Autoplay Policy Changes

Note: The Autoplay Policy launched in M66 Stable for audio and video elements and is effectively blocking roughly half of unwanted media autoplays in Chrome. For the Web Audio API, the autoplay policy will launch in M70. This affects web games, some WebRTC applications, and other web pages using audio features. Developers will need to update their code to take advantage of the policy. More details can be found in the Web Audio API section below.

By François Beaufort

Dives into Chromium source code

Chrome's autoplay policies will change in April of 2018 and I'm here to tell you why and how this is going to affect video playback with sound. Spoiler alert: users are going to love it!

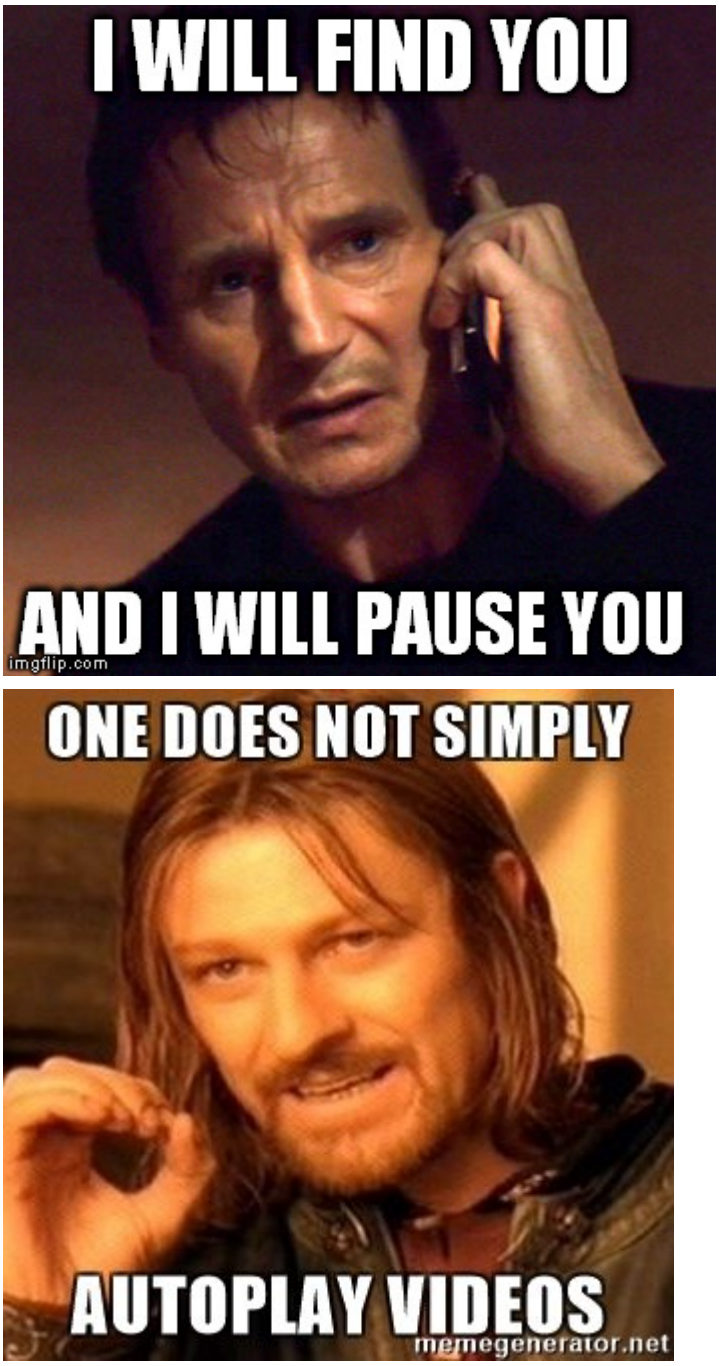


Figure 1. Internet memes tagged "autoplay" found on Imgflip and Imgur

New behaviors

As you may have noticed, web browsers are moving towards stricter autoplay policies in order to improve the user experience, minimize incentives to install ad blockers, and reduce data consumption on expensive and/or constrained networks. These changes are intended to give greater control of playback to users and to benefit publishers with legitimate use cases.

Chrome's autoplay policies are simple:

- Muted autoplay is always allowed.
- Autoplay with sound is allowed if:
  - User has interacted with the domain (click, tap, etc.).
  - On desktop, the user's Media Engagement Index threshold has been crossed, meaning the user has previously play video with sound.
  - On mobile, the user has added the site to his or her home screen.
- Top frames can delegate autoplay permission to their iframes to allow autoplay with sound.

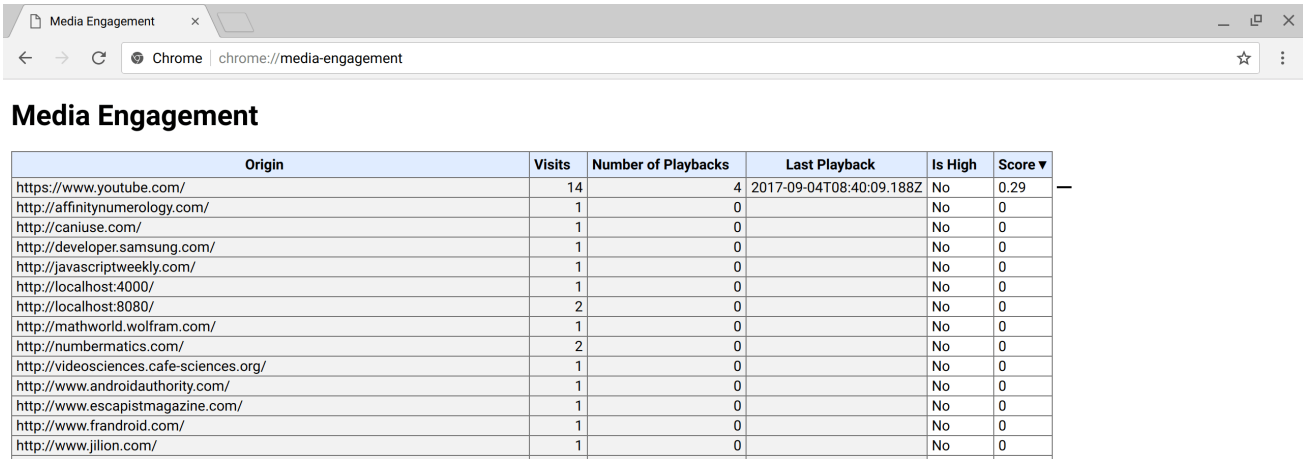
Media Engagement Index (MEI)

The MEI measures an individual's propensity to consume media on a site. Chrome's current approach is a ratio of visits to significant media playback events per origin:

- Consumption of the media (audio/video) must be greater than 7 seconds.
- Audio must be present and unmuted.
- Tab with video is active.
- Size of the video (in px) must be greater than 200x140.

From that, Chrome calculates a media engagement score which is highest on sites where media is played on a regular basis. When it is high enough, media playback is allowed to autoplay on desktop only.

User's MEI is available at the *chrome://media-engagement* internal page.



Origin	Visits	Number of Playbacks	Last Playback	Is High	Score ▾
https://www.youtube.com/	14	4	2017-09-04T08:40:09.188Z	No	0.29
http://affinitynumerology.com/	1	0		No	0
http://caniuse.com/	1	0		No	0
http://developer.samsung.com/	1	0		No	0
http://javascriptweekly.com/	1	0		No	0
http://localhost:4000/	1	0		No	0
http://localhost:8080/	2	0		No	0
http://mathworld.wolfram.com/	1	0		No	0
http://numbermatics.com/	2	0		No	0
http://videosciences.cafe-sciences.org/	1	0		No	0
http://www.androidauthority.com/	1	0		No	0
http://www.escapistmagazine.com/	1	0		No	0
http://www.frandroid.com/	1	0		No	0
http://www.jilion.com/	1	0		No	0

Figure 2. Screenshot of the *chrome://media-engagement* internal page

Developer switches

As a developer, you may want to change Chrome autoplay policy behaviour locally to test your website depending on user engagement.

- You can decide to disable entirely the autoplay policy by setting the Chrome flag "Autoplay Policy" to "No user gesture is required" at `chrome://flags/#autoplay-policy`. This allows you to test your website as if user were strongly engaged with your site and playback autoplay would be always allowed.
- You can also decide to make sure playback autoplay is never allowed by disabling use of MEI, applying autoplay policy to Web Audio, and whether sites with the highest overall MEI get playback autoplay by default for new users. This can be done with three internal switches with `chrome.exe --disable-features=PreloadMediaEngagementData,AutoplayIgnoreWebAudio, MediaEngagementBypassAutoplayPolicies`.

Iframe delegation

A feature policy allows developers to selectively enable and disable use of various browser features and APIs. Once an origin has received autoplay permission, it can delegate that permission to cross-origin iframes with a new feature policy for autoplay. Note that autoplay is allowed by default on same-origin iframes.

```
<!-- Autoplay is allowed. -->
<iframe src="https://cross-origin.com/myvideo.html" allow="autoplay">

<!-- Autoplay and Fullscreen are allowed. -->
<iframe src="https://cross-origin.com/myvideo.html" allow="autoplay; fullscreen">
```

When the feature policy for autoplay is disabled, calls to `play()` without a user gesture will reject the promise with a `NotAllowedError` DOMException. And the autoplay attribute will also be ignored.

Warning: Older articles incorrectly recommend using the attribute `gesture=media` which is not supported.

Example scenarios

Example 1: Every time a user visits *VideoSubscriptionSite.com* on their laptop they watch a TV show or a movie. As their media engagement score is high, autoplay is allowed.

Example 2: *GlobalNewsSite.com* has both text and video content. Most users go to the site for text content and watch videos only occasionally. Users' media engagement score is low, so autoplay wouldn't be allowed if a user navigates directly from a social media page or search.

Example 3: *LocalNewsSite.com* has both text and video content. Most people enter the site through the homepage and then click on the news articles. Autoplay on the news article pages would be allowed because of user interaction with the domain. However, care should be taken to make sure users aren't surprised by autoplaying content.

Example 4: *MyMovieReviewBlog.com* embeds an iframe with a movie trailer to go along with their review. The user interacted with the domain to get to the specific blog, so autoplay is allowed. However, the blog needs to explicitly delegate that privilege to the iframe in order for the content to autoplay.

Chrome enterprise policies

It is possible to change this new autoplay behaviour with Chrome enterprise policies for use cases such as kiosks or unattended systems. Check out the Policy List help page to learn how to set these new autoplay related enterprise policies:

- The "AutoplayAllowed" policy controls whether autoplay is allowed or not.
- The "AutoplayWhitelist" policy allows you to specify a whitelist of URL patterns where autoplay will always be enabled.

Best practices for web developers

Audio/Video elements

Here's the one thing to remember: Don't ever assume a video will play, and don't show a pause button when the video is not actually playing. It is so important that I'm going to write it one more time below for those who simply skim through that post.

Key Point: Don't ever assume a video will play, and don't show a pause button when the video is not actually playing.

You should always look at the Promise returned by the play function to see if it was rejected:

```
var promise = document.querySelector('video').play();

if (promise !== undefined) {
  promise.then(_ => {
    // Autoplay started!
  }).catch(error => {
    // Autoplay was prevented.
    // Show a "Play" button so that user can start playback.
  });
}
```

Warning: Don't play interstitial ads without showing any media controls as they may not autoplay and users will have no way of starting playback.

One cool way to engage users is about using muted autoplay and let them chose to unmute (see code snippet below). Some websites already do this effectively, including Facebook, Instagram, Twitter, and YouTube.

```
<video id="video" muted autoplay>
<button id="unmuteButton"></button>

<script>
  unmuteButton.addEventListener('click', function() {
    video.muted = false;
  });
</script>
```

Web Audio

Note: The Web Audio API will be included in the Chrome autoplay policy with M70 (October 2018).

First, be reminded that it is good practice to wait for a user interaction before starting audio playback as user is aware of something happening. Think of a "play" button or "on/off" switch for instance. You can also simply add an "unmute" button depending on the flow of the app.

Key Point: If an `AudioContext` is created prior to the document receiving a user gesture, it will be created in the "suspended" state, and you will need to call `resume()` after a user gesture is received.

If you create your `AudioContext` on page load, you'll have to call `resume()` at some time after the user interacted with the page (e.g., user clicked a button).

```
// Existing code unchanged.
window.onload = function() {
  var context = new AudioContext();
  // Setup all nodes
  ...
}

// One-liner to resume playback when user interacted with the page.
document.querySelector('button').addEventListener('click', function() {
  context.resume().then(() => {
    console.log('Playback resumed successfully');
  });
});
```

You may also create the **`AudioContext`** only when user interacts wit the page.

```
document.querySelector('button').addEventListener('click', function() {
  var context = new AudioContext();
  // Setup all nodes
  ...
});
```

To detect whether browser will require user interaction to play audio, you can check the **`state`** of the **`AudioContext`** after you've created it. If you are allowed to play, it should immediately switch to **`running`**. Otherwise it will be **`suspended`**. If you listen to the **`statechange`** event, you can detect changes asynchronously.

For info, checkout the small Pull Request that fixes Web Audio playback due to these autoplay policy changes for https://airhorner.com.

By default, the Web Audio API is not currently affected by the autoplay policy. To enable the autoplay policy for Web Audio, launch Chrome with the following internal switch: **`chrome.exe --disable-features=AutoplayIgnoreWebAudio`**.

Note: Web Audio FAQs can be found here.

Feedback

Was this page helpful?

YES NO

Great! Thank you for the feedback.

Sorry to hear that. Please open an issue and tell us how we can improve.

Last updated July 24, 2018.