Secure development of Metro style apps with HTML5

Security best practices

September 10, 2011

Abstract

This paper provides information about the secure development of Metro style apps with HTML5 for Windows® operating systems. It provides guidelines for developers to create applications that follow security best practices. It assumes that the reader is familiar with HTML5 development

This information applies to the following operating systems:

Windows® Developer Preview

Windows Server® Developer Preview

References and resources discussed here are listed at the end of this paper.

The current version of this paper is maintained on the Web at:   
 <http://go.microsoft.com/fwlink/?LinkId=228386>

**Disclaimer**: This document is provided “as-is”. Information and views expressed in this document, including URL and other Internet website references, may change without notice. Some information relates to pre-released product which may be substantially modified before it’s commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here. You bear the risk of using it.

Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

© 2011 Microsoft. All rights reserved.



Contents

[Introduction 3](#_Toc303248029)

[Secure data validation 3](#_Toc303248030)

[Sources of untrusted data 3](#_Toc303248031)

[Local context script filtering 4](#_Toc303248032)

[Validating the origin of postMessage data. 5](#_Toc303248033)

[Manual script filtering 5](#_Toc303248034)

[Defending script inputs 6](#_Toc303248035)

[Using secure parsers 7](#_Toc303248036)

[Protecting sensitive data 8](#_Toc303248037)

[Forcing HTTPS for all traffic 8](#_Toc303248038)

[Using the XDomainRequest 9](#_Toc303248039)

[Restricting navigation domains 9](#_Toc303248040)

[Use the HTML5 sandbox attribute where possible 10](#_Toc303248041)

[Conclusion 11](#_Toc303248042)

[Resources 11](#_Toc303248043)

# Introduction

Windows 8 support for the development of Metro style HTML5 apps enables developers to combine the versatility and expressiveness of web technologies alongside a wide range of powerful APIs available in the Windows Runtime. Web applications running on Windows 8 have all the power of conventional native applications and thus need to be equivalently secure. Consequently, the Windows 8 web application platform has been designed with security in mind and includes security “guardrails” which, when combined with best practices, make it easy to create a secure application. This white paper introduces practical techniques and strategies for creating secure applications

# Secure data validation

## Sources of untrusted data

The HTML5 development model in Windows 8 brings strong integration with web technologies and the vast sources of data which exist on the web today. Each of these sources of data requires secure handling by developers and should always be sanitized to ensure that malicious scripts or other dangerous content cannot cause a compromise of the application.

The need to sanitize data remains even when an application obtains data from what is perceived to be a trusted source, like a back-end web service. This need arises because data transferred in plain text may be modified by attackers in transit or may have been tampered with if the back-end service itself was compromised. To ensure the security of the application and to protect its users, all data that the application receives from the network should be validated.

The first step in validation is to identify points where data flows into the application. The Windows 8 application model includes several new and existing sources of data such as these common examples:

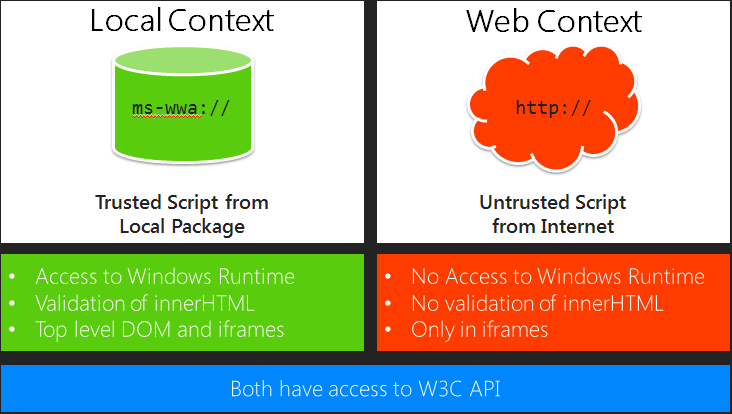
* HTML DOM APIs:
  + XMLHttpRequest content
* HTML5 APIs
  + PostMessage content
  + WebSocket content
* Windows Runtime APIs
  + SMS Messages
  + RSS and Atom content

This is just a partial list of untrusted data sources. Taking inventory of the data that originates from the network and sanitizing that data is the first step in securing your application.

## Local context script filtering

The Metro style app infrastructure was designed with security in mind and helps developers to avoid common security issues that can result from the unsafe handling of untrusted data. One of the methods introduced in the Metro style app infrastructure to improve security is support for local and web contexts. These contexts enable developers to design their applications from the ground up to segregate untrusted content such as remote JavaScript from trusted code and data included within an application package.

The web and local contexts are separated via standard frame isolation, and their contexts are determined automatically based on the origin of the content. For example, content referred via the ms-wwa:// protocol will be automatically loaded into a local context frame, while remote web content loaded via a tag such as <iframe src=http://www.contoso.com”> will always be loaded with a web context. The two contexts have different security properties and access levels, which are outlined in detail in the diagram below. While the two contexts have different access levels, convenient intra-frame communication is available via the postMessage infrastructure.



The automatic script filtering of the DOM methods will drop any dynamic content that is deemed unsafe and prevent it from being injected into the DOM, while allowing benign markup to remain intact.

While automatic script filtering is a great “guardrail,” which can prevent unexpected security attacks from harming the user, it should not be an application’s only defense. When an automatic script filtering event occurs, an error is thrown that can affect the user experience. In addition, third-party libraries that operate on data may use APIs that are not automatically filtered, which can lead to security issues. Thus it is a security best practice to avoid relying on automatic script filtering as a universal defense and instead standardize on pro-active filtering within your code. Performing explicit filtering of untrusted data is the best way to ensure that your app remains safe.

## Validating the origin of postMessage data.

The [postMessage](http://msdn.microsoft.com/en-us/library/cc197015(v=vs.85).aspx) API is one of the primary methods for transferring data between the local and web contexts in a Windows 8 HTML5 application. Developers are encouraged to use postMessage to enable secure “mash-up” designs which previously would have called for directly including a remote script unsafely via “<script src=’….’/>” or through cross-domain hacks. Using frame-based separation between local and remote sources enables developers to include remote content such as maps or advertisements while the document and resource access remains isolated through same-origin policy.

To support safely using postMessage to pass messages between local and remote documents, developers should check the source origin of a postMessage response before operating on the data. This is necessary because many applications will include multiple remote iframes within a single application, each with a different origin and trust level. By default, a single message handler function is set up for each page that is executed when an ‘onmessage’ event is triggered, and any frame can send a message.

Insecure message handler, does not implement origin check

addEventListener(document, 'onmessage', function (e) {

div.innerHTML = window.toStaticHTML(e.data);

});

//Secure message handler, validates message domain origin

addEventListener(document, 'onmessage', function (e) {

if (e.origin == 'data.contoso.com') {

div.innerHTML = window.toStaticHTML(e.data);

}

});

Figure Validating postMessage Origin

In most cases, an application will want to accept messages from one or two iframes that send map coordinates or other legitimate mash-up content, while rejecting other message content from untrusted entities such as advertisements or comment streams. Filtering messages by origin is a great way to reduce the attack surface of a web application and ensure that untrusted data is rejected. Origin checking should be used everywhere that an ‘onmessage’ handler exists.

## Manual script filtering

Script filtering in the HTML5 Metro style app environment is simple to implement and should be performed on all untrusted data sources. The specific technique for validating data is dependent on the intended usage for the content. For content that is intended to be displayed as simple static data, it is preferable to use DOM APIs that ignore dynamic elements to add the content to the DOM. This will enable the content to be displayed safely because any script or dynamic elements in the data will be displayed as simple text instead of interpreted as code, which could lead to security issues. Methods such as createTextNode can be used to populate an element with the untrusted data, which can then be added to the document DOM with appendChild or importNode. If the content is intended to be added to an existing DOM element, the innerText or outerText properties can be used safely.

//Avoid adding untrusted dynamic content

//Unsafe method 1

var myDiv = document.createElement("div");

myDiv.innerHTML = xhr.responseText

document.body.appendChild(myDiv);

//Unsafe method 2

document.writeln(xhr.responseText);

Figure Unsafe methods for adding dynamic methods

//Forcing untrusted content into static text is safe

//method 1

var myDiv = document.createElement("div");

myDiv.innerText = xhr.responseText

document.body.appendChild(myDiv);

//method 2

var myData = document.createTextNode(xhr.responseText);

document.body.appendChild(myData);

//method3

var oDiv = document.getElementById("div1");

oDiv.outerText = xhr.responseText;

Figure Safe methods for adding dynamic methods

In scenarios where the untrusted data is intended to contain markup, the DOM method window.toStaticHTML can be used to drop all unsafe mark-up while keeping safe data intact.

//the untrusted data contains unsafe dynamic content

var unTrustedData = "<img src='http://www.contoso.com/logo.jpg' on-click='calltoUnsafeCode();'/>";

//safe dynamic content can be added to the DOM without introducing errors

var safeData = window.toStaticHTML(unTrustedData);

//the content of the data is now "<img src='http://www.contoso.com/logo.jpg'/>" and is safe to add because of filtering

document.write(safeData);

Figure Method for safe insertion of dynamic content

## Defending script inputs

Within the application infrastructure, automatic script filtering operates on DOM APIs that allow dynamic markup injection while leaving explicit execution inputs such as [eval](http://msdn.microsoft.com/en-us/library/12k71sw7(v=vs.94).aspx) untouched. This allows developers the freedom to intentionally evaluate script while preventing accidental introduction of security issues. Based on this distinction, the following script execution methods should be used with great care in development of Metro style HTML5 apps and should never be used directly on untrusted or unsanitized web data.

* eval
* execScript
* setTimeout
* setInterval
* function
* msWWA.execUnsafeLocalFunction

These methods are not filtered with Windows HTML5 applications methods and therefore should never be used on untrusted data without filtering or encoding first. The msWWA.execUnsafeLocalFunction DOM method was introduced in Windows 8 and is designed to temporarily eliminate the automatic script filtering in web application “local context” introduced earlier. This is helpful when a developer needs to leverage a markup-generating template library such as JQuery for a defined period of time that would otherwise be impeded by automatic filtering. Similar to the security repercussions of eval, using msWWA.execUnsafeLocalFunction on an arbitrary method that processes foreign data sources can lead to security issues. Security problems are possible because the template libraries may internally write the untrusted data to the document. When in doubt about the origins of any data, a [safe encoding library](http://wpl.codeplex.com/) or the [toStaticHTML](http://msdn.microsoft.com/en-us/library/cc848922%28v=vs.85%29.aspx) method should be used to ensure that data is sanitized prior to use with unsafe methods.

## Using secure parsers

In many cases, developers are familiar with the possible dangers of accepting untrusted input to explicit script execution sinks like those listed above. However, there are a few common development idioms based on script evaluation that can lead to security issues when used in the HTML5 Metro style apps:

<!DOCTYPE html>

<html>

<head>

</head>

<body>

<div id="foo"></div>

<script type="text/javascript">

var xhr = new XMLHttpRequest();

xhr.open("GET", "http://contoso.com/json.js", false);

xhr.onreadystatechange = function () {

if (xhr.readyState == 4) {

if (xhr.status == 200) {

//DO NOT USE, UNSAFE

var myObject = eval('(' + xhr.responseTxt + ')');

}

}

};

xhr.send();

</script>

</body>

</html>

Figure Example of Dangerous eval usage

Using the JavaScript eval method to generate a local JavaScript object from a server side [JSON](http://msdn.microsoft.com/en-us/library/bb299886.aspx) response is a common but dangerous web development practice. Evaluating a JSON string literal to generate a JavaScript object works because JSON is a proper subset of JavaScript and the eval entry point will correctly parse the text and produce an object structure. However this can lead to security issues because the eval method can both parse JSON and also interpret valid JavaScript statements that are surreptitiously supplied to it.

A security issue can occur if an attacker can use a Man-in-the-Middle method to tamper with the data supplied to eval. Using the JSON.Parse DOM method is the secure alternative for converting JSON text to an object. This method rejects any non-JSON script that could introduce an attack.

<!DOCTYPE html>

<html>

<head>

</head>

<body>

<div id="foo"></div>

<script type="text/javascript">

var xhr = new XMLHttpRequest();

xhr.open("GET", "http://remote-server.com/json.js", false);

xhr.onreadystatechange = function () {

if (xhr.readyState == 4) {

if (xhr.status == 200) {

var myObject = JSON.parse(xhr.responseText);

}

}

};

xhr.send();

</script>

</body>

</html>

</body>

</html>

Figure Using JSON.Parse for safe parsing

Leveraging the JSON.Parse API will ensure that JSON data is safely parsed without the threat of untrusted script evaluation.

# Protecting sensitive data

Windows 8 HTML5 applications should be designed to handle sensitive data and other important operations as robustly as their traditional counterparts. Using SSL whenever sensitive data is sent back and forth to remote servers is paramount for maintaining safe custody of user data. Security attack tools are widely available to bad guys, and developers should assume that parties with malicious intent will inevitably seek to attack any application that is not using SSL to intercept private data.

In addition to privacy protection, SSL provides benefits for the integrity of applications because it keeps attackers from using Man-in-the-Middle techniques to tamper with JSON or other network data streams. Another situation where protecting the integrity of traffic is critical is where network data is being supplied to the Windows Runtime, which leverages the capabilities requested by the app.

## Forcing HTTPS for all traffic

The Windows HTML5 application environment makes it simple to force all remote content to be transmitted and received via HTTPS through a simple HTML meta-tag:

<meta name="ms-https-connections-only" value="true"/>

Figure HTTPS-only Meta-tag

Adding this meta-tag will cause any non-HTTPS navigations to throw an error and protect users of the application from any connection tampering such as DNS cache poisoning or ARP cache spoofing. The errors that are thrown by non-SSL connections are easy to debug through a custom error message in Visual Studio. For application developers who are connecting to servers that do not have an existing PKI infrastructure, Windows 8 HTML5 applications support private certificate bundling with the application package.

## Using the XDomainRequest

Unlike conventional browsers, local script and markup in the HTML5 applications on Windows 8 have unrestricted cross-domain access to web services using an XmlHttpRequest. The XHR by default will make cookie-authenticated requests to a given site when grabbing web service or JSON data. The usage of cookies with cross-domain requests can have two potential effects on the security of an app:

1. If the app is compromised, attackers may gain access to the cookies for all sites authenticated to the local context.
2. Any non-SSL XmlHttpRequest traffic may be intercepted.

xdr = new XDomainRequest();

if (xdr) {

xdr.onerror = err;

xdr.ontimeout = timeo;

xdr.onprogress = progres;

xdr.onload = loadd;

xdr.timeout = tbTO.value;

xdr.open("get", tbURL.value);

xdr.send();

}

Figure XDomainRequest Usage

Because of these concerns, the [XDomainRequest](http://msdn.microsoft.com/en-us/library/cc288060(v=vs.85).aspx) object should be used only when cookie-authenticated XHR access is *not required*. If authenticated access is required, then the XmlHttpRequest object over SSL is preferred. The XDomainRequest allows for cross-origin traffic similar to an XHR but without cookies or referrer headers. This increases both the security and privacy of cross-domain requests, leading to an all-around safer app.

## Restricting navigation domains

The navigation domain of a Metro style app developed with HTML5 is set within the app manifest inside the application package. The enumeration of the domain set within the manifest serves as a whitelist of possible navigation destinations for the top-level window of a Windows HTML5 application. Restricting the list of possible navigation domains is important to avoid the possibility that attackers may redirect a user’s application session to a domain that they can control and then perform phishing attacks or other activities which are harmful to users.

A simple best practice is to avoid using wildcards where possible, because many sub-domains of large web services provide the ability to host arbitrary pages or other content that an attacker could take advantage of to redirect the Metro style app developed with HTML5 and perform phishing attacks on users.

<!— Overly permissive domain -->

<ApplicationContentUris>

<Rule Type=”include” Match=”http://\*.example.com/”/>

</ApplicationContentUris>

<!— Safer approach -->

<ApplicationContentUris>

<Rule Type=”include” Match=”http://mycontent.example.com/”/>

<Rule Type=”include” Match=”http://myimages.example.com/”/>

</ApplicationContentUris>

Figure Navigation Domain Examples

## Use the HTML5 sandbox attribute where possible

The HTML5 Iframe ‘sandbox’ attribute introduces additional restrictions on the content stored within an iframe. Although the standard Windows 8 web application environment has additional security protections based on content origin, the HTML5 sandbox attribute can be used in tandem as an additional layer of defense-in-depth to help developers restrict untrusted content within a page, such as advertisements or other arbitrary content. Using the sandbox attribute, a developer can choose to restrict the capabilities of content in an iframe to the minimum privileges required. Some of the capabilities that can be used to achieve fine-grained control of the sandbox attribute are:

* Removing all access to DOM of the parent page
* Removing the ability to execute scripts
* Removing embedded forms
* Restricting the ability to read or write cookies, local storage, or local SQL databases

<!DOCTYPE html/>

<html>

<head>

<title>Sandboxes are easy to use!</title>

</head>

<body>

<iframe sandbox src="http://contoso.com/untrusted\_ad\_content.html"></iframe>

</body>

</html>

Figure Use of HTML 5 Sandbox attribute

Using the sandbox attribute to remove scripting or other capabilities from web content that don’t require this type of access can greatly increase the security posture of a Windows HTML5 application.

# Conclusion

Today more than ever, customers are aware of the security reputations of the applications they use. Customers clearly care about the security of private data they entrust to apps. Given the choice, customers will often opt to use an app with a strong security reputation. The Windows HTML5 Metro style app platform was created with security in mind, and many of the most important security features operate without any extra code or effort from developers. Using the simple tips and best practices outlined in this article to complement the platform’s security “guardrails” will ensure that your application is at top of the list when it comes to security.

# Resources

Inspect Your Gadget

http://msdn.microsoft.com/en-us/library/bb498012.aspx

Anti-Cross Site Scripting Library

http://msdn.microsoft.com/en-us/security/aa973814.aspx

Improving Web Application Security: Threats and Countermeasures

http://msdn.microsoft.com/library/aa302420.aspx

postMessage Method

http://msdn.microsoft.com/en-us/library/cc197015(VS.85).aspx