# chrome.tabCapture.captureOffscreenTab - New Chrome API Proposal

**Proposal Date**
July 2015

**Who are the primary Eng and PM contacts for this API? (email addresses)**
Eng: miu@chromium.org, mfoltz@chromium.org
PM: skonig@chromium.org

**What are the relevant bug(s)?**
High-level: https://crbug.com/412331
Tracking: https://crbug.com/490890
Plan is for this to be used by the Cast Extension replacement: https://crbug.com/450767

**Which team will be responsible for this API? (team email address)**
The Chrome Multiscreen team.  I myself (miu@) am already listed in src/chrome/browser/extensions/api/tab_capture/OWNERS for the current chrome.tabCapture.capture() API.  My team does not have a non-Google-internal e-mail address, so feel free to use me directly as a contact for now.

**Do you know anyone else, internal or external, that is also interested in this API?**
This would enable the development of multi-screen web sites, and it will be a functional component of the Chrome implementation of new W3C Presentation API (http://w3c.github.io/presentation-api/).

**What's a reasonable Chrome milestone target for landing most of the code on Trunk?**
M47.  Implementation is close to complete (candidate: https://codereview.chromium.org/1221483002/), as our working prototype was simple and was able to leverage a lot of the existing tab capture implementation.

**Overview**
Creates an off-screen tab that renders web content and captures its audio/video into a MediaStream.  The MediaStream can then be connected to a remoting technology, such as WebRTC or Cast Streaming, for display on a remote device.  This API is identical to the current chrome.tabCapture.capture() API, except that it adds the creation and ownership of an off-screen tab.

Off-screen tabs are isolated from the user's normal browser experience.  They do not show up in the browser window or tab strip, nor are they visible to extensions (e.g., via the chrome.tabs.* APIs). They cannot access any interactive resources such as keyboard/mouse input, media recording devices (e.g., web cams), copy/paste to/from the system clipboard, etc. Navigation only to http, https, and data URIs will be allowed.

Sandboxing: By default, an off-screen tab uses its own isolated, off-the-record "incognito" profile and has no access to any personal data such as cookies, HTTP auth credentials, or local filesystem access. It will be allowed navigation to any domain.

Initially, this will be a dev-channel-only API. After a maturity and external developer review period, release it as a stable-channel API. This is what we did for chrome.tabCapture.capture().

**Use cases**
This new API would enable the creation of web sites that have a local display/interaction component and a remote content display component.

The original motivation for this API is to provide a piece of functionality needed for Chrome's implementation of the new W3C Presentation API. The W3C Presentation API provides the means to use the web platform for starting multi-screen presentations. There are two main operating modes: 1) The "two user-agent" mode, where a presentation page is fully rendered on a remote device; and 2) the "one user-agent" mode, where the remote device lacks web rendering capabilities and so Chrome needs to simulate that rendering locally. This latter operating mode is where the ability to create and capture an off-screen tab is needed.

More context: While the W3C Presentation API provides the means on the web platform for starting multi-screen presentations, the Chrome Media Router framework is used to make decisions as to how content will be sourced, rendered and delivered to the desired remote devices. It makes these decisions based on the user choosing a remote device and the remote device's capabilities. From there, if the 1UA operating mode is required, this new extension API will be used to create and capture an off-screen tab.

Other use case examples:

- A Google Slides presentation's navigation display and outline notes can be shown on a user's laptop while the presentation slides are shown to an audience on a large projector display.
- A media jukebox app, where a playlist is built and controlled locally, and the content is played back on a remote display for a wider audience.
- A "board game," where multiple users can interact with the game on their own devices and share a single display of the game on a family room TV.
- Performance rendering: Most desktops/laptops have much better performance for rendering complex content (e.g., 3D scenes or HD video) than web-capable rendering devices (e.g., Chromecast, Android TV, or other "smart TV" implementations).

**Can you imagine any other similar use cases which your API might address?**

1. Tamper-proofing the DOM. Example: A web site may wish to render a point-of-sale receipt and take a single-frame screen-shot image. This would prevent possible tampering of the DOM by the user before a receipt is saved/printed.
2. Automated testing infrastructure, where a “test driver” can spawn off-screen tabs having “blank slate” profiles, and then record the results and compare them to expected results.
3. “DVR” recording app. Example: Record a livestreamed tennis match for personal playback at a later time.
4. Local multi-screen app development: Create an off-screen tab that has the exact “environment” expected for production, and view it on the local screen via a video element in another tab.

**How would you implement your desired features if this API didn't exist?**
It would be extremely difficult, and lead to an overall unmanageable, complex design. This API was primarily designed to be used by the Chrome Media Router component extension, a non-user-visible extension shipped with Chrome releases, which is an integral component in executing the W3C Presentation API features in Chrome. If this API didn’t exist, we’d have to fall-back to having a completely internal implementation, but would then need to employ some kind of complex hybrid design to special-case the handling of starting and routing presentation content internally (versus all other media routing being handled by the component extension).

**Is this something that could/should be part of the web platform, or an existing chrome.* API?**
This is new browser-level “container” functionality, and so it is more appropriate as a chrome.* API. Also, please see use case discussion above on how this relates to the web platform (W3C Presentation API).

**Does this API expose any functionality to the web?**
This API provides functionality similar to window.open(url, ’_blank’) (i.e., opening a new tab), but only to extensions/apps, and with significant additional restrictions as described throughout this document. Unlike window.open(), it does not give the extension direct access to its DOM, JavaScript environment, etc.; and, in fact, does not provide any control whatsoever other than implicit shutdown when the MediaStream is closed.

**Do you expect this API to be fairly stable? How might it be extended or changed in the future?**
It should start off pretty stable and likely remain that way. It may change a little as all the functional requirements of the W3C Presentation API and Chrome Media Router framework are tweaked, but we cannot fathom at this time what such changes might look like.

**If multiple apps/extensions used this API at the same time, could they conflict with each other? If so, how do you propose to mitigate this problem?**

No, there should be no conflicts between extensions.  This API is stateless insofar as there is no state preserved across extension reloads or browser restarts.  There is per-extension state to make sure only one instance of the same presentation URL+ID is spawned (future requests just create additional MediaStreams on the same off-screen tab).  In all, no extension can be affected by use of this API by any other extension.

**List every UI surface belonging to or potentially affected by your API:**
Absolutely none.  Users of this API must provide the UI surface for activation.  For example, the Chrome Media Router framework provides a UI to allow the creation of the off-screen tab and pick a remote device.

**Actions taken with app/extension APIs should be obviously attributable to an app/extension. Will users be able to tell when this new API is being used? How? Can it be spoofed?**
Well-behaving extensions will provide sufficient UI to make users aware of when this functionality is being used.  However, there is nothing the API directly provides itself to notify users that this API is being used.

*[[TBD--How to mitigate extensions that simply act as part of a “bot net” then?]]*

**Does this API impose any requirements on the Chrome Web Store ?**
No.

**Does this API have any interaction with other Chrome APIs? Does it impose any restrictions on other APIs that can be used in conjunction?**
Extensions using this API to spawn off-screen tabs will not have any mechanism to interact with the tabs programmatically.  In fact, by design, the off-screen tabs spawned by this API will not even be visible to the chrome.tabs.* APIs, any other extension APIs, or any other extension instances.

However, it’s worth noting here that programmatic messaging to/from the page content of the off-screen tabs will be possible via the web platform (W3C Presentation API) and/or any custom cloud messaging mechanisms provided by external web sites.

**How could this API be abused?**
An obvious concern is whether an extension could use this API to load an URL and then send the video capture of private user information or auth credentials to a remote site.  The design addresses this by requiring that the off-screen tab’s web content be rendered in a sandboxed environment via an incognito profile (**not** the shared incognito profile).

Performance-wise, tab capture can be expensive in terms of system resources (GPU readback, CPU for encoding audio/video, etc.).  To prevent a DOS attack on the local system, the implementation will restrict the number of simultaneously running off-screen tabs per extension.

**Imagine you're Dr. Evil Extension Writer, list the three worst evil deeds you could commit with your API (if you've got good ones, feel free to add more):**

1. Tracking user activity: Pull the titles from all open tabs in the browser, create an off-screen tab with a data URI containing the information, and send the resulting video capture of the web rendering to an evil cloud storage site.
2. Tracking user location: Create an off-screen tab that navigates to https://www.whatismyip.com/what-my-ip-says-about-me, capture and send the resulting video to an evil cloud storage site.
3. Grabbing a company's private data: An off-screen tab could be pointed to an "intranet" URL not protected by user auth, then capture and send the resulting video to an evil cloud storage site.
4. Take-over attack: Launch an off-screen tab that loads Flash content, leverages an as-yet-unknown zero-day exploit, breaks the sandbox, and therefore takes control over the user's machine.
5. Distributed computing: Extension spawns extra off-screen tabs that pull work from the cloud and execute JavaScript to utilize the local CPU/GPU to compute Julia sets, do protein folding simulations, or something even more evil.
6. Bot net: Malware extension spawns off-screen tabs that act as a member of a bot-net to, for example, launch DOS attacks on web sites.

**What security UI or other mitigations do you propose to limit evilness made possible by this new API?**
As discussed above, the main API restrictions will be:

1. Require the tabCapture permission.
2. Provide indication to the user of off-screen tab activity.

Content security issues are roughly the same as for extension APIs that can open tabs, inject/access content into/from tabs; as well as those considered when using the existing (now mature) chrome.tabCapture.capture() API. After much discussion, we don't believe off-screen tabs, as proposed here, would introduce significant additional issues.

**Could a consumer of your API cause any permanent change to the user's system using your API that would not be reversed when that consumer is removed from the system?**
No. The API is providing a stateless service.

**Draft Manifest Changes**
The API will start out as available on dev-channel only, requiring tabCapture permission to be activated. Private/internal Chrome extensions, such as the Chrome Media Router component extension, can be whitelisted for stable-channel use.

## Draft API spec

```
// Locates, or creates on-demand, an off-screen tab identified by a
// |startUrl| and |id|.  When created, the off-screen tab is automatically
// navigated to the given |startUrl|.  Then, capture is started and a
// MediaStream is returned via |callback|.
//
// |id| can be any client-provided string.  It will be used to distinguish
// off-screen tabs with the same |startUrl|.
//
// Off-screen tabs are isolated from the user's normal browser experience.
// They do not show up in the browser window or tab strip, nor are they
// visible to extensions (e.g., via the chrome.tabs.* APIs).
//
// An off-screen tab remains alive until one of three events occurs: 1. All
// MediaStreams providing its captured content are closed; 2. the page
// self-closes (e.g., via window.close()); 3. the extension that called
// captureOffscreenTab() is unloaded.
//
// Sandboxing: The off-screen tab does not have any access whatsoever to the
// local user profile (including cookies, HTTP auth, etc.).  Instead, it is
// provided its own dedicated, non-shared off-the-record profile.  Also, it
// cannot access any interactive resources such as keyboard/mouse input,
// media recording devices (e.g., web cams), copy/paste to/from the system
// clipboard, etc.
//
// Note: This is a new API, currently only available on Dev channel, and may
// change without notice.
//
// |options| : Constraints for the capture and returned MediaStream.
// |callback| : <code>null</code> on error.
static void captureOffscreenTab(DOMString startUrl,
                                DOMString id,
                                CaptureOptions options,
                                GetTabMediaCallback callback);
```

## Open questions

For later discussion, if we decide to move forward with promoting to stable-channel:

1. ~~Is it at all useful to ever provide this as a public API?  Meaning, do we wish to support the kinds of use cases not handled by the web platform (W3C Presentation API)?~~ *Yes; now proposed.*
2. ~~Should extensions be restricted to using a "safe" subset of extension APIs when using this API, to prevent leaking user information or browsing activity?~~ *Conclusion: We're fine.*
3. ~~Should the navigation of the off-screen tab be constrained only to sites having a specific origin?~~ *No, because the off-screen tab does not have access to the normal user profile and so cannot access private resources.*