see origin from a downloaded file chrome

Allowing cross-origin use of images and canvas.

HTML provides a crossorigin attribute for images that, in combination with an appropriate CORS header, allows images defined by the <img> element that are loaded from foreign origins to be used in a <canvas> as if they had been loaded from the current origin.

See CORS settings attributes for details on how the crossorigin attribute is used.

Security and tainted canvases.

Because the pixels in a canvas's bitmap can come from a variety of sources, including images or videos retrieved from other hosts, it's inevitable that security problems may arise.

As soon as you draw into a canvas any data that was loaded from another origin without CORS approval, the canvas becomes tainted . A tainted canvas is one which is no longer considered secure, and any attempts to retrieve image data back from the canvas will cause an exception to be thrown.

If the source of the foreign content is an HTML <img> or SVG <svg> element, attempting to retrieve the contents of the canvas isn't allowed.

If the foreign content comes from an image obtained from either as HTMLCanvasElement or ImageBitMap , and the image source doesn't meet the same origin rules, attempts to read the canvas's contents are blocked.

Calling any of the following on a tainted canvas will result in an error:

Calling getImageData() on the canvas's context Calling toBlob() on the <canvas> element itself Calling toDataURL() on the canvas.

Attempting any of these when the canvas is tainted will cause a SecurityError to be thrown. This protects users from having private data exposed by using images to pull information from remote web sites without permission.

Storing an image from a foreign origin.

In this example, we wish to permit images from a foreign origin to be retrieved and saved to local storage. Implementing this requires configuring the server as well as writing code for the web site itself.

Web server configuration.

The first thing we need is a server that's configured to host images with the Access-Control-Allow-Origin header configured to permit cross-origin access to image files.

Let's assume we're serving our site using Apache. Consider the HTML5 Boilerplate Apache server configuration file for CORS images, shown below:

In short, this configures the server to allow graphic files (those with the extensions ".bmp", ".cur", ".gif", ".ico", ".jpg", ".jpeg", ".png", ".svg", ".svgz", and ".webp") to be accessed cross-origin from anywhere on the internet.

Implementing the save feature.

Now that the server has been configured to allow retrieval of the images cross-origin, we can write the code that allows the user to save them to local storage, just as if they were being served from the same domain the code is running on.

The key is to use the crossorigin attribute by setting crossOrigin on the HTMLImageElement into which the image will be loaded. This tells the browser to request cross-origin access when trying to download the image data.

Starting the download.

The code that starts the download (say, when the user clicks a "Download" button), looks like this:

We're using a hard-coded URL here (imageURL), but that could easily come from anywhere. To begin downloading the image, we create a new HTMLImageElement object by using the Image() constructor. The image is then configured to allow cross-origin downloading by setting its crossOrigin attribute to "Anonymous" (that is, allow non-authenticated downloading of the image cross-origin). An event listener is added for the load event being fired on the image element, which means the image data has been received.

Finally, the image's src attribute is set to the URL of the image to download; this triggers the download to begin.

Receiving and saving the image.

The code that handles the newly-downloaded image is found in the imageReceived() method:

imageReceived() is called to handle the "load" event on the HTMLImageElement that receives the downloaded image. This event is triggered once the downloaded data is all available. It begins by creating a new <canvas> element that we'll use to convert the image into a data URL, and by getting access to the canvas's 2D drawing context (CanvasRenderingContext2D) in the variable context .

The canvas's size is adjusted to match the received image, then the image is drawn into the canvas using drawImage() . The canvas is then inserted into the document so the image is visible.

Now it's time to actually save the image locally. To do this, we use the Web Storage API's local storage mechanism, which is accessed through the localStorage global. The canvas method toDataURL() is used to convert the image into a data:// URL representing a PNG image, which is then saved into local storage using setItem() .

Google Chrome won't allow IFRAME to load an HTML file.

I've got this iframe that is working fine on FF, but it doesn't seem to load up on Google Chrome.

Could someone tell me why this is happening?

My iframe :

5 Answers 5.

I tested it and it works OK in chrome on my end. My guess is that you have it set up so the page containing the iframe is being accessed via https. If the page is https, you cannot load a iframe on that page that is not https. It will result in a "mixed-content error" and for security purposes it will not load.

It works in FF because FF is more lax about this security restriction and Chrome happens to be more strict on mixed-content errors.

How to force open links in Chrome not download them?

I want to open a link that is .psd format with Photoshop when clicked in Google Chrome like Firefox that asks me to open or download the file. But Google Chrome downloads the file automatically. How can I force to open the links in Chrome without downloading? The links are for local files.

6 Answers 6.

To make certain file types OPEN on your computer, instead of Chrome Downloading.

You have to download the file type once, then right after that download, look at the status bar at the bottom of the browser. Click the arrow next to that file and choose "always open files of this type". DONE.

Now the file type will always OPEN using your default program.

To reset this feature, go to Settings / Advance Settings and under the "Download.." section, there's a button to reset 'all' Auto Downloads.

Hope this helps.. :-)

Visual Instructions found here:

It can be achieved via an extension:

For Chrome, load undisposition If the file loading is ASCII then colour coding may be desirable, that can be done via the Syntaxtic extension btw, for Firefox load the InlineDisposition add-on.

Google, as of now, cannot open w/out saving. As a workaround, I use IE Tab from the Chrome Store. It is an extension that runs IE - which does allow opening w/ out saving- inside of the Chrome browser application.

Not the best solution, but it's an effective "patch" for now.

Just found your question whilst trying to solve another problem I'm having, you will find that currently Google isn't able to perform a temporary download so therefore you have to download instead.

I think the question was about to open a local file directly instead of downloading a local file to the download folder and open the file in the download folder, which seems not possible in Chrome, except some add-on mentioned above.

My workaround would be to right click -> Copy the link location Windows + R and paste the link there and Enter It will go to the file directly.

To open docs automatically in Chrome without them being saved;

Go to the the three vertical dots on your top far right corner in Chrome.

Scroll down to Settings and click.

Scroll down to Show advance settings.

Scroll down to Downloads under Download location: click the Change button and chose tmp folder. Then just close the screen.

Click on any attachments and a small box to the left will appear, it should automatically open if you click on it.

When the bottom left box appears it will contain an arrow; click on it and choose the option "Always open files of this type". Going forward it will open the file instantly instead of the small box appearing to the left and you having to click on it to open. You will have to do it just once for various files such PDF, Excel 2010, Excel 2013 Word, ect.

Background.

Windows uses a simple technique to keep track of which binary files were downloaded from the Internet (or a network share).

Each downloaded file is is tagged with a hidden NTFS Alternate Data Stream file named Zone.Identifier. You can check for the presence of this "Mark of the Web" (MotW) using dir /r or programmatically, and you can view the contents of the MotW stream using Notepad:

Within the file, the ZoneTransfer element contains a ZoneId element with the ordinal value of the URLMon Zone from which the file came. The value 3 indicates that the file is from the Internet Zone .

Aside: One common question is "Why does the file contain a Zone Id rather than the original URL? There's a lot of cool things that we could do if a URL was preserved!" The answer is mostly related to privacy—storing a URL within a hidden data stream is a foot gun that would likely lead to accidental disclosure of private URLs. This problem isn't just theoretical—the Alternate Data Stream is only one mechanism used for MotW. Another mechanism involves writing a <!–saved from url=> comment to HTML markup; that form does include a raw URL. A few years ago, attackers noticed that they could use Google to search for files containing a MOTW of the form <!—saved from url(0042)ftp://username:secretpassword@host.com –> and collect credentials. Oops.

Update : Microsoft apparently either forgot or decided the tradeoff was worth it. Windows 10 includes the referrer URL, source URL and other information.

Browsers and other internet clients (e.g. email and chat programs) can participate in the MOTW-marking system by using the IAttachmentExecute interface's methods or by writing the Alternate Data Stream directly. Update: Chrome uses IAttachmentExecute and thus includes the URL information on Windows 10. Firefox writes the Alternate Data Stream directly (but as of February 2021, it includes the URL).

Handling Marked Files.

Windows and some applications treat files with a Mark-of-the-Web differently than those without. For instance, examining a downloaded executable file's properties shows the following notice:

More importantly, attempting to run the executable using Windows Explorer or ShellExecute() will first trigger evaluation using SmartScreen Application Reputation (Win8+) and any installed anti-virus scanners. The file's digital signature will be checked, and execution will be confirmed with the user, either using the older Attachment Execution Services prompt, or the newer UAC elevation prompt:

Microsoft Office documents bearing a MotW open in Protected View, a security sandbox that attempts to block many forms of malicious content.

Trivia: Some applications inherit protections against files bearing a MotW, but don't have any user-interface that explains what is going on. For instance, if you download a CHM with a MotW, its HTML content will not render until you unblock it using the "Always ask before opening this file" or the "Unblock" button:

What Could Go Wrong?

With such a simple scheme, what could go wrong? Unfortunately, quite a lot.

Internet Clients Must Participate.

The first hurdle is that Internet clients must explicitly mark their downloads using the Mark-of-the-Web, either by calling IAttachmentExecute or by writing the Alternate Data Stream directly. Most popular clients will do so, but support is neither universal nor comprehensive.

For instance, for a few years, Firefox failed to mark downloads if the user used the Open command instead of Save.

In other cases, some browser plugins may allow attackers to save files to disk and bypass MotW tagging.

Microsoft Outlook (tested v2010) and Microsoft Windows Live Mail Desktop (tested v2012 16.4.3563.0918) both tag message attachments with a MotW you double-click on an attachment or right-click and choose Save As. Unfortunately, however, both clients fail to tag attachments if the user uses drag-and-drop to copy the attachment to somewhere in their filesystem. This oversight is likely to be seen in many different clients, owing to the complexity in determining the drop destination.

Target File System Must Be NTFS.

The Zone.Identifier stream can only be saved in an NTFS stream. These streams are not available on FAT32-formatted devices (e.g. some USB Flash drives), CD/DVDs, or the new ReFS file system introduced with Windows Server 2012.

If you copy a file tagged with a MotW to a non-NTFS filesystem (or try to save it to such a file system to start with), the Mark of the Web is omitted and the protection is lost.

Originating Location Must Be Internet or Restricted Zone.

The IAttachmentExecute:Save API will not write the MotW unless the URL provided in the SetSource method is in the Internet or Restricted Sites zone.

On some systems, code may have added domains to the user's Trusted Sites zone without their knowledge.

The Attachment Execution Services API also will not write a MoTW to an Internet Zone download if that Zones' Launching applications and unsafe files option is set to Enable . Normally, Windows will howl about this unsafe configuration, but there's a registry key that turns off the security-settings-are-bad warning.

Not Disabled by Policy.

Writing of the MoTW can be suppressed in the AttachmentExecuteServices API via Group Policy. In GPEdit.msc , see Administrative Templates > Windows Components > Attachment Manager > Do not preserve zone information in file attachments.

A value of 1 will prevent the MoTW from being written to files.

Source scheme must be HTTP/HTTPS/FILE/FTP-like.

If the source of the download is a data URI, the browser has no great way to know what marking to put on the file. blob URIs have a similar issue, but because blob URIs only exist within a security context (https://example can create blob:https://example/guid) you can write that security context URL as the MoTW to get proper handling.

For data URIs or other anonymous sources, writing a default of about:internet is a common conservative choice to ensure that the file was treated as if it came from the Internet Zone. Similarly, about:untrusted causes the file to be treated as originating from the Restricted Sites Zone.

Origin Laundering via Archives.

One simple trick that attackers use to try to circumvent MotW protections is to enclose their data within an archive like a .ZIP, .7z, or .RAR file. Attackers may go further and add a password to block virus scanners; the password is provided to the victim in the attacking webpage or email.

In order to remain secure, archive extractors must correctly propagate the MotW from the archive file itself to each file extracted from the archive.

Despite being one of the worst ZIP clients available, Windows Explorer gets this right:

In contrast, 7-zip does not reliably get this right. Malware within a 7-zip archive can be extracted without propagation of the MotW. 7-zip v15.14 will add a MotW if you double-click an exe within an archive, but not if you extract it first. The older 7-zip v9.2 does not tag with MotW either way.

A quick look at popular archive extractors shows:

Windows Explorer – Not vulnerable WinRar 5.31 – Not vulnerable WinZip 20.0.11649 – Not vulnerable 7-Zip 15.14 – Vulnerable (bug) IZArc 4.2 – Vulnerable (Developer says: "Will be fixed in next version")

Tested another? Let me know your findings in the comments.

SmartScreen & the User May Unmark.

Finally, users may unmark files using the Unblock button on the file's Properties dialog, or by unticking the "Always ask before opening this file" checkbox. Similarly, on systems with Microsoft SmartScreen, SmartScreen itself may unmark the file (actually, it replaces the ZoneId with an (undocumented) value AppZoneId=4).

Allow Google Chrome to use XMLHttpRequest to load a URL from a local file.

When trying to do a HTTP request using XMLHttpRequest from a local file, it basically fails due to Access-Control-Allow-Origin violation.

However, I'm using the local web page myself, so I was wondering if there is any way to make Google Chrome allow these requests, which are from a local file to a URL on the Internet.

E.g., $.get('http://www.google.com/') fails when executing in a local file, but I've scripted the page myself and I'm using it myself, so it would be extremely useful if I could suppress it and load the URL.

So, how can I allow Google Chrome to load URLs using XMLHttpRequest from local files?