# Unraveling some of the Mysteries around DOM-based XSS

Dave Wichers
Aspect Security, COO
OWASP Board Member
OWASP Top 10 Project Lead

ASPECT SECURITY
Application Security Experts

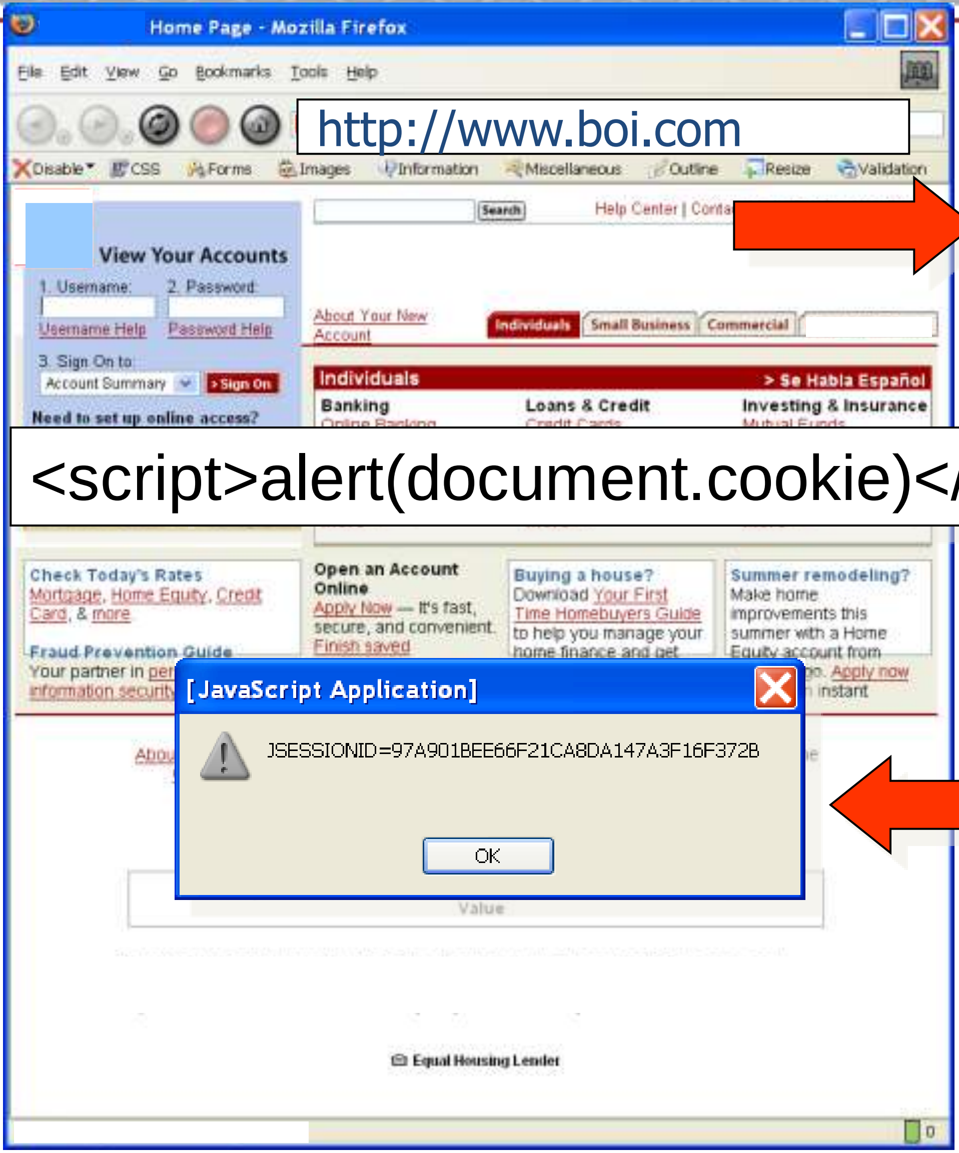
dave.wichers@aspectsecurity.com

# Me – Dave Wichers

- COO / Cofounder – Aspect Security
  - 20+ Years in Application Security Consulting
  - Code Review / Pen Testing 100s of Applications
  - Taught 100s of Secure Coding Courses
  - Helped numerous organizations improve their ability to produce secure software
  - Board Member, OWASP Top 10 Lead, ASVS Coauthor
    - Open Web Application Security Project (OWASP)
- Aspect Security (www.aspectsecurity.com)
  - Focused exclusively on Application Security
  - Code Review / Pen Testing of Applications
  - Secure Coding Training (over a dozen courses including secure mobile)
  - Secure Development Lifecycle consulting
    - Tools, Techniques, Processes for Developing Secure Code
  - Contrast – Intrinsic Application Security Testing (IAST) for JavaEE

    https://www.aspectsecurity.com/Contrast

# XSS – Illustrated


Home Page - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Disable CSS Forms Images Information Miscellaneous Outline Resize Validation
Search
Help Center | Conta
View Your Accounts
1. Username: 2. Password:
Username Help Password Help
3. Sign On to:
Account Summary
› Sign On
Need to set up online access?
About Your New Account
Individuals Small Business Commercial
Individuals
› Se Habla Español
Banking
Loans & Credit
Investing & Insurance
Check Today's Rates
Mortgage, Home Equity, Credit Card, & more
Fraud Prevention Guide
Open an Account Online
Apply Now — It's fast, secure, and convenient.
Finish saved
Buying a house?
Download Your First Time Homebuyers Guide to help you manage your home finance and get
Summer remodeling?
Make home improvements this summer with a Home Equity account from
Apply now
[JavaScript Application]
OK
Equal Housing Lender

http://www.boi.com

<script>alert(document.cookie)</script>

JSESSIONID=97A901BEE66F21CA8DA147A3F16F372B

Search-field input is often reflected back to user.

Site reflects the script back to user where it executes and displays the session cookie in a pop-up.

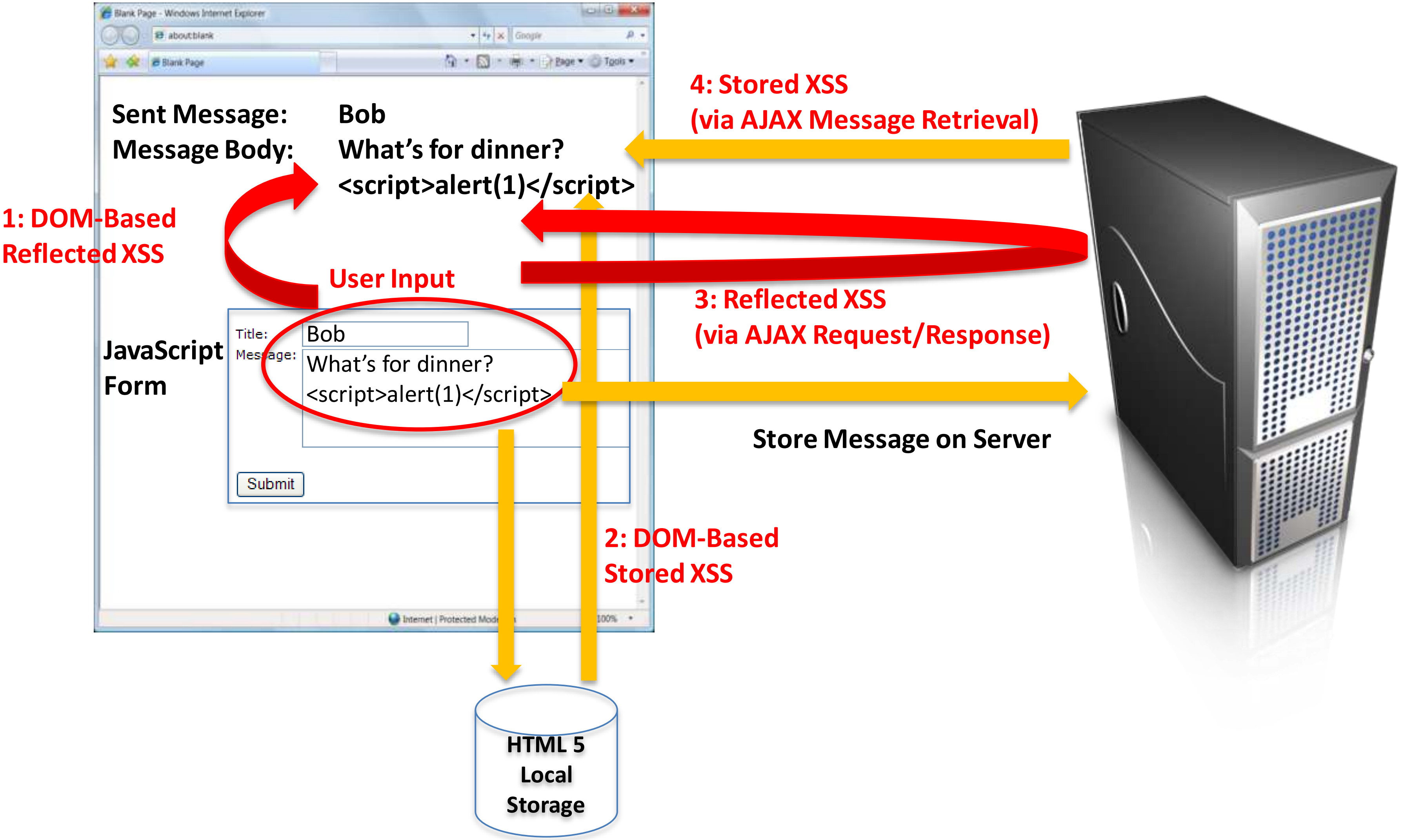
# Types of Cross-Site Scripting (XSS)

| Cross-Site Scripting Vulnerabilities |
|---|
| Type 2: Stored XSS (aka Persistent) |
| Type 1: Reflected XSS (aka non-Persistent) |
| Type 0: DOM-Based XSS |
| Sources: https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)<br>http://projects.webappsec.org/w/page/13246920/Cross Site Scripting |

“There’s also a third kind of XSS attacks - the ones that do not rely on sending the malicious data to the server in the first place!” Amit Klein – Discoverer of DOM-Based XSS

“DOM-based vulnerabilities occur in the content processing stage performed on the client, typically in client-side JavaScript.” – http://en.wikipedia.org/wiki.Cross-site_scripting

# XSS – Illustrated



Sent Message: Bob

Message Body: What’s for dinner? <script>alert(1)</script>

JavaScript Form

Title: Bob

Message: What’s for dinner? <script>alert(1)</script>

Submit

Blank Page - Windows Internet Explorer
about:blank
Google
Blank Page
Page
Tools
Internet | Protected Mode
100%
User Input
1: DOM-Based Reflected XSS
2: DOM-Based Stored XSS
3: Reflected XSS (via AJAX Request/Response)
4: Stored XSS (via AJAX Message Retrieval)
Store Message on Server
HTML 5 Local Storage

# XSS Terminology Confusing!!

PROBLEM: Current terms are overlapping and difficult to understand

SOLUTION: Define types of XSS based on where the dangerous sink is. This is the right approach because it's very clear and helps the developer know where the problem that needs fixing is.

- Server XSS – Untrusted data is included in generated HTML.
  - Best defense is context-sensitive output encoding.
- Client XSS – Untrusted data is added to DOM through unsafe JavaScript call.
  - Best defense is using safe JavaScript APIs.

Either can be Stored or Reflected

Untrusted data can come from client or server

# New Terminology Illustrated

Where untrusted data is used

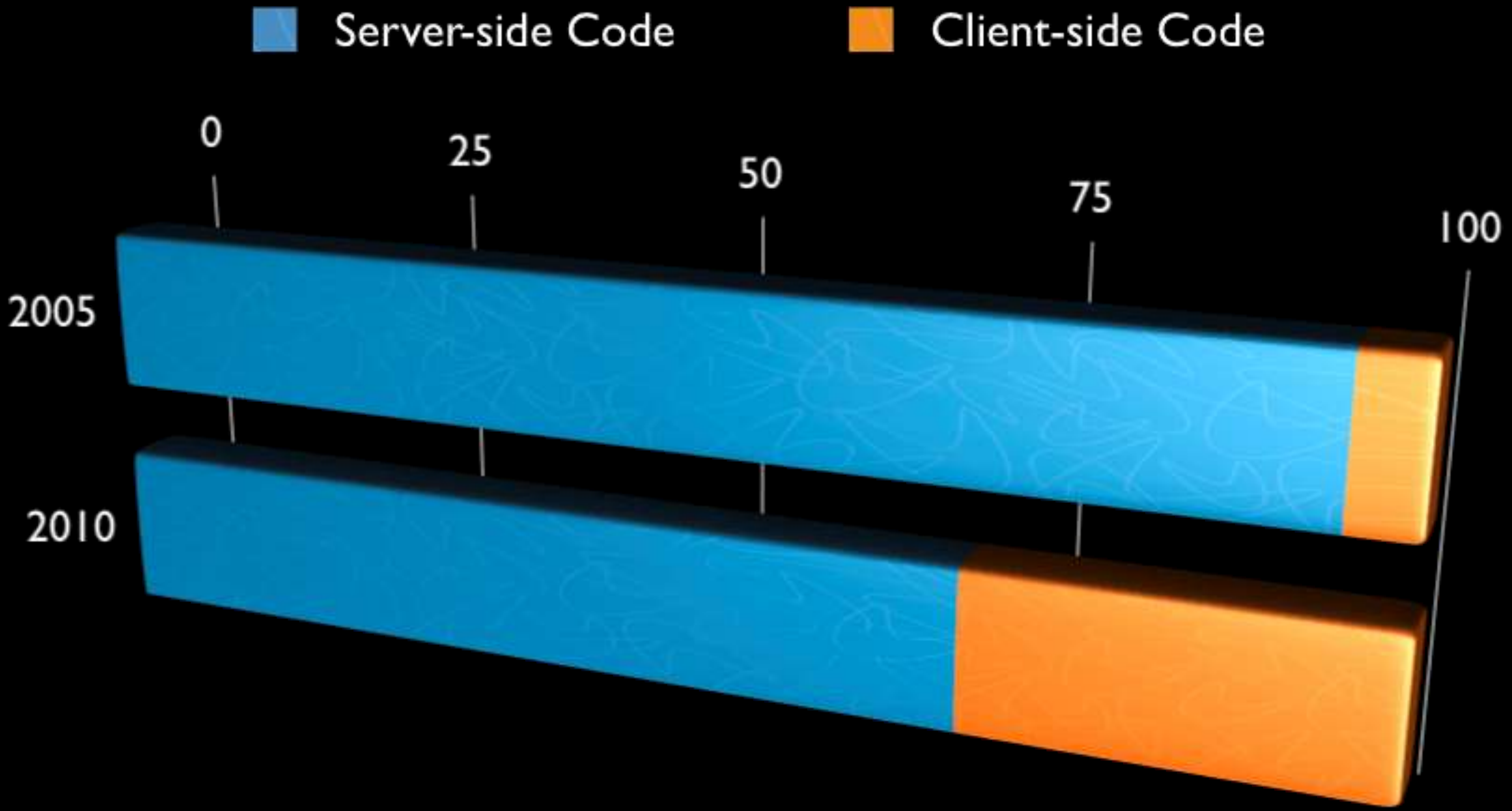
Data Persistence

| XSS | Server | Client |
|---|---|---|
| Stored | Stored<br>Server XSS | Stored<br>Client XSS |
| Reflected | Reflected<br>Server XSS | Reflected<br>Client XSS |

- ❑ **DOM-Based XSS is simply Client XSS where the data source is from the client rather than the server**
- ❑ **Stored vs. Reflected only affects the likelihood of successful attack, not nature of vulnerability or defense**

# Logic is Migrating from Server to Client…

Server-side vs. Client-side LoC in popular web applications in 2005 and in 2010



Source: IBM

# And How Vulnerable are Apps Today?

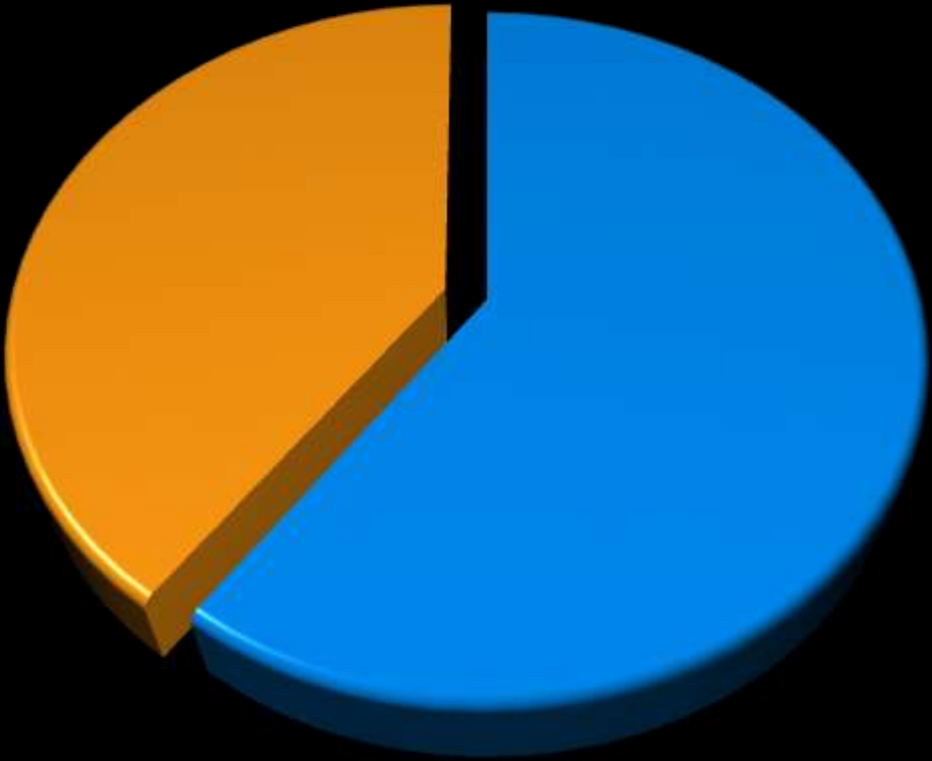
Using IBM's JavaScript Security Analyzer (JSA), IBM tested Fortune 500 + Top 178 Sites and found

40%

Vulnerable to Client-side JavaScript vulnerabilities,

90%

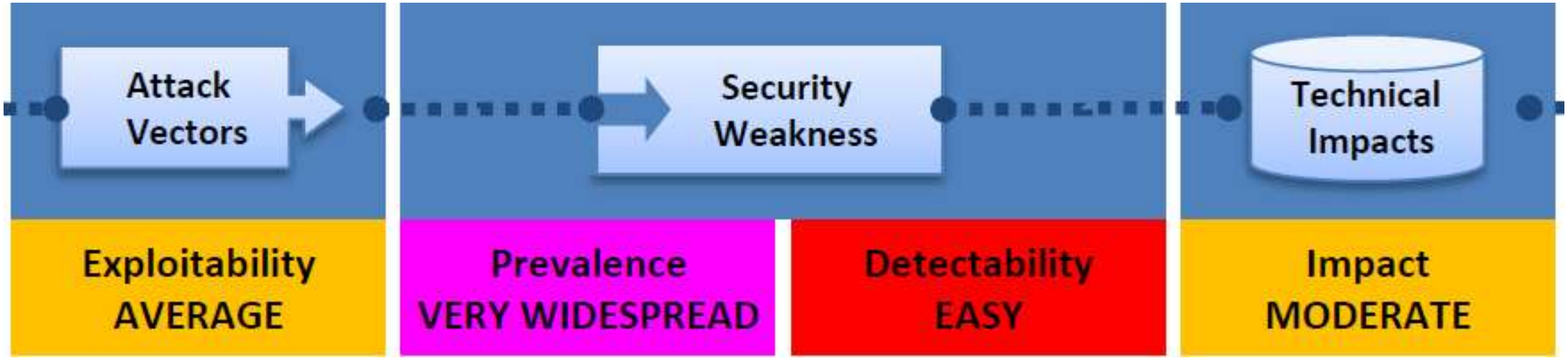
of which was caused by 3rd party JavaScript Libraries



Source: IBM

# What’s the Risk of Client/DOM-Based XSS?

- XSS Risk from OWASP Top 10



Attack Vectors → Security Weakness → Technical Impacts

| Exploitability | Prevalence | Detectability | Impact |
|---|---|---|---|
| AVERAGE | VERY WIDESPREAD | EASY | MODERATE |

  - Stored XSS attack more likely to succeed than reflected but impact is the same
- Risks are the SAME for Server and Client XSS
  - Detectability is lower for Client XSS as its harder for attackers (and defenders) to find

# DOM-Based XSS – The Classic Example

For: http://www.vulnerable.site/welcome.html#name=Joe<script>alert(1)</script>?name=Joe

```
<HTML>
<TITLE>Welcome!</TITLE>
Hi
<SCRIPT>
  var pos=document.URL.indexOf("name=")+5;

document.write(document.URL.substring(pos,document.URL
.length));
</SCRIPT>
Welcome to our system …
</HTML>
```

Notice that both data source and dangerous sink are on the client.

src: http://projects.webappsec.org/w/page/13246920/Cross Site Scripting

# Why is finding Client XSS So Hard?

- Document Object Model
  - "…convention for representing and interacting with objects in HTML, XHTML and XML documents.[1] Objects in the DOM tree **may be addressed and manipulated by using methods** on the objects." (http://en.wikipedia.org/wiki/Document_Object_Model)
  - Existing JavaScript can update the DOM and new data can also contain JavaScript
- Its like trying to find code flaws in the middle of a dynamically compiled, running, self modifying, continuously updating engine while all the gears are spinning and changing.
- **Self modifying code** has basically been banned in programming for years, and yet that's exactly what we have in the DOM.

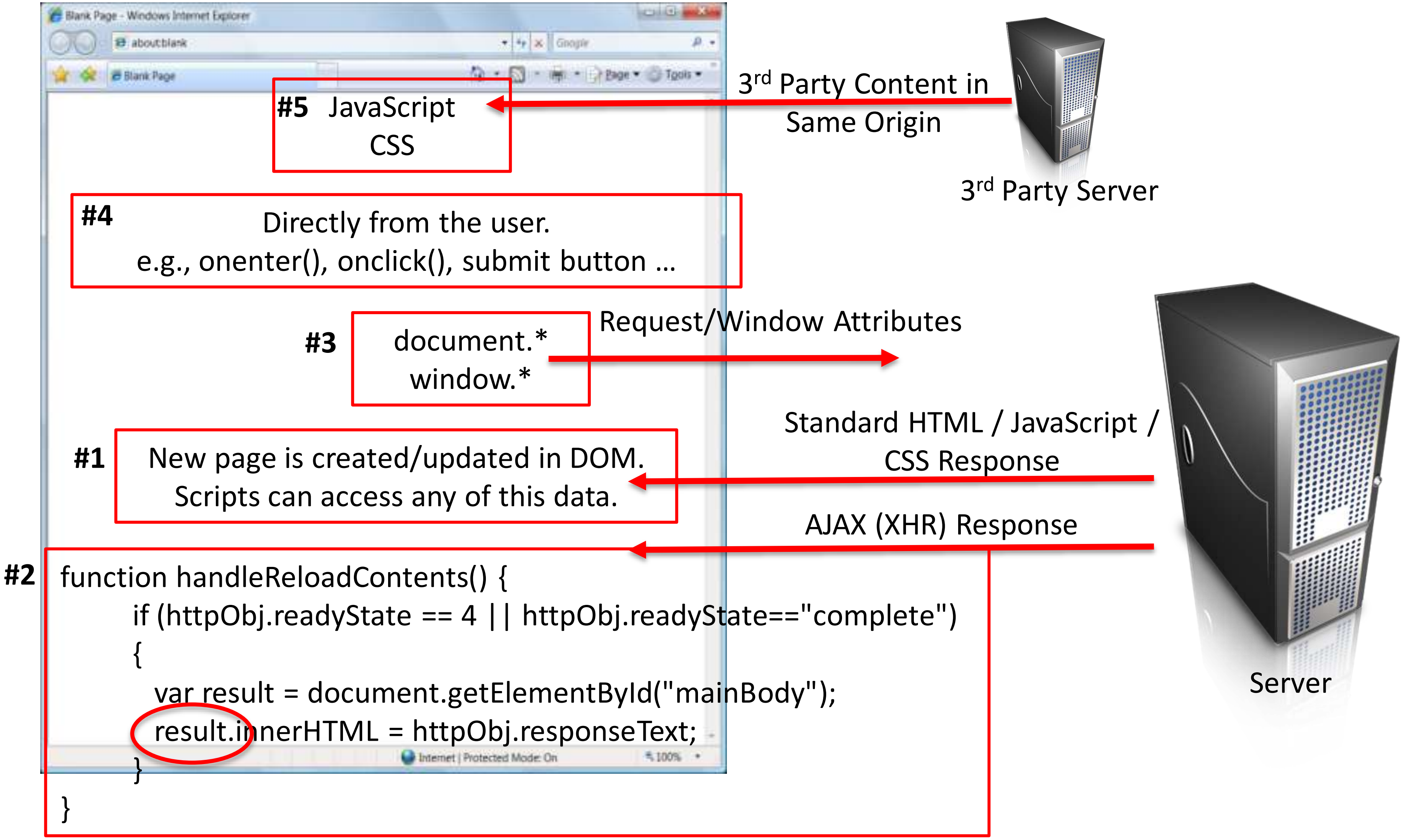
"Manual code review is hell – have you seen JavaScript lately?" Ory Segal, IBM

# How do we make detecting/avoiding Client XSS more of a Science?

## Better Understanding of

- Dangerous Sources
- Propagators (not covered here)
- Unsafe Sinks
- Defense Techniques

# Dangerous Sources (of Browser Input)



Blank Page - Windows Internet Explorer
about:blank
Google
Blank Page
Page
Tools
Internet | Protected Mode: On
100%

#5 JavaScript
CSS

3rd Party Content in
Same Origin

3rd Party Server

#4 Directly from the user.
e.g., onenter(), onclick(), submit button …

#3 document.*
window.*

Request/Window Attributes

Standard HTML / JavaScript /
CSS Response

#1 New page is created/updated in DOM.
Scripts can access any of this data.

AJAX (XHR) Response

Server

#2

```
function handleReloadContents() {
        if (httpObj.readyState == 4 || httpObj.readyState=="complete")
        {
          var result = document.getElementById("mainBody");
          result.innerHTML = httpObj.responseText;
        }
}
```

# Dangerous Request/Window Attributes

- Page: https://code.google.com/p/domxsswiki/wiki/Sources
  - For: Browsers: IE 8, Firefox 3.6.15 – 4, Chrome 6.0.472.53 beta, Opera 10.61 (Safari data TBD)
- Describes return values for document.URL / documentURI / location.*
  (https://code.google.com/p/domxsswiki/wiki/LocationSources)

**scheme://user:pass@host/path/to/page.ext/Pathinfo;semicolon?search.location=value#hash=value&hash2=value2**

**Example: http://host/path/to/page.ext/test<a"'%0A`= +%20>;test<a"'%0A`= +%20>?test<a"'%0A`= +%20>;#test<a"'%0A`= +%20>;**

**document.url output:**

http://host/path/to/page.ext/test%3Ca%22'%0A%60=%20+%20%3E;test%3Ca%22'%0A%60=%20+%20%3E?test<a"'%0A`=%20+%20>;#test<a"'%0A`=%20+%20>;

- Similar info for other direct browser data sources including
  - document.cookie (https://code.google.com/p/domxsswiki/wiki/TheCookiesSources)
  - document.referer (https://code.google.com/p/domxsswiki/wiki/TheReferrerSource)
  - window.name (https://code.google.com/p/domxsswiki/wiki/TheWindowNameSource)

# Some Dangerous JavaScript Sinks

**Direct execution**

- eval()
- window.execScript()/function()/setInterval()/setTimeout()
- script.src(), iframe.src()

**Build HTML/Javascript**

- document.write(), document.writeln()
- elem.innerHTML = danger, elem.outerHTML = danger
- elem.setAttribute(“dangerous attribute”, danger) – attributes like: href, src, onclick, onload, onblur, etc.

**Within execution context**

- onclick()
- onload()
- onblur(), etc.

Gleaned from: https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet

# Some Safe JavaScript Sinks

## Setting a value

- elem.innerText(danger)
- formfield.val(danger)
- node.textContent(danger)
- document.createTextNode(danger)

## Safe JSON parsing

- JSON.parse(danger)
  - rather than eval()

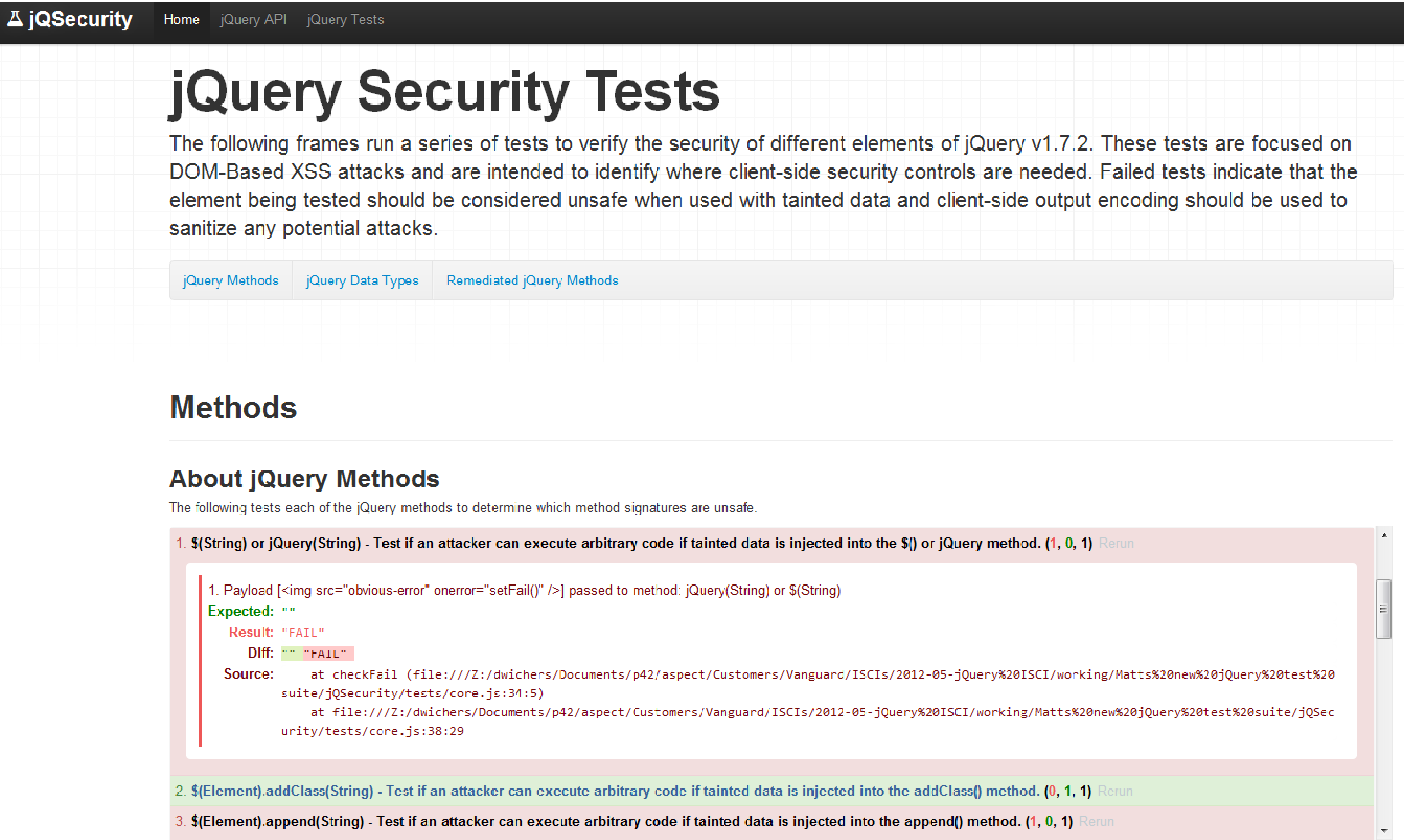
# Popular JavaScript Library #1: jQuery

**Note: This may not be the whole list!!**

| Dangerous jQuery 1.7.2 Data Types | |
|---|---|
| CSS | Some Attribute Settings |
| HTML | URL (Potential Redirect) |

| jQuery methods that directly update DOM or can execute JavaScript | |
|---|---|
| $() or jQuery() | .attr() |
| .add() | .css() |
| .after() | .html() |
| .animate() | .insertAfter() |
| .append() | .insertBefore() |
| .appendTo() | Note: .text() updates DOM, but is safe. |

| jQuery methods that accept URLs to potentially unsafe content | |
|---|---|
| jQuery.ajax() | jQuery.post() |
| jQuery.get() | load() |
| jQuery.getScript() | |



# jQuery Security Tests

The following frames run a series of tests to verify the security of different elements of jQuery v1.7.2. These tests are focused on DOM-Based XSS attacks and are intended to identify where client-side security controls are needed. Failed tests indicate that the element being tested should be considered unsafe when used with tainted data and client-side output encoding should be used to sanitize any potential attacks.

## Methods

### About jQuery Methods

The following tests each of the jQuery methods to determine which method signatures are unsafe.

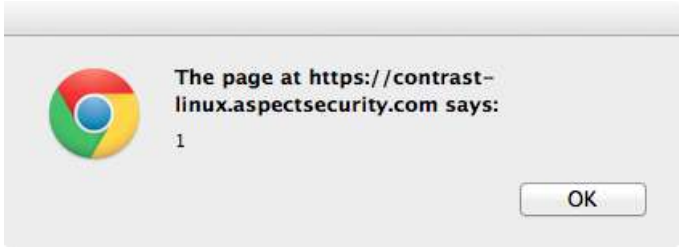
1. **$(String) or jQuery(String)** - **Test if an attacker can execute arbitrary code if tainted data is injected into the $() or jQuery method. (1, 0, 1)** Rerun

```
1. Payload [<img src="obvious-error" onerror="setFail()" />] passed to method: jQuery(String) or $(String)
Expected: ""
  Result: "FAIL"
    Diff: "" "FAIL"
  Source:     at checkFail (file:///Z:/dwichers/Documents/p42/aspect/Customers/Vanguard/ISCIs/2012-05-jQuery%20ISCI/working/Matts%20new%20jQuery%20test%20
          suite/jQSecurity/tests/core.js:34:5)
              at file:///Z:/dwichers/Documents/p42/aspect/Customers/Vanguard/ISCIs/2012-05-jQuery%20ISCI/working/Matts%20new%20jQuery%20test%20suite/jQSec
          urity/tests/core.js:38:29
```

2. **$(Element).addClass(String)** - **Test if an attacker can execute arbitrary code if tainted data is injected into the addClass() method. (0, 1, 1)** Rerun

3. **$(Element).append(String)** - **Test if an attacker can execute arbitrary code if tainted data is injected into the append() method. (1, 0, 1)** Rerun

# jQuery – But there's more…



The page at https://contrast-linux.aspectsecurity.com says:

1

- Example
  - jQuery(danger) or $(danger)
    - Immediately evaluates the input!!
    - E.g., $("<img src=x onerror=alert(1)>")
- Same safe examples
  - .text(danger), .val(danger)
- Some serious research needs to be done to identify all the safe vs. unsafe methods
  - There are about 300 methods in jQuery

We've started a list at:
http://code.google.com/p/domxsswiki/wiki/jQuery

# What about other Popular JavaScript Libraries?

Y U I
LIBRARY

mootools
a compact javascript framework

Home / Products
Ext JS 3.4

script.aculo.us

dōjō

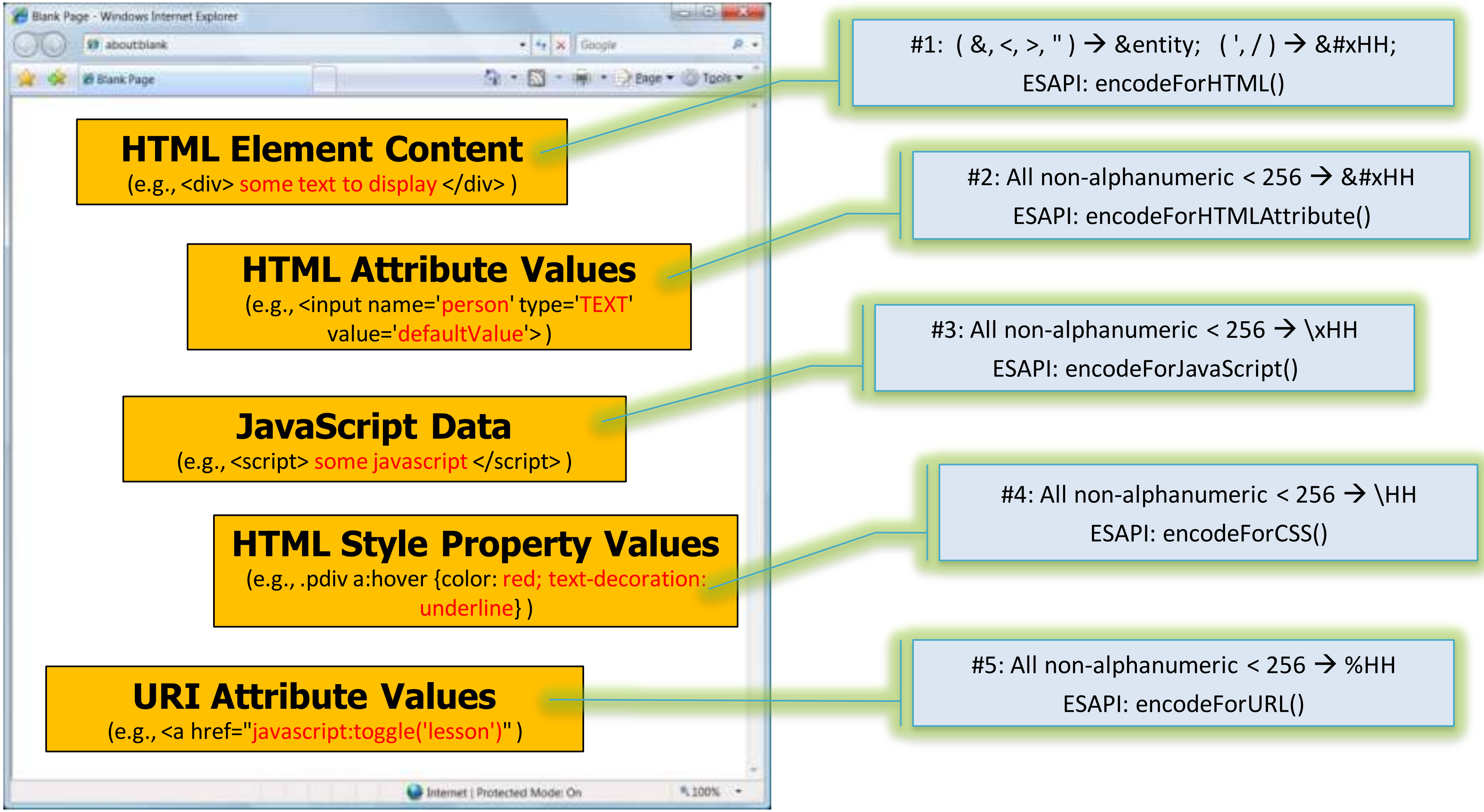
<swfobject>

prototype
JavaScript framework

# Defense Techniques
# Server XSS vs. Client XSS

| Technique | Server XSS | Client XSS |
|---|---|---|
| Avoid untrusted data | Don't include in response | Don't accept as input |
| Context Sensitive Output Escaping | Server-Side | Client-Side |
| Input Validation (Hard!) | Server-Side | Client-Side |
| Avoid JavaScript Interpreter | Can't. Page always interpreted | Avoid unsafe JavaScript / JS Library methods |

# Primary XSS Defense: Context Sensitive Output Escaping


Blank Page - Windows Internet Explorer
about:blank
Google
Blank Page
Page
Tools
Internet | Protected Mode: On
100%

**HTML Element Content**
(e.g., <div> some text to display </div> )

**HTML Attribute Values**
(e.g., <input name='person' type='TEXT' value='defaultValue'> )

**JavaScript Data**
(e.g., <script> some javascript </script> )

**HTML Style Property Values**
(e.g., .pdiv a:hover {color: red; text-decoration: underline} )

**URI Attribute Values**
(e.g., <a href="javascript:toggle('lesson')" )

#1: ( &, <, >, " ) → &entity; ( ', / ) → &#xHH;
ESAPI: encodeForHTML()

#2: All non-alphanumeric < 256 → &#xHH
ESAPI: encodeForHTMLAttribute()

#3: All non-alphanumeric < 256 → \xHH
ESAPI: encodeForJavaScript()

#4: All non-alphanumeric < 256 → \HH
ESAPI: encodeForCSS()

#5: All non-alphanumeric < 256 → %HH
ESAPI: encodeForURL()

See: www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet for more details

# Client-Side Context Sensitive Output Escaping

| Context | Escaping Scheme | Example |
|---|---|---|
| HTML Element | ( &, <, >, " ) → &entity;<br>( ', / ) → &#xHH; | $ESAPI.encoder().<br>encodeForHTML() |
| HTML Attribute | All non-alphanumeric < 256 → &#xHH | $ESAPI.encoder().<br>encodeForHTMLAttribute() |
| JavaScript | All non-alphanumeric < 256 → \xHH | $ESAPI.encoder().<br>encodeForJavaScript() |
| HTML Style | All non-alphanumeric < 256 → \HH | $ESAPI.encoder().<br>encodeForCSS() |
| URI Attribute | All non-alphanumeric < 256 → %HH | $ESAPI.encoder().<br>encodeForURL() |

ESAPI for JavaScript Library Home Page: https://www.owasp.org/index.php/ESAPI_JavaScript_Readme
Identical encoding methods also built into a jquery-encoder: https://github.com/chrisisbeef/jquery-encoder

Note: Nested contexts like HTML within JavaScript, and decoding before encoding to prevent double encoding are other issues not specifically addressed here.

# Client-Side Input Validation

- Input Validation is HARD
- We recommend output escaping instead
- But if you must, it usually looks something like this:

```
<script>
function check5DigitZip(value) {
var re5digit=/^\d{5}$/ //regex for 5 digit number
if (value.search(re5digit)==-1) //if match failed
    return false;
  else return true;
}
</script>
```

**Example inspired by:**
http://www.javascriptkit.com/javatutors/re.shtml

# Avoid JavaScript Interpreter

- This is what I recommend for Client XSS
- Trick is knowing which calls are safe or not
  - We covered some examples of safe/unsafe sinks but serious research needs to be done here

# DOM-Based XSS While Creating Form

**Attack URL Value:** http://a.com/foo?**" onblur="alert(123)**

**Vulnerable Code:**

```
var html = ['<form class="config">',
     '<fieldset>',
     '<label for="appSuite">Enter URL:</label>',
     '<input type="text" name="appSuite" id="appSuite"
          value="', options.appSuiteUrl, '" />',
     '</fieldset>',
   '</form>'];
dlg = $(html).appendTo($('body'));
         ...
```

**DOM Result:** <input type="text" name="appSuite" id="appSuite"
value="http://a.com/foo?**" onblur="alert(123)**">

# Fix #1: Input Validation

**Fix #1:**

```
regexp = /http(s):\/\/(\w+:{0,1}\w*@)?(\S+)(:[0-
9]+)?(\/|\/([\w#!:.?+=&%@!\-\/]))?/;
buttons: { 'Set': function () {
      var u = $.trim(appSuite.val());
      if (!regexp.test(u) || u.indexOf('"') >= 0) {
        Util.ErrorDlg.show('Please enter a valid
URL.');
      return;
    } ...
```

Note: This is client-side input validation. However, in this particular instance, the data was sent to the server and then back to the client. As such, this defense could be bypassed. But the next one couldn't...

# Fix #2 – Safe Form Construction

**Fix #2:**

```
var html = ['<form class="config">',
    '<fieldset>',
      '<label for="appSuite"> Enter URL:</label>',
      '<input type="text" name="appSuite"
          id="appSuite"/>',
    '</fieldset>',
  '</form>'];

dlg=$(html).appendTo($('body'));    // No user input in
HTML to be interpreted

appSuite.val(options.appSuiteUrl); // Set value safely

...
```

# Dangerous jQuery Function Example

```
namespace.events = {
  spyOn: function(selector, eName) {
    var handler = function(e) {
      data.spiedEvents[[selector, eName]]= e;
    };
    $(selector).bind(eName, handler);
    data.handlers.push(handler);
 }, …
```

Passing data as a selector to the $() function is VERY common.
Problem is that $() is also a JavaScript interpreter, not just a way of getting references to things.
Current Defense: a) Validate selector to make sure its safe, or
    b) JavaScript encode any < it may contain.
Future: Maybe I can convince jQuery to make a safe selector handler method.

# Techniques for Finding Client XSS #1

## Test like normal XSS in obvious inputs

- Step 1: Enter test script: dave<script>alert(1)</script>
- Step 2: Inspect response and DOM for ‘dave’
- Step 3: If found, determine if encoding is done (or not needed)
- Step 4: Adjust test to actually work if necessary
  - E.g., dave" /><script>alert(1)</script>
  - dave" onblur="(alert(2))

Tools: Normal manual Pen Test Tools like WebScarab/ZAP/Burp can be used here
Automated scanners can help, but many have no Client/DOM-Based XSS test features
More tips at: https://www.owasp.org/index.php/Testing_for_DOM-based_Cross_site_scripting_(OWASP-DV-003)

# Techniques for Finding Client XSS #2

## Inspect JavaScript loaded into DOM

- Step 1: look for references to user controlled input
  - Remember 5 browser sources referenced previously?
- Step 2: follow data flow to dangerous sinks
  - Lots of dangerous sinks mentioned before
- Step 3: if data is properly validated or encoded before reaching dangerous sink (its safe)
  - Validation/encoding could occur server or client side
- NOTE: THIS IS **REALLY** HARD!!

Browser Plugins REALLY USEFUL: Firebug, Chrome Developer Tools
Free Tools: DOMinator, DOM Snitch, Ra.2 try to automate this type of analysis
IBM's AppScan does this too

# Unexploitable XSS ?? Not Always …

## XSS Flaws Aren’t Always Easily Exploited

- Scenario 1: Reflected XSS protected with CSRF Token
  - Attacker workaround: Clickjacking vulnerable page
- Scenario 2: DOM-Based XSS starting with user input to form
  - Can’t force user to fill out form right? Yes – Clickjacking
  - Or, if Client XSS, but data passes through server:
    - Force the request to the server, instead of filling out the form. Works for Stored XSS, but not Reflected XSS, since XHR won’t be waiting for response.

# Its not just Client/DOM-Based XSS

## Unchecked Redirect

- window.location.href = danger, window.location.replace()

## HTML 5 Shenanigans

- Client-side SQL Injection
- ‘Pure’ DOM-Based Stored XSS (Discussed before)
- Local storage data left and data persistence (super cookies)
- Notification API Phishing, Web Storage API Poisoning, Web Worker Script URL Manipulation, (all coined by IBM)
- Web Sockets ???

## Lots more … ☹

# Free - Open Source Detection Tools

## DOMinator – by Stefano DiPaola

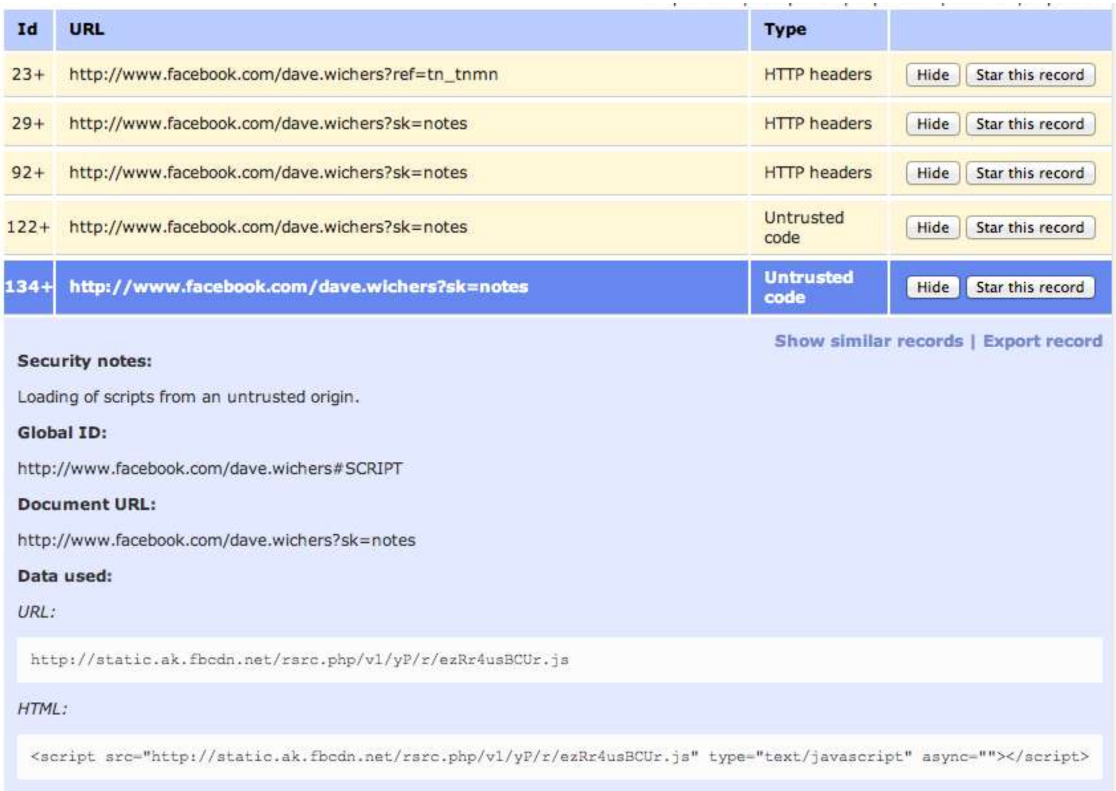
Minded security

- Firefox Plugin
- Works by adding taint propagation to strings within the browser
- Update just released
  - Adds support for some HTML5 features like cross domain requests, new tags, etc.
- http://code.google.com/p/dominator/

# Free - Open Source Detection Tools cont'd

- DOM Snitch
  - Experimental tool from Google (Dec, 2011)
    - **Real-time:** Flags DOM modifications as they happen.
    - **Easy:** Automatically flags issues with details.
    - Really Easy to Install
    - Really Easy to Use

http://code.google.com/p/domsnitch/



| Id | URL | Type | |
|---|---|---|---|
| 23+ | http://www.facebook.com/dave.wichers?ref=tn_tnmn | HTTP headers | Hide \| Star this record |
| 29+ | http://www.facebook.com/dave.wichers?sk=notes | HTTP headers | Hide \| Star this record |
| 92+ | http://www.facebook.com/dave.wichers?sk=notes | HTTP headers | Hide \| Star this record |
| 122+ | http://www.facebook.com/dave.wichers?sk=notes | Untrusted code | Hide \| Star this record |
| **134+** | **http://www.facebook.com/dave.wichers?sk=notes** | **Untrusted code** | Hide \| Star this record |

**Security notes:**

Loading of scripts from an untrusted origin.

**Global ID:**

http://www.facebook.com/dave.wichers#SCRIPT

**Document URL:**

http://www.facebook.com/dave.wichers?sk=notes

**Data used:**

*URL:*

```
http://static.ak.fbcdn.net/rsrc.php/v1/yP/r/ezRr4usBCUr.js
```

*HTML:*

```
<script src="http://static.ak.fbcdn.net/rsrc.php/v1/yP/r/ezRr4usBCUr.js" type="text/javascript" async=""></script>
```

# Free - Open Source Detection Tools cont’d

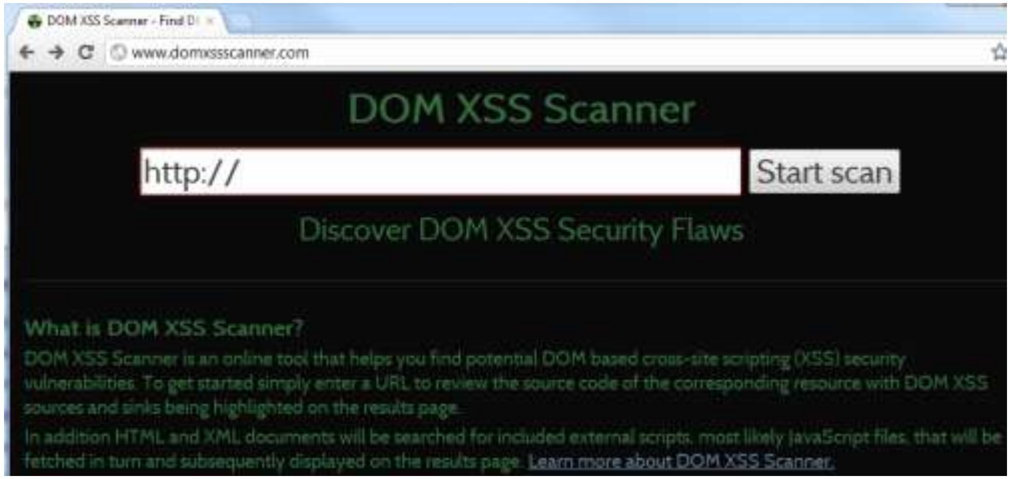
Ra.2 ra2-dom-xss-scanner

Ra.2 - Blackbox DOM XSS Scanner

- Nishant Das Patnaik/Security Engineer & Sarathi Sabyasachi Sahoo/Software Engineer, Yahoo, India
- FireFox added on, first discussed Feb, 2012
  - Downloads added to Google project Apr 5, 2012
- Large database of DOM-Based XSS injection vectors.
- Fuzzes sources with these attacks and flags sinks where the attacks actually execute.
- Intended to be mostly point and click
- http://code.google.com/p/ra2-dom-xss-scanner/

# Free - Open Source Detection Tools cont'd

- DOM XSS Scanner – from Ramiro Gómez
  - Online service
  - Just type in your URL and hit go
  - Simplistic string matching source and sink detector
  - Purely a human aide



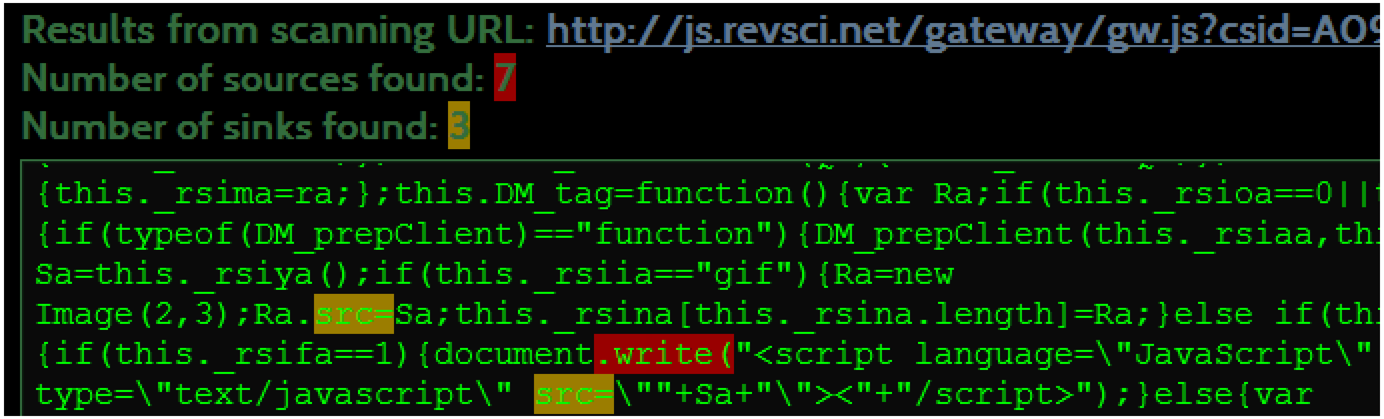
Discover DOM XSS Security Flaws

What is DOM XSS Scanner?

DOM XSS Scanner is an online tool that helps you find potential DOM based cross-site scripting (XSS) security vulnerabilities. To get started simply enter a URL to review the source code of the corresponding resource with DOM XSS sources and sinks being highlighted on the results page.

In addition HTML and XML documents will be searched for included external scripts, most likely JavaScript files, that will be fetched in turn and subsequently displayed on the results page. Learn more about DOM XSS Scanner.

http://www.domxssscanner.com/



Results from scanning URL: http://js.revsci.net/gateway/gw.js?csid=AO9

Number of sources found: 7

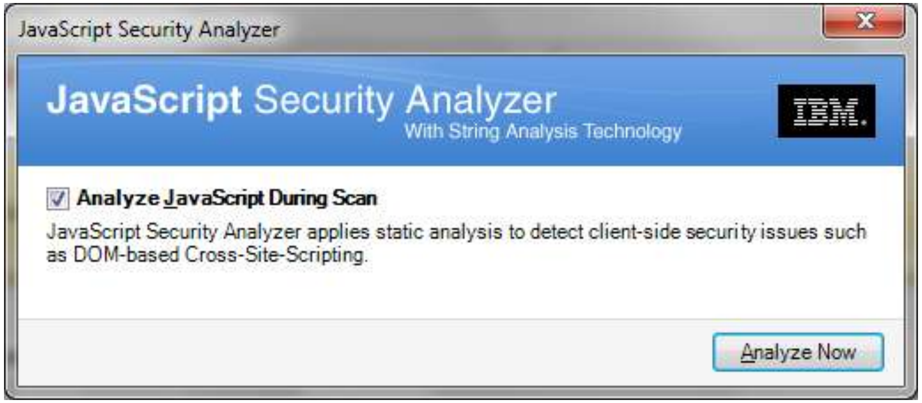
Number of sinks found: 3

```
{this._rsima=ra;};this.DM_tag=function(){var Ra;if(this._rsioa==0||
{if(typeof(DM_prepClient)=="function"){DM_prepClient(this._rsiaa,th
Sa=this._rsiya();if(this._rsiia=="gif"){Ra=new
Image(2,3);Ra.src=Sa;this._rsina[this._rsina.length]=Ra;}else if(th
{if(this._rsifa==1){document.write("<script language=\"JavaScript\"
type=\"text/javascript\" src=\""+Sa+"\"><"+"/script>");}else{var
```

# Commercial Tools

IBM's JavaScript Security Analyzer (JSA)

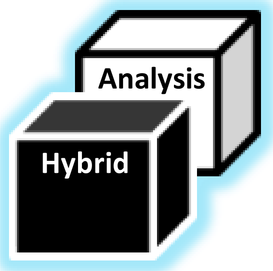
- Built into AppScan
- Crawls target site
- Copies ALL JavaScript
- Then does source code analysis on it



JavaScript Security Analyzer
JavaScript Security Analyzer
With String Analysis Technology
IBM.

☑ Analyze JavaScript During Scan

JavaScript Security Analyzer applies static analysis to detect client-side security issues such as DOM-based Cross-Site-Scripting.



JavaScript Vulnerability Types

DOM-based XSS

Phishing through Open Redirect

HTML5 Notification API Phishing

HTML5 Web Storage API Poisoning

HTML5 Client-side SQL Injection

HTML5 Client-side Stored XSS

HTML5 Web Worker Script URL Manipulation

Email Attribute Spoofing

# Commercial Tools cont'd

- acunetix Web Vulnerability Scanner (WVS)
  - has Client Script Analyzer (CSA) for detecting DOM-Based XSS

  http://www.acunetix.com/blog/web-security-zone/articles/dom-xss/
- DOMINATOR PRO Commercial Edition
- Any other commercial tools??

# Conclusion

- Client/DOM-Based XSS is becoming WAY more prevalent
- Its generally being ignored
- We need to KNOW what JavaScript APIs are safe vs. unsafe
- We need more systematic techniques for finding Client XSS flaws
- We need better guidance on how to avoid / fix such flaws