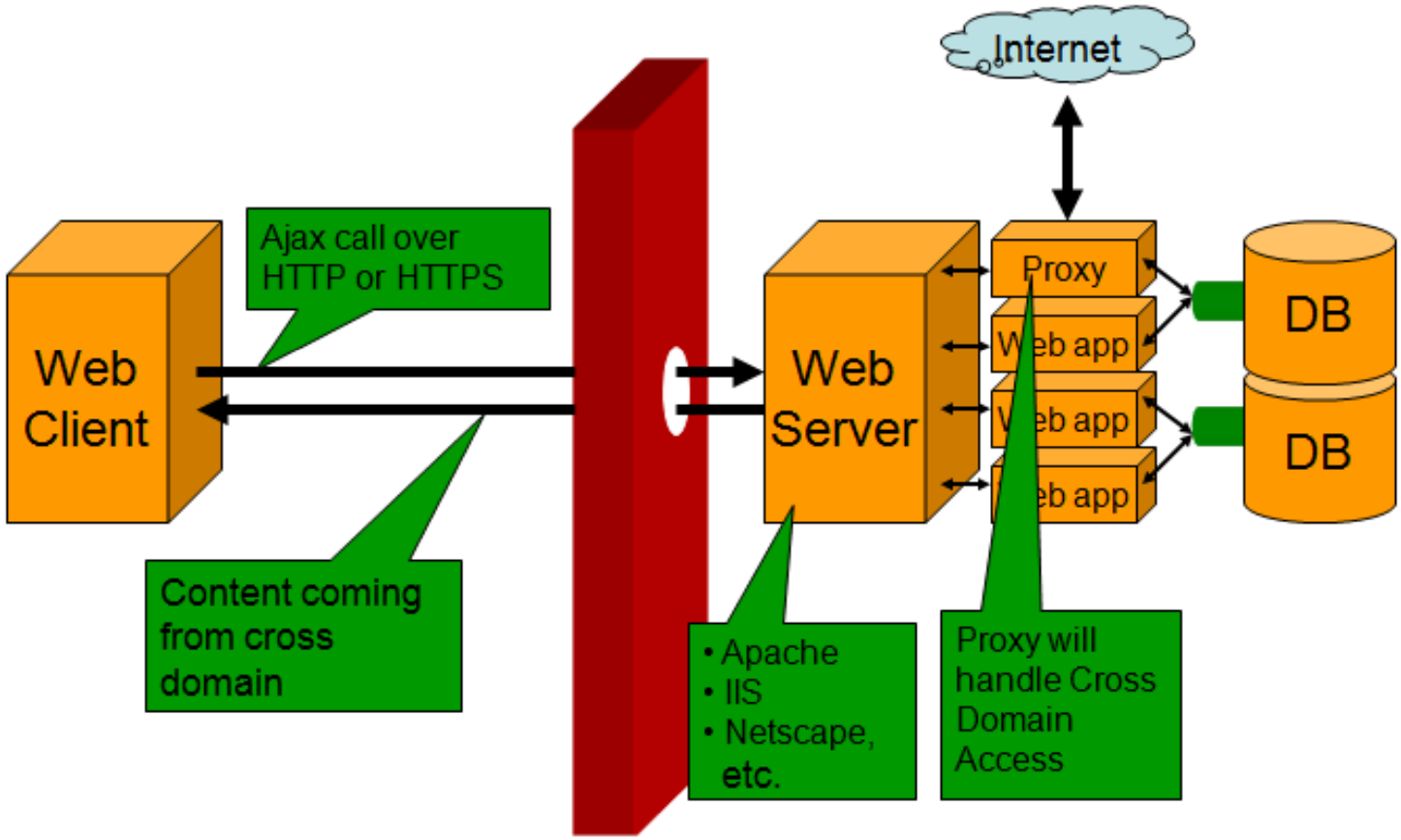
Feeds and Mashup Hacking



Feeds and Mashups

- XML driven feeds – RSS or ATOM, popular for data sharing
- It tunnels through the application
- Sources are not known or untrusted
- It can be registered by user itself
- Mashups are man in the middle and allow aggregation of data sources
- Opens attack surface

SOP bypass and stream access



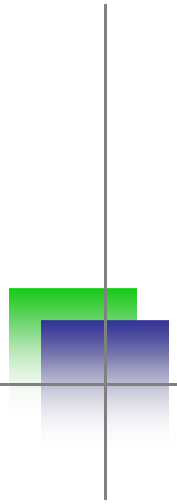
Feed Hacking and Mashups


```
<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);'
```

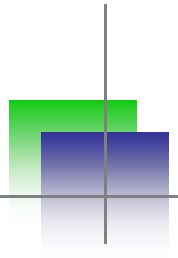
```
//-------------------------------------------------------------
//-------------------------------------------------------------
function processRSS (divname, response) {
    var html = "";
    var doc = response.documentElement;
    var items = doc.getElementsByTagName('item');
    for (var i=0; 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;
}
```

Demos

- RSS Feed Hacking 
- Mashup Hacks 
- Cross Domain Callback Hacking 



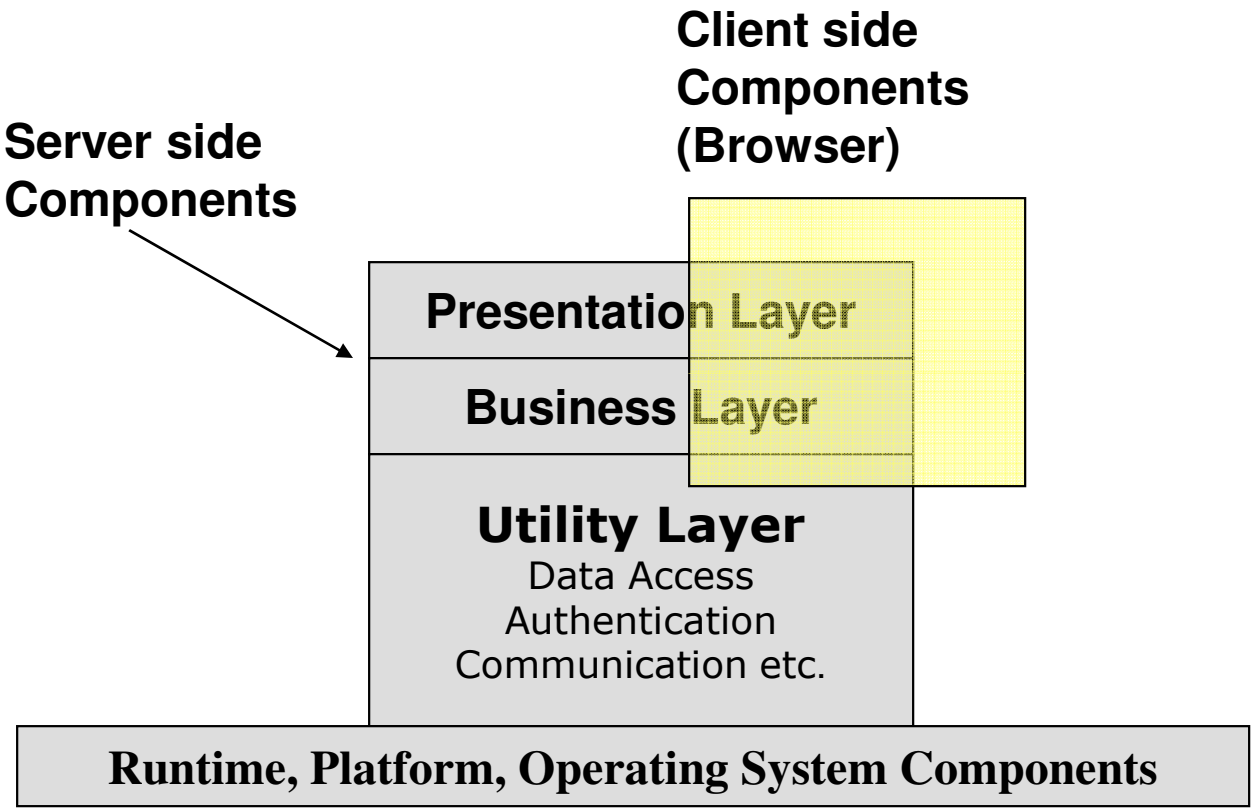
DOM reverse engineering

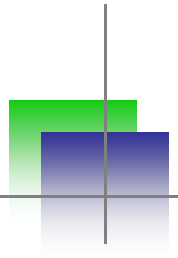


Reverse Engineering

- It is easy to reverse engineer the application
- If JavaScript then possible to profile or debug the script
- It shows interesting set of information
- Also, decompiling Flash and Silverlight may show cross DOM access
- It leads to possible vulnerabilities or exploitation scenario

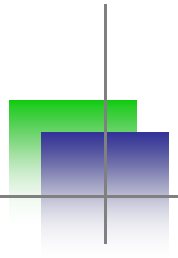
Layers in the client code





Demos

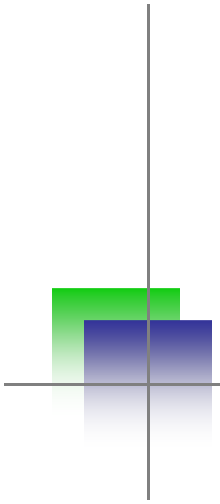
- Analyzing JavaScript and accessing logic directly ★★
- Decompiling Flash and Silverlight ★



Countermeasures

- Threat modeling from DOM perspective
- JavaScript – Static code analysis
- Source of information and dependencies analysis
- Proxy level of filtering for all Cross Domain Calls
- Content-Type checks and restrictions
- Securing the DOM calls

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



Conclusion and Questions