

Unraveling some of the Mysteries around DOM-based XSS

@ AppSec USA 2012 - Austin, TX

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





dave.wichers@aspectsecurity.com



Me – Dave Wichers

- COO / Cofounder Aspect Security
 - ~25 Years in Application Security Consulting
 - OWASP Board Member, Top 10 Lead, ASVS Coauthor
 - Code Review / Pen Tested 100s of Applications
 - Taught 100s of Secure Coding Courses
- Aspect Security (<u>www.aspectsecurity.com</u>)
 - Application Security Consulting
 - Application Code Review / Pen Testing
 - Secure Coding Training (Live, Virtual)
 - Secure Development Lifecycle Process Improvement



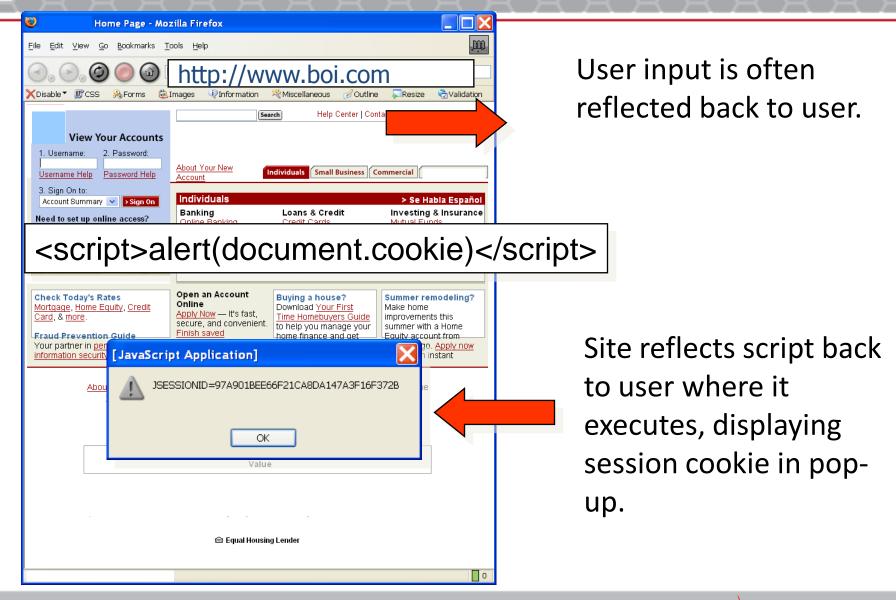
 Contrast – Vulnerability Detection through Application Monitoring for JavaEE

https://www.aspectsecurity.com/Contrast

eLearning – over 50 eLearning modules



Cross-Site Scripting (XSS) – Example





XSS Types (Current Terminology)

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)

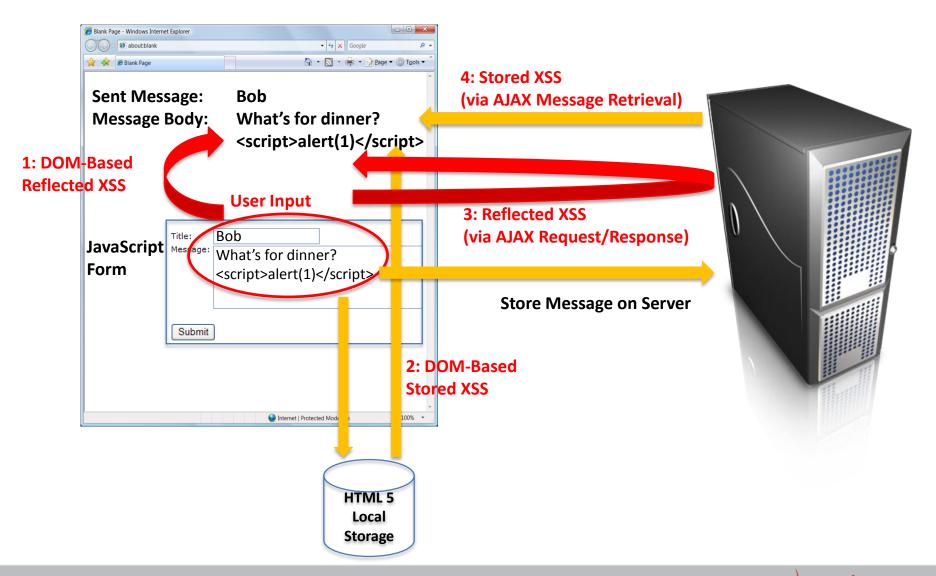
http://projects.webappsec.org/w/page/13246920/Cross Site Scripting

"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 Types – Illustrated





XSS Terminology Confusing!!

PROBLEM: Current terms are overlapping and difficult to understand

SOLUTION: Define new TERMS!!

Define types of XSS based on where the dangerous sink is (client side or server side code). This

- clarifies where in the application the problem is
- helps the developer understand how to fix it



New XSS Terms

- Server XSS Untrusted data is included in generated HTML/JavaScript
 - Best defense is context-sensitive output encoding
- Client XSS Untrusted data is added to DOM (or eval'd) 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 XSS Terminology Chart

Where untrusted data is used

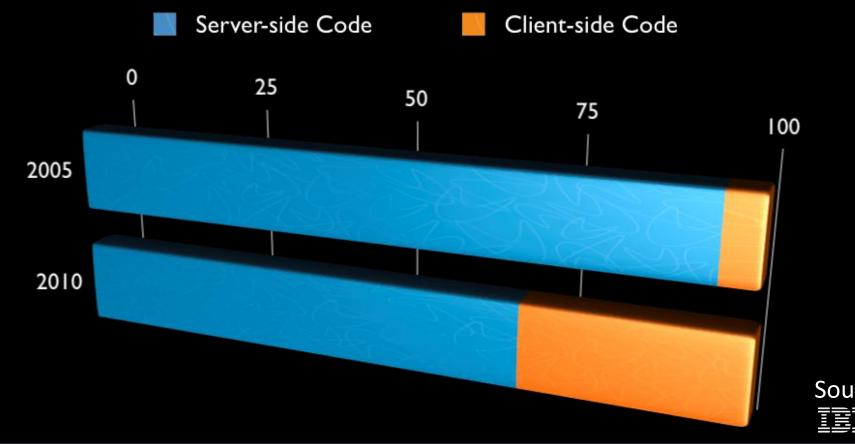
| | XSS | Server | Client |
|------------------|-----------|-------------------------|----------------------|
| Data Persistence | Stored | Stored Server XSS | Stored Client XSS |
| | Reflected | Reflected Server XSS | |

- □ DOM-Based XSS is a subset of Client XSS (where the data source is from the client only)
- ☐ 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





And How Vulnerable are Apps Today?

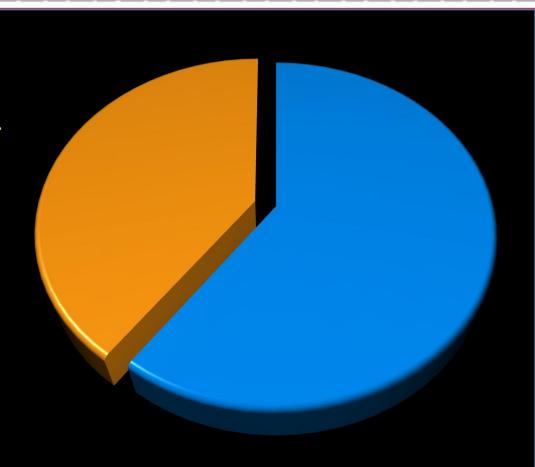
Using IBM's JavaScript Security Analyzer (JSA), IBM tested Fortune 500 + Top 178 Sites and found

40%

Vulnerable to Client-side JavaScript vulnerabilities,

90%

of which was caused by 3rd party JavaScript Libraries

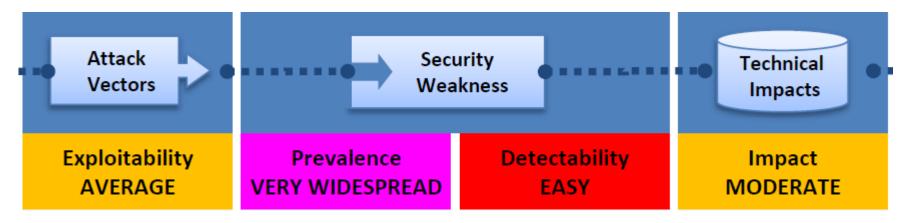






What's the Risk of Client XSS?

XSS Risk from OWASP Top 10



- 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

```
<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 <u>banned</u> 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



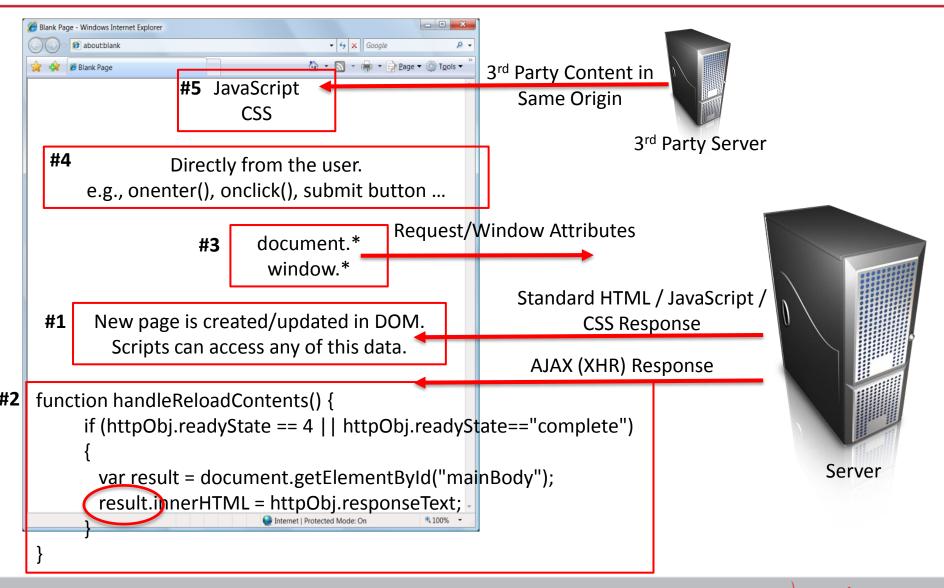
How do we make detecting/avoiding Client XSS more of a Science?

Better Understanding of

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



Dangerous Sources of Input





Dangerous Browser Provided Data

Dangerous Request/Window Attributes

Page: https://code.google.com/p/domxsswiki/wiki/Sources

Describes return values for

- document.URL / documentURI / location.* (https://code.google.com/p/domxsswiki/wiki/LocationSources)
- 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)

domxsswiki started by Stefano DiPaola – but its still incomplete



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()



Some Dangerous Methods in jQuery

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

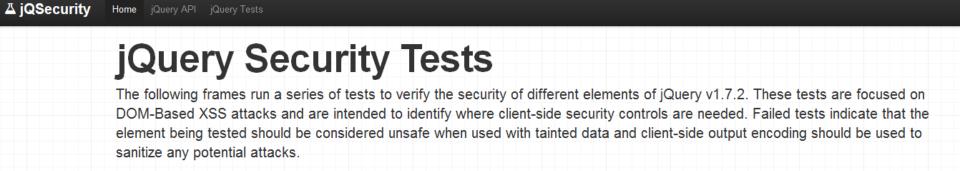
| jQuery methods that directly update DOM or can execute JavaScript | | | |
|---|---|--|--|
| \$() or jQuery() | .attr() | | |
| .add() | .css() | | |
| .after() | .html() | | |
| .animate() | .insertAfter() | | |
| .append() | .insertBefore() | | |
| .appendTo() | Note: .text() updates DOM, but is safe.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

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



Note: This is NOT a complete list. Out of 310 methods in jQuery, Aspect formally tested ~20 methods, and analyzed another 170 methods by hand.

JavaScript Library Test Harness



Methods

jQuery Methods

About jQuery Methods

¡Query Data Types

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

Remediated jQuery Methods

```
1. $\(\sqrt{\text{String}}\) or jQuery(\text{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 [<imp src="obvious-error" onerror="setFail()" />] passed to method: jQuery(\text{String}) or $\(\text{String}\)\)

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/jQSecurity/tests/core.js:38:29

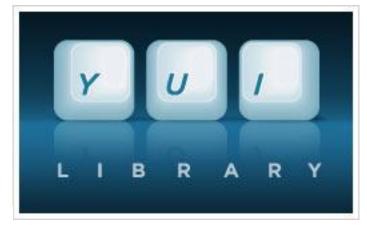
2. $\(\text{Element}\)\).addClass(\text{String}\)\) - Test if an attacker can execute arbitrary code if tainted data is injected into the addClass() method. (0, 1, 1) Rerun

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

Aspect developed this for a client engagement. We are working with two Hopkins students (Ryan Atienza & Kavya Racharla) to test all of jQuery and create an OWASP project with the test harness, test cases, and the results. Target release is Q1 – 2013



But what about all the other Popular JavaScript Libraries????





Home / Products













Another Example – Where's the flaw?

A customer app I looked at last month ...

```
public void onSuccess(Request request) {
   approvalPanel.getMessageText().setText("Some
message to the user:");
   String name < request.getApplicantName();</pre>
   approvalPanel.getNameText().setText(name);
   approvalPanel.getEmailText().setText(
   request.getApplicantEmail());
   approvalPanel.getPhoneText().setText(
   request.getApplicantPhone());
                                        Is setText() safe???
                              NO!!
                                       But it should be
        });
```

Dangerous user input

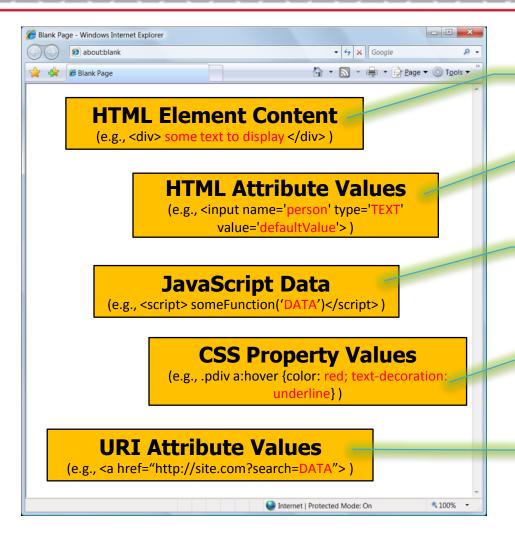
these return com.extjs.gxt.ui.client.widget.text objects

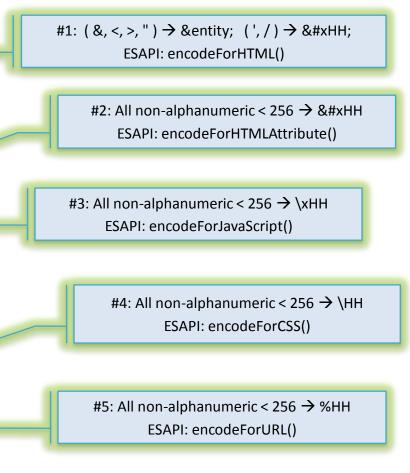


Defense Techniques Server XSS vs. Client XSS

| 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**





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 (', /) → &#xHH; | \$ESAPI.encoder(). encodeForHTML() | |
| HTML Attribute Values | All non-alphanumeric < 256 → &#xHH</td><td>\$ESAPI.encoder(). encodeForHTMLAttribute()</td></tr><tr><td>JavaScript Data</td><td>All non-alphanumeric < 256 → \xHH</td><td>\$ESAPI.encoder(). encodeForJavaScript()</td></tr><tr><td>CSS Property Values</td><td>All non-alphanumeric < 256 → \HH</td><td>\$ESAPI.encoder(). encodeForCSS()</td></tr><tr><td>URI Attribute Values</td><td>All non-alphanumeric < 256 → %HH</td><td>\$ESAPI.encoder(). encodeForURL()</td></tr></tbody></table> | | |

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/jqueryencoder

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
 - We covered some examples of safe/unsafe sinks but serious research needs to be done here



Client XSS in JavaScript 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"</pre>
          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 Technique #1: Input Validation

Fix #1:

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 Technique #2 – Avoid User Input to \$()

Fix #2:

```
var html = ['<form class="config">',
   '<fieldset>',
     '<label for="appSuite"> Enter URL:</label>',
     '<input type="text" name="appSuite"</pre>
         id="appSuite"/>',
   '</fieldset>',
 '</form>'];
dlg=$(html).appendTo($('body')); // No user input in
HTML to be interpreted
appSuite.val(options.appSuiteUrl); // Set value safely
```



iQuery based Client XSS 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.

¡Query 1.9 UPDATE: \$() only dangerous if first character is '<'



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 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 ... 😊



Dom-Based XSS Detection Tools



by Stefano DiPaola



- Free (Open Source) and Commercial Versions
- Firefox Plugin
- Works by adding taint propagation to strings within the browser
- Initial release June 2012, Update released Oct 6, 2012
 - Adds support for some HTML5 features like cross domain requests, new tags, etc.
- http://code.google.com/p/dominator/



Free - Open Source Detection Tools

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/





Free - Open Source Detection Tools cont'd



- 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



http://www.domxssscanner.com/

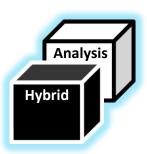
Results from scanning URL: http://js.revsci.net/gateway/gw.js?csid=AO Number of sources found: 7 Number of sinks found: 📱 {this. rsima=ra;};this.DM tag=function() {var Ra;if(this. rsioa==0| {if(typeof(DM prepClient) == "function") {DM prepClient(this. rsiaa,th Sa=this. rsiya(); if(this. rsiia=="gif") {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\" \""+Sa+"\"><"+"/script>");}else{var

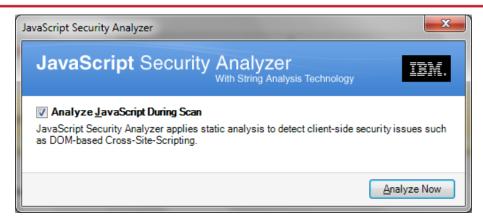


Commercial Tools

IBM's JavaScript Security Analyzer (JSA)

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





JavaScript Vulnerability Types

DOM-based XSS

Phishing through Open Redirect

HTML5 Notification API Phishing

HTML5 Web Storage API Poisoning

HTML5 Client-side SQL Injection

HTML 5 Client-side Stored XSS

HTML5 Web Worker Script URL Manipulation

Email Attribute Spoofing



Commercial Tools cont'd

Scanner (WVS)

 has Client Script Analyzer (CSA) for detecting DOM-Based XSS

http://www.acunetix.com/blog/websecurity-zone/articles/dom-xss/

Any other commercial tools??



- Client 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