

Silverlight Security Overview

Nick Kramer, Microsoft Corporation – April 2010

***Abstract***: Security on the Web continues to be a significant concern to consumers and enterprises alike. Security becomes increasingly important as we see the migration of more and more everyday activities onto the Web is driving the explosive growth in applications built on Web development platforms such as Microsoft Silverlight.

In this environment really secure applications are a result of both protection built into development platforms and adoption of secure practices by developers.

This document describes how Silverlight protects end-users from attack by malicious web sites, and how to build a secure Silverlight application.

# Contents

[1. Introduction 3](#_Toc260240046)

[2. How Silverlight protects end-users from malicious web sites 4](#_Toc260240047)

[2.1. Sandbox features and restrictions 4](#_Toc260240050)

[2.2. Networking 5](#_Toc260240051)

[2.3. Out of browser applications 6](#_Toc260240052)

[2.4. Installing & updating Silverlight 7](#_Toc260240053)

[2.5. Implementation of the sandbox 8](#_Toc260240054)

[3. Creating secure web sites using Silverlight 9](#_Toc260240055)

[4.1. Cross site scripting (XSS) 9](#_Toc260240058)

[4.2. Isolating Silverlight from HTML/JavaScript 9](#_Toc260240059)

[4.3. Isolating Silverlight code from other Silverlight code 10](#_Toc260240060)

[4.4. Sending data between Silverlight apps 10](#_Toc260240061)

[4.5. Preventing unauthorized reuse of your .xap 10](#_Toc260240062)

[4.6. Validating server input 11](#_Toc260240063)

[4.7. Protecting information inside a .xap 11](#_Toc260240064)

[4.8. Protecting data in isolated storage 12](#_Toc260240065)

[4.9. Special considerations for Trusted Apps 12](#_Toc260240066)

[4.10. Security APIs 13](#_Toc260240067)

[4. Conclusion 14](#_Toc260240068)

[5. Additional resources 14](#_Toc260240069)

# Introduction

Silverlight is a powerful development platform for creating engaging, interactive user experiences for Web, desktop, and mobile applications when online or offline.Silverlight is a free plug-in, powered by the .NET framework and compatible with multiple browsers, devices and operating systems, bringing a new level of interactivity wherever the Web works.

In the web browser, Silverlight applications are contained within a larger HTML page, and the HTML contains an <object> tag that points to the Silverlight app. Silverlight applications are packaged into .xap files (pronounced "zap"), which are ZIP files that contain managed code .DLLs and XAML files (pronounced "zammel”). Xaml is the XML format for UI & vector graphics.

Silverlight applications can then run in three different security modes:

* **In browser** -- the application runs as part of a larger webpage, and is subject to the same sandboxed protection as other web content.
* **Out of browser - sandboxed** -- the application can run in browser, but it can also be installed onto the user's Start menu; Silverlight will prompt the user before installing. Sandboxed out of browser applications have the same security restrictions as ‘in browser’ applications.
* **Out of browser - trusted applications** -- like sandboxed out of browser, these applications could be run in browser with ‘in browser’ security restrictions. The difference for trusted applications is when the application is installed and run out of browser, the application has additional privileges, subject to user granting permission and group policy settings in a corporate environment. From a security perspective, running this class of Silverlight application is similar to running a .exe -- it has access to a subset of the file system (‘My Documents”) and calling out to COM objects.

# How Silverlight protects end-users from malicious web sites

Silverlight follows basic Web security principles while providing additional safe functionality for webpages to use. Silverlight has to assume that all webpages/in-browser apps are potentially malicious, and runs applications in a sandbox -- allowing those applications only to do things which are provably safe. Silverlight extends the sandbox of the current generation of browsers -- there is additional functionality that couldn't be done without a browser plug in like Silverlight -- but those extensions are all safe for use by untrusted webpages.

Two browser principles in particular deserve a brief explanation:

**Requires user initiated** -- we say a feature "requires user initiated" when that feature may only be called by an application in response to user input, ie during a keydown/KeyUp/Mouse\*Down/Mouse\*Up event. User initiated helps convey a sense of which app is asking, which is important since many Silverlight dialogs (eg OpenFileDialog) don't say which application is asking, and users frequently don't read that all that carefully anyway. User initiated also limits how frequently an attacker can eg pop up annoying dialogs.

**Same origin policy –** As in HTML, whentwo files (e.g. xaps or html pages) are downloaded from the same domain name, they are considered to have the same origin. Similar to browser iframes, Silverlight imposes additional restrictions when communicating between xaps and pages that do not have the same origin. Same origin considers the entire domain name – eg, <http://downloads.microsoft.com> is separate from http://mail.microsoft.com.

## Sandbox features and restrictions

**OpenFileDialog/SaveFileDialog** -- Silverlight allows apps to read and write files on the hard drive, but only after the user selects them in a dialog box. The application may not suggest a default file name/location. User initiated is required to help the user understand which webpage is asking to open/save a file, as well as to mitigate malicious apps going into an infinite loop of popping up dialogs. For saved files, Silverlight will give the file the "mark of the web" to indicate to subsequent readers that the file came from the Internet zone not the local machine zone. Unlike the browser "upload" button, the Silverlight app gets access to the file as soon as the user clicks "ok" on the dialog -- no need to send it to a server.

**Isolated storage** -- Silverlight allows a webpage to save data to a special directory on the local hard drive. Webpages from a given domain share an isolated storage, which cannot be read or written by pages from other domains. By default, each domain is limited to 1 MB of storage, but applications may prompt the user to expand this quota up to 100 MB. (The API to show the prompt requires user initiated) Like cookies, the user may clear isolated storage. To avoid pestering the user, the dialog may only be shown once per page.

**Full screen mode** -- Silverlight applications may enter full screen mode, whereby they take over the entire screen -- e.g., no browser status bar at the bottom. Entering fullscreen mode requires user initiated. Upon entering fullscreen, Silverlight displays “Press ESC to exit full screen mode." along with the name of the website for a period of time. Upon pressing escape or switching to another window, Silverlight will exit full screen mode; the application may not prevent the exiting the fullscreen mode. In order to prevent full-screen applications from spoofing an operating system password dialog, fullscreen apps do not receive alphanumeric keystrokes.

**Webcam/microphone** -- Silverlight applications can access the webcam & microphone after prompting the user for permission; once granted, permission lasts until the page is closed/navigated away from. Launching the permission prompt requires user initiated. Administrators may prevent all websites from accessing the webcam/microphone (& showing the prompt) by setting a regkey.

**Printing** -- Silverlight applications may print after showing a print dialog. Like other prompts in Silverlight, this requires user initiated.

**Clipboard** -- After showing a security prompt, Silverlight apps may read and write plain text from the clipboard. Launching the prompt requires user initiated; once granted, permission lasts until the pages closed/navigated away from. Accessing the clipboard also requires user initiation, in order to prevent a malicious site from monitoring clipboard data intended for other applications (read) and prevent spamming the clipboard (write).

**Drag and drop target** -- If the Silverlight application is so configured, users can drag and drop files onto the Silverlight application, upon which the app can read the contents of the file as if it had been selected in a OpenFileDialog. Silverlight does not support dragging files out of Silverlight onto non-Silverlight.

**Right-click** -- if the application does not handle right-click, Silverlight will show a small context menu providing access to Silverlight settings (such as isolated storage quotas). If the application chooses to handle right-click, this context menu will not be shown, instead the user may access Silverlight settings via the Start menu. (Or equivalent on Mac)

## Networking

In the presence of firewalls, the Silverlight computer may be able to talk to machines (known as third-party servers) that the site of origin web server couldn't talk to directly. To prevent unauthorized access, Silverlight requires that the third-party server have a cross-domain policy file granting access to the domain the .xap file came from.

For http/https requests, Silverlight supports two different kinds of cross domain policy files:

* **clientaccesspolicy.xml** -- This file lists which domains of origin are allowed to talk to the third-party, with support for wildcards like \*.microsoft.com. clientaccesspolicy.xml may also restrict access to specific paths on the server -- e.g. http://thirdparty.com/public but not http://thirdparty.com/private.
* **crossdomain.xml** (For Adobe Flash compatibility) -- must be served from the root of the domain granting access (e.g. http://thirdparty.com/crossdomain.xml), and must grant access to all domains: <allow-http-request-headers-from domain="\*" headers="\*"/>

Silverlight supports two different modes for HTTP networking -- **BrowserHttpWebRequest** and **ClientHttpWebRequest** (shortened to BHWR/CHWR or browser stack/client stack). The browser stack is fully integrated with the browser, e.g. using the browser's cookie store, authentication, and network settings such as proxy server. However, due to browser limitations, the browser stack can't use the full set of HTTP verbs (it's limited to GET and POST), nor does it give much control over cookies or headers. The client stack enables additional functionality by using the underlying operating system's networking stack. As a result, the client stack will use the operating system's network settings rather than the browser's settings (for Internet Explorer/Windows and Safari/Mac, browser and operating system settings are one in the same). The client stack does not use the browser cookie store, in fact it does not persist those cookies at all, storing them in memory. (An app could manually persist its cookies using isolated storage, except for HTTPONLY cookies which can't be read by the app)

Like <img> in html, the Silverlight **<Image>** and **<Media>** elements are able to download image and media files from servers without cross domain policy files. To prevent information disclosure back to the site of origin, the app may not see the content of these image/media files, or their constituent pixels, or even tell the difference between a corrupt file versus a file that doesn't exist.

In addition to HTTP, Silverlight also allows apps to use **sockets**. Using sockets requires a clientaccesspolicy.xml served over port 943 that grants access to the specific port. To reduce the potential for conflict with other services, sockets are restricted to ports 4502-4534. Silverlight only supports outgoing sockets not listening sockets. Silverlight 3 supports TCP sockets

Silverlight 4 has support for UDP multicast sockets, both single source multicast (SSM) and any source multicast (ASM). In both cases, Silverlight sends a policy check message to the source (or for ASM, all sources) and waits for a response before allowing the application to access the socket. Silverlight cannot create a multicast group, only join existing groups.

For more details about cross domain policy files, see [http://msdn.microsoft.com/en-us/library/cc197955(VS.95).aspx](http://msdn.microsoft.com/en-us/library/cc197955%28VS.95%29.aspx). For more details about security restrictions around Silverlight networking, see [http://msdn.microsoft.com/en-us/library/cc645032(VS.95).aspx](http://msdn.microsoft.com/en-us/library/cc645032%28VS.95%29.aspx).

## Out of browser applications

Out of browser applications are in-browser applications that the user chooses to install and run out of browser. Installation begins either by the user choosing "install" from the Silverlight context menu, or by the application launching the install process in response to user initiated input. In either case, Silverlight displays a dialog asking them to confirm that they wish to install this application from this web site.

Silverlight out of browser applications come in two flavors -- **sandboxed** and **trusted**. (Sandboxed is the default) The sandbox for out of browser applications is identical to the in-browser sandbox except for the following minor differences:

* **Increased isolated storage size** -- Because the user has made an explicit decision to install the application, there is less concern of denial of service attacks, so the default isolated storage quota size is increased to 25MB versus 1MB in-browser.
* **Ability to resize the application window** -- In order to prevent click jacking attacks, where the Silverlight window moves out from under you just as you're about to click it, Silverlight apps may only resize themselves in response to user initiated. Silverlight apps may decide their initial location upon launch, subject to the constraint that they must be fully on the screen, however once launched the app may not programmatically change his position. (The end-user may of course move the window as they see fit)

Trusted apps run outside the sandbox. You should not install/run a trusted app unless you would be willing to run a .exe from the same site. Trusted apps get the following additional permissions that sandboxed apps don't have:

* native integration e.g. automation via COM
* reading/writing files on the local disk (not restricted to isolated storage)
* user initiated requirements waived
* cross domain networking is allowed without cross domain policy files
* sockets are allowed without policy file

With privilege comes responsibility. Trusted apps can have security holes, e.g. an mail application that automatically runs any .exe that's mailed to you. Because Silverlight apps are written in managed code, they are less susceptible to common security mistakes such as buffer overruns and integer overflows, however care must still be taken.

Trusted applications do not have completely unrestricted access to all APIs, however the limitations are for portability rather than security reasons. Somewhat confusingly, some of these APIs throw SecurityExceptions even though the motivation is portability rather than security.

Trusted apps are not trusted until they are installed & run out of browser -- the same .xap file run inside the browser is not trusted.

## Installing & updating Silverlight

Silverlight requires administrator privileges to install.

<http://www.microsoft.com/silverlight/resources/documentation/grouppolicysettings.aspx> contains information about using group policy to configure Silverlight. Silverlight is secure by default and does not require any changes to the default configuration, however for additional protection registry keys are available to disable the following features: webcam (AllowWebcam=0), UDP multicast networking (AllowUdpMulticastClientApi=0), install trusted apps (AllowInstallOfElevatedTrustApps=0), and run trusted apps (AllowLaunchOfElevatedTrustApps=0). On Windows, the keys are DWORD values under HKey\_Local\_Machine\Software\Microsoft\Silverlight , on Mac they are boolean preferences under under com.microsoft.silverlight.

Silverlight receives updates via Microsoft Update and/or WSUS. In addition, Silverlight has an "auto updater", whereby Silverlight will check for updates when it's run. For performance and privacy reasons, Silverlight will not check on every launch, rather every 30 days if user has opted into Microsoft Update, otherwise every 7 days. Macs do not have Microsoft Update but do support the auto updater. Microsoft Silverlight uses an in-place update versioning model, so each release of Silverlight is effectively a patch to the previous release. There is never more than one version of Silverlight installed on the machine.

## Implementation of the sandbox

Silverlight has been developed entirely under Microsoft's Secure Development Lifecycle (SDL), a methodology that includes threat modeling of designs, fuzz testing file formats and network interfaces, penetration testing, and static analysis tools. See <http://msdn.microsoft.com/en-us/security/cc448177.aspx> for more information about the Secure Development Lifecycle.

Silverlight is implemented with a combination of native and managed code. To reduce security surface area, the managed portion is divided into three categories -- Transparent, SafeCritical, and Critical. Transparent code (the default) runs with the same sandboxed permissions as application code. Critical and SafeCritical, on the other hand, can do unsafe things such as call into native code. Transparent code cannot call directly into Critical code, so SafeCritical acts as a boundary layer where most of the security checks are done. For more information about this discipline, see [http://msdn.microsoft.com/en-us/library/dd470128(VS.95).aspx](http://msdn.microsoft.com/en-us/library/dd470128%28VS.95%29.aspx).

# Creating secure web sites using Silverlight

Web sites that use Silverlight will have many of the same considerations as web sites that don't use Silverlight.

## Cross site scripting (XSS)

Like traditional HTML development, Silverlight application authors need to be aware of cross site scripting (XSS) issues. An XSS vulnerability allows the attacker to run code on the client machine as if it were from the victim site (victim.com). This gives the attacker access to any cookies, isolated storage, and authentication info that the browser would give victim.com access to. The impact of this depends on the site, for an online retailer, the attacker might be able to use those cookies to order merchandise in the user's name. For more information about XSS, see <http://en.wikipedia.org/wiki/Cross-site_scripting>.

Silverlight XSS issues are possible but less likely than in traditional HTML development. XSS issues typically arise from inserting attacker-controlled strings into markup, without first validating or escaping the attacker controlled string. However, Silverlight programs typically do not combine strings together to create xaml. For example, to display a string on the screen, one would write

textblock.Text = attackerString;

rather than doing it by combining markup strings:

XamlReader.Load("<TextBlock Text=’” + attackerString + "’/>”);

In the first approach, because TextBlock.Text understands the string is to be displayed rather than parsed & run as markup/code, there is no vulnerability. (One could also use data binding rather than setting the Property programmatically)

XSS issues are possible in Silverlight, some examples include:

* Calling XamlReader.Load() on an attacker-provided string (or partially provided string).
* Calling AssemblyPart.Load() on an attacker-provided DLL
* Creating XML by combining strings, e.g. in order to send to a REST service. (System.Xml provides better ways of creating XML, including System.Xml.XmlWriter and System.Xml.Linq.XElement)
* using Silverlight to create HTML via System.Windows.Browser (see also EnableHtmlAccess)
* Serving an attacker’s .xap from your web server, perhaps by allowing user uploads. (Note that Silverlight will only recognize a file as a xap if the MIME type is application/x-silverlight-app, which prevents "gifar” attacks where a valid .docx file is also a valid .xap, see <http://hackaday.com/2008/08/04/the-gifar-image-vulnerability> for more information about gifar attacks)

## Isolating Silverlight from HTML/JavaScript

Silverlight apps run as part of a larger HTML page, but what happens if the Silverlight app doesn't trust the HTML page or vice versa? Silverlight allows one to lock down communications in both directions, and defaults to the most locked down setting when the .xap and the HTML page come from different domains.

The EnableHtmlAccess property controls whether the Silverlight .xap can call JavaScript methods & modify the HTML page. EnableHtmlAccess is specified on the <object> tag in the hosting HTML page. If the .xap and the HTML page come from the same domain, the property defaults to true; if they are from different domains, the property defaults to false.

The ExternalCallersFromCrossDomain property and the [ScriptableTypeAttribute] / [ScriptableMemberAttribute] are specified in the .xap, and control whether JavaScript can call methods in the Silverlight .xap. In order for JavaScript to call a method, it must pass the ExternalCallersFromCrossDomain check, and the class in question must be decorated with [ScriptableTypeAttribute], and the method in question decorated with [ScriptableMemberAttribute]. No classes/methods provided by Silverlight are decorated with [Scriptable\*], so JavaScript can only call classes/methods specified by the application. ExternalCallersFromCrossDomain has two settings, ScriptableOnly and NoAccess, with NoAccess the default.

## Isolating Silverlight code from other Silverlight code

In order to isolate your Silverlight code from other Silverlight code, you must run in separate applications hosted on separate domains. If your application loads QuestionableCode.dll, that code will be run with the same privileges & domain of origin as your application.

## Sending data between Silverlight apps

Silverlight apps running in separate <object> tags may communicate either through JavaScript (see previous section) or by using System.Windows.Messaging.LocalMessageReceiver/LocalMessageSender, also known as **local messaging**. Local messaging allows apps to send strings to other apps listening on an agreed-upon channel name. By default, senders send to and receivers receive only from apps from the same domain of origin. Senders and receivers may also agreed to send/receive messages from specific other domains.

Receivers & senders may also agree to use the Global receiver name scope. Global should be used with caution, as both sender and receiver can come from any domain, with no way for the sender to ensure that they are sending to the "right" receiver that claims that channel. (The receiver, on the other hand, is told which domain of origin sent the message)

## Preventing unauthorized reuse of your .xap

To check that the HTML page hosting your .xap is the page you're expecting, include the following code in your App constructor:

 public App() {

 // Check that the xap is hosted on the page we

 // expect it to be hosted on. Make sure to do

 // this in the App constructor rather than the

 // Startup event or page constructor, as exceptions

 // thrown then won't bring down the app.

 **if (App.Current.Host.Settings.EnableHTMLAccess == false)**

 **throw new Exception();**

 **string htmlurl = System.Windows.Browser.HtmlPage.Document.DocumentUri.ToString();**

 **if (htmlurl != "http://foo.com/mypage.html")**

 **throw new Exception();**

 this.Startup += this.Application\_Startup;

 this.Exit += this.Application\_Exit;

 this.UnhandledException += this.Application\_UnhandledException;

 InitializeComponent();

 }

(this requires the expected page to grant HTML access)

To prevent your web site from being displayed inside an iframe, you can use standard HTML frame busting techniques such as <http://grizzlyweb.com/webmaster/javascripts/framesbuster.asp>.

## Validating server input

Silverlight doesn't change the way you validate and authenticate requests on the server.

**Validate input** -- like non-Silverlight sites, servers will want to validate input to make sure the input data doesn't contain SQL commands, HTML/JavaScript that will later be displayed on the HTML portion of the website, or other malicious data. Bear in mind that the attacker may not use the Silverlight .xap at all -- he may send raw http requests to your server.

**Securing Web service interfaces** -- Silverlight applications, like AJAX apps, often involve web service calls. These interfaces need to be secured & input to these APIs needs to be validated.

**Cross-site request forgery (CSRF)** -- For sensitive Web service calls/http requests, the server will want to verify they came from the right web page. In a cross-site request forgery, the request comes from a valid user's browser, but while the user is viewing a page from evil.com. (Typically this is while the user is legitimately logged in to your site in another browser tab) Typical mitigations include use of the HTTP Referer header in conjunction with a unique session ID.

 See <http://www.microsoft.com/downloads/details.aspx?FamilyID=7cef15a8-8ae6-48eb-9621-ee35c2547773&displaylang=en> for more details.

## Protecting information inside a .xap

Storing sensitive information (e.g., passwords, private keys, or proprietary algorithms) in a .xap file is problematic. In general, it's not possible to protect such information from a determined attacker. DLLs may be obfuscated using .NET obfuscators.

(see <http://www.olsonsoft.com/blogs/stefanolson/post/Selecting-an-obfuscator-for-Silverlight.aspx> for recommendations from a non-Microsoft source).

However while obfuscation may slow attacker down it can't prevent the attacker from the compiling & understanding the application logic. We are not aware of any tools that obfuscate xaml.

## Protecting data in isolated storage

When putting sensitive data in isolated storage, consider encrypting it. Like cookies, isolated storage can be accessed by the machine's administrator. Cookies & isolated storage are accessible to any app from the same domain of origin. This means that an attacker who controls access to DNS from the user's (victim's) machine, thereby compromising the DNS server itself or via a man in the middle attack against the victim machine, can create a web page that appears to come from your web site and can access the cookies/isolated storage. Finally, if an attacker can get your site to host his .xap, again his .xap will be from your site of origin and have access to the cookies/isolated storage.

For more details on isolated storage, see <http://msdn.microsoft.com/en-us/magazine/dd458794.aspx> .

## Special considerations for Trusted Apps

When writing a trusted out of browser application, you'll want to take extra care because a security hole in your application can allow an attacker to run trusted code out of the sandbox. In addition to the considerations above, Microsoft recommends the following for trusted apps:

**Use sandboxed out of browser apps where possible** -- Don't use trusted apps if you don't need to.

**Be extra vigilant about XSS issues** – an XSS problem in your application can let the attacker take control of the application. And if that application is trusted and running outside of the sandbox, the attacker is outside of the sandbox. See section 3.1 for details on XSS.

**Sign your application** – Use signtool.exe to put an Authenticode signature on your .xap file. This will help your users be sure they are getting the genuine application and not a tampered version of your app. It also helps prevent man in the middle attacks against the install and update of your app. If you can't use a signature issued by a Certificate Authority (CA), you should at least self-signed the application to prevent man in the middle attacks against upgrading your app.

**Consider protecting code downloads against man in the middle attacks** -- A "man in the middle" (MITM) attack is one where the attacker can listen to, intercept, and modify any network traffic between the client and server computers. If the attacker can do a MITM attack, then he can modify any code, .xap’s, and .dll’s that the client downloads from the server. If the client executes that code, the attacker controls the program.

Code signing protects your main .xap against MITM -- the attacker can modify the .xap, but it won't have your signature. For the install of the tampered .xap, Silverlight will detect the lack of a valid signature and display the "unknown publisher" install dialog. When upgrading the application using Silverlight's Application.CheckAndDownloadUpdateAsync() method, Silverlight will check that the signature on the updated .xap matches the signature on the original .xap. (This is why even a self signed certificate is better than no certificate)

Code signing only applies to the main .xap. If your application downloads additional .xaps and .dll’s, there is no code signing support for them. Consider using HTTPS for such downloads, as HTTPS offers protection against MITM attacks.

**Verify your site of origin** -- check that your .xap file was downloaded from the website you expect by checking the following in your App constructor:

 public App() {

 // Check that the xap is hosted on the server we

 // expect it to be hosted from. Make sure to do

 // this in the App constructor rather than the

 // Startup event or page constructor, as exceptions

 // thrown then won't bring down the app.

 **string xapServer = this.Host.Source.ToString();**

 **if (xapServer != "http://localhost:60338/TestApp.xap") {**

 **throw new Exception("Application came from an unexpected server");**

 **}**

 this.Startup += this.Application\_Startup;

 this.Exit += this.Application\_Exit;

 this.UnhandledException += this.Application\_UnhandledException;

 InitializeComponent();

 }

The reason this can be a problem is that the trusted app may assume that its server of origin is trusted. Suppose your application was intended to come from good.example.com, but evil.example.com copied your .xap onto their server and serves it out from their domain. Then if the application assumes that its server of origin (now evil.example.com) is trustworthy, and downloads code from that server, the app is compromised. Similarly, the app could unwittingly send sensitive user data to the server.

**No administrator privileges** -- As a defense in depth measure on Windows, Silverlight out of browser apps run without administrator privileges. If the Silverlight app is Run As administrator, Silverlight will remove administrator privileges by launching a second restricted process and ending the original process. The restricted permissions are similar to the User Account Control (UAC) restrictions on Windows Vista and above. Like UAC, this is defense in depth only and shouldn't be relied upon as a foolproof security measure -- do not install trusted apps if you don't trust them with the user's full permission set.

## Security APIs

Silverlight offers a number of security services & APIs to ensure the integrity and privacy of data passed over the network.

**HTTPS** is supported, both for the .xap itself and for any web requests it makes. If the .xap file itself is loaded over HTTPS, the application must do all of its http networking over https rather than http.

Silverlight has support for the HTTP **Referer** header, allowing the server to know the domain that the .xap came from. (The full path to the .xap may or may not be provided depending on the browser) Referer is always supplied when using ClientHttpWebRequest and when using Image/MediaElement (although some proxies may remove it); when using BrowserHttpWebRequest, some browsers set it and others do not. It is not possible for Silverlight apps to change the value of this header. Server authors should be careful relying on the Referer header for security, as attackers running non-Silverlight apps outside of the browser may set the Referer header to anything they want.

System.Security.Cryptography contains APIs for **AES private key encryption, SHA1** and **SHA256 hashing,** and **HMAC digital signatures.** (Silverlight does not include public-key encryption) See [http://msdn.microsoft.com/en-us/library/cc265159(VS.95).aspx](http://msdn.microsoft.com/en-us/library/cc265159%28VS.95%29.aspx) for more details.

Silverlight supports **PlayReady digital rights management (DRM)** to control access to media files. Silverlight supports protecting VC-1 content, H.264 content and protecting off-line scenarios. See [http://msdn.microsoft.com/en-us/library/cc838192(VS.95).aspx](http://msdn.microsoft.com/en-us/library/cc838192%28VS.95%29.aspx) for more details.

# Conclusion

Silverlight was built from the ground up with security as a top priority. Silverlight sandboxes web apps to keep the user safe from malicious web sites. Silverlight also helps keep your web site safe from attack by providing secure defaults and making XSS bugs less difficult to write. By understanding the Silverlight security model you can build secure web sites that use Silverlight.

# Additional resources

<http://www.microsoft.com/downloads/details.aspx?FamilyID=7cef15a8-8ae6-48eb-9621-ee35c2547773&displaylang=en> contains additional information about XSS, CSRF, and other attacks that Silverlight web sites may face, and how to use the Silverlight APIs to mitigate those attacks.