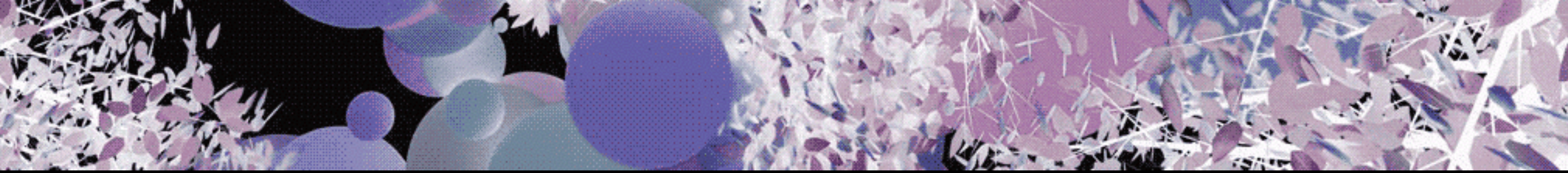


EXTENSION DEVELOPMENT FOR CHROME

PREREQUISITES

- Terminal
- HTML / CSS
- JavaScript
- Problem-solving *

WHAT IS AN EXTENSION?

At the basic level, a Chrome extension is just some HTML, CSS and JavaScript that allows you to add functionality to Chrome via the JavaScript APIs it exposes.

WHAT IS AN EXTENSION?

Another way to think about it: an extension is as a web page hosted within Chrome that can access some additional APIs.

BROWSER ACTIONS

One basic type of Chrome extension is called a *Browser Action*. This kind of extension add a button to the Chrome toolbar that will show an HTML page when clicked and optionally execute some JavaScript.

BASIC EXTENSION FUNCTIONALITY

- Add a button
- Modify pages
- Request resources
- Do stuff in background

HACKING THE BROWSER

One basic type of Chrome extension task is the *Browser Action*. An example is adding a button to the Chrome toolbar. When clicked, the button will show an HTML/CSS page and optionally execute some JavaScript.

CONTENT-SCRIPTS

Another common part of an extension is called a *Content-Script*. This refers to a bit of JavaScript that is *injected* into some (or all) pages that Chrome loads

BACKGROUND PAGE

Another common part of an extension is called the *background page*. This refers to a webpage (usually with JavaScript) that is running in the background, invisible to the user

DOCUMENTATION

https://developer.chrome.com/extensions

OUR FIRST EXTENSION

Lets dive in! For this first example we will simply add a button to the toolbar that loads some info about the current page. To start, we need a folder that will hold our extension resources. Lets call it 'Example1'…

MANIFEST.JSON

Every Chrome extensions require a manifest file. The manifest tells Chrome everything it needs to know to properly load the extension.

So lets create a manifest.json file in the folder we created. We can leave it blank for now.

EXTENSION ICON

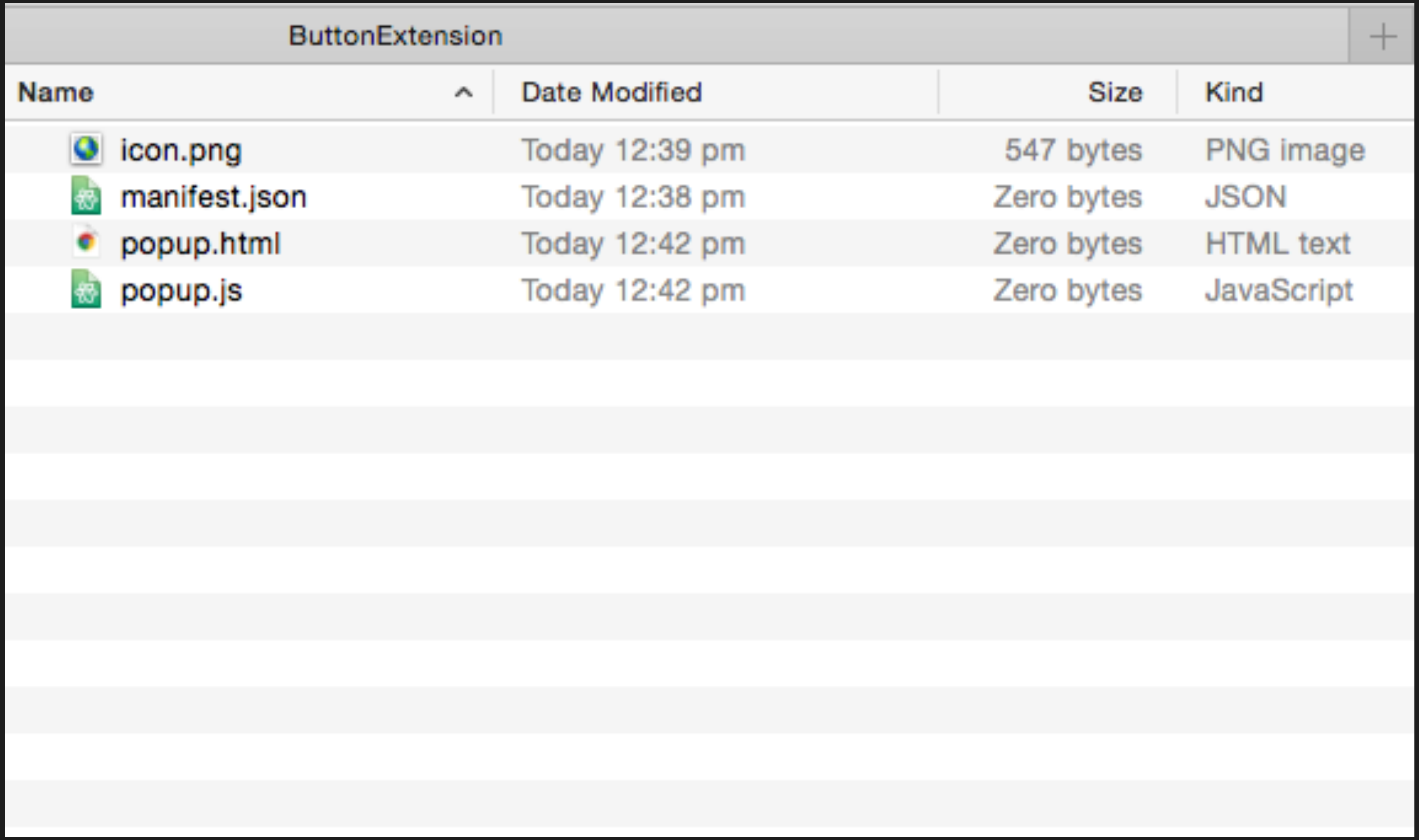
Next lets grab a simple icon for our extension

This should be 19x19 .png file called icon.png. If you don't have one, just do a web search for "19 x19 icon" and pick one to test with.

POPUP HTML

Next we’ll need an HTML page to show when our icon is pressed, so lets add a popup.html file and a popup.js file in our folder.

So far, our ButtonExtension folder should have four files, like this:



ButtonExtension

Name	Date Modified	Size	Kind
icon.png	Today 12:39 pm	547 bytes	PNG image
manifest.json	Today 12:38 pm	Zero bytes	JSON
popup.html	Today 12:42 pm	Zero bytes	HTML text
popup.js	Today 12:42 pm	Zero bytes	JavaScript

Now lets add some code.
You can do find all the code
for this workshop at:

https://github.com/dhowe/BrowserHacking

To grab them with git, open
your terminal and do:

```
$ git clone git@github.com:dhowe/BrowserHacking.git
```

Now open up manifest.json and enter the following code:

```
{
  "manifest_version": 2,

  "name": “SimpleButton",
  "description": “Adds a button to chrome",
  "version": "1.0",

  "browser_action": {
   "default_icon": "icon.png",
   "default_popup": "popup.html"
  },
  "permissions": [
   "activeTab"
  ]
}
```

PERMISSIONS

Note the *permissions* section where we ask to access the *activeTab*. This is required in order to enable us to get the URL of the current tab to pass on to our code.

Many of the APIs Chrome APIs require us to specify whatever permissions you require in the manifest.

```
{
  ...

  "permissions": [
   "activeTab"
   ]
}
```

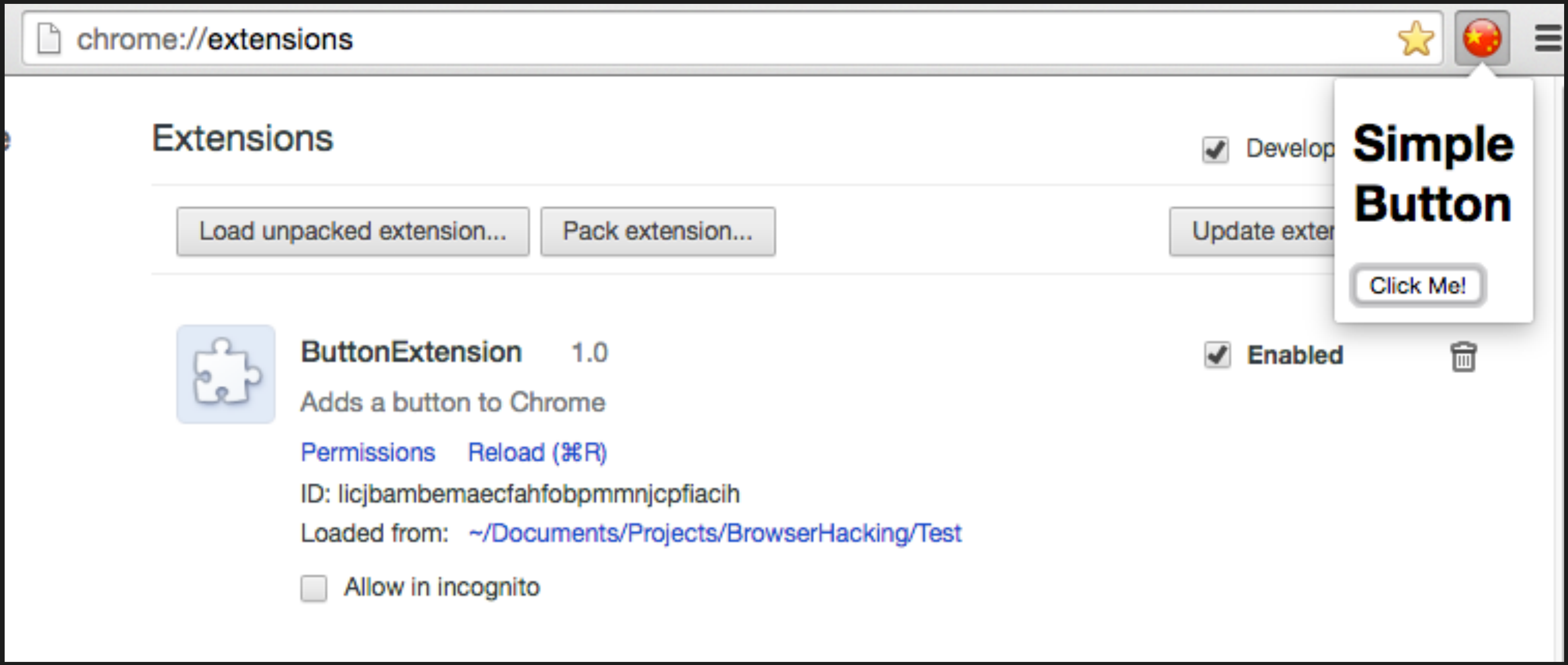
Now lets add some very simple code to our popup.html

```html
<!doctype html>

<html>

  <head>
    <title>Great Firewall Check</title>
    <script src="popup.js"></script>
  </head>

  <body>
    <h1>Great Firewall Check</h1>
    <button id="button1">Check this page now!</button>
  </body>

</html>
```

Note that we include our popup.js file

```
<!doctype html>

<html>

  <head>
    <title>Great Firewall Check</title>

    <script src="popup.js"></script>
  </head>

  <body>
    <h1>Great Firewall Check</h1>
    <button id="button1">Check this page now!</button>
  </body>

</html>
```

Ok, lets test what we have so far …



chrome://extensions

Extensions

✓ Develop

Load unpacked extension... Pack extension... Update exte

Simple Button

Click Me!

ButtonExtension 1.0

Adds a button to Chrome

Permissions Reload (⌘R)

ID: licjbambemaecfahfobpmmnjcpfiacih

Loaded from: ~/Documents/Projects/BrowserHacking/Test

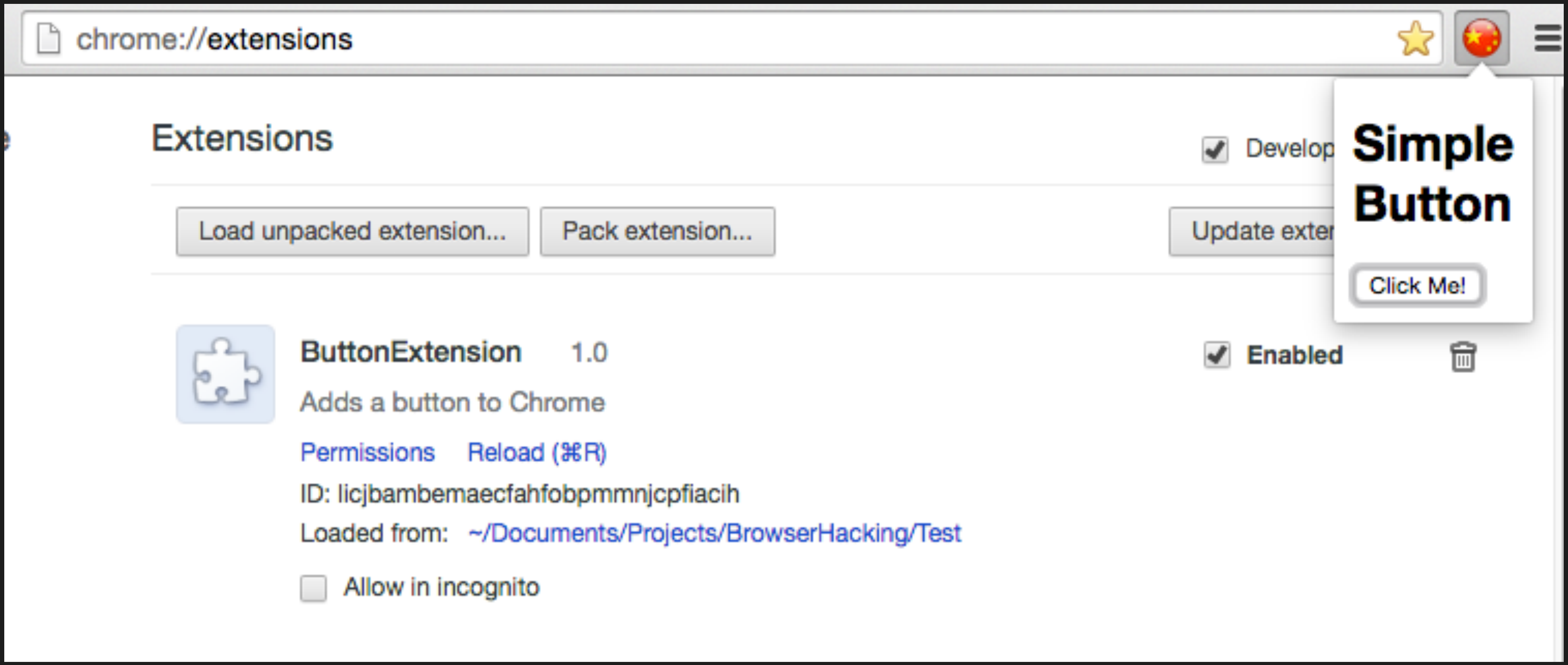
☐ Allow in incognito

✓ Enabled

Now for popup.js …

```
// called when our popup is loaded

document.addEventListener('DOMContentLoaded', function () {

    console.log("Popup loaded!");
});
```

Ok, lets test what we have so far …



chrome://extensions

Extensions

Develop

Load unpacked extension... Pack extension... Update exter

Simple Button

Click Me!

ButtonExtension 1.0

Adds a button to Chrome

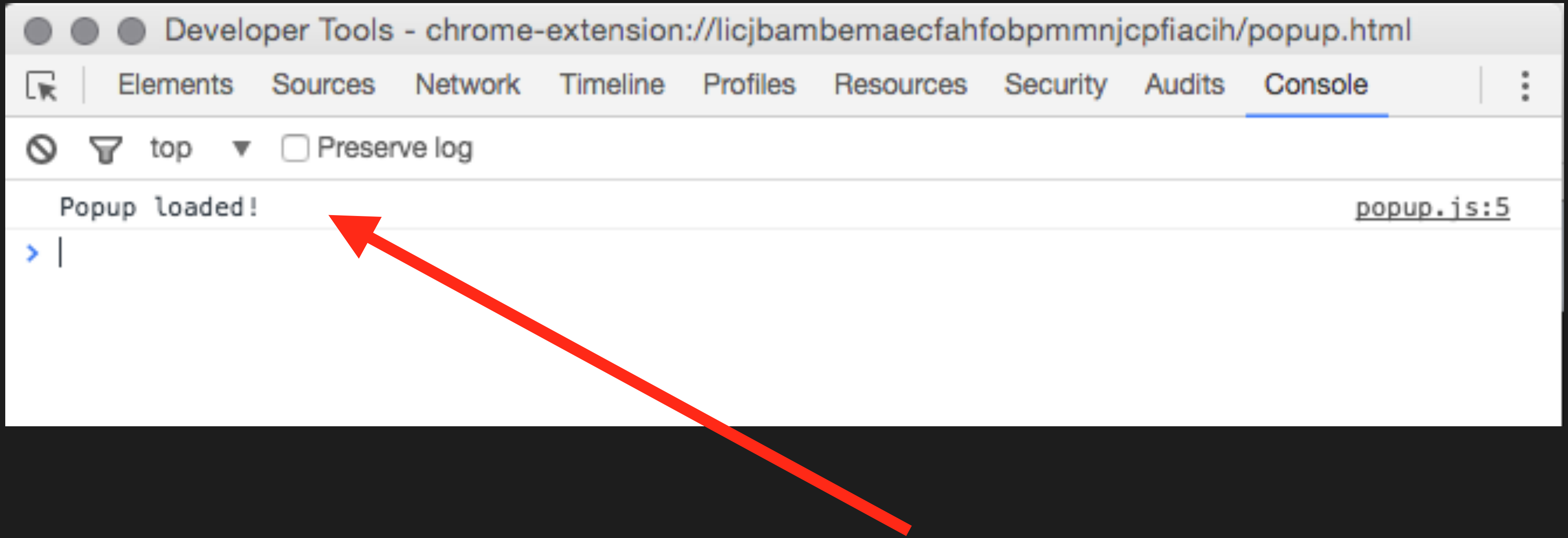
Permissions Reload (⌘R)

ID: licjbambemaecfahfobpmmnjcpfiacih

Loaded from: ~/Documents/Projects/BrowserHacking/Test

Allow in incognito

Enabled

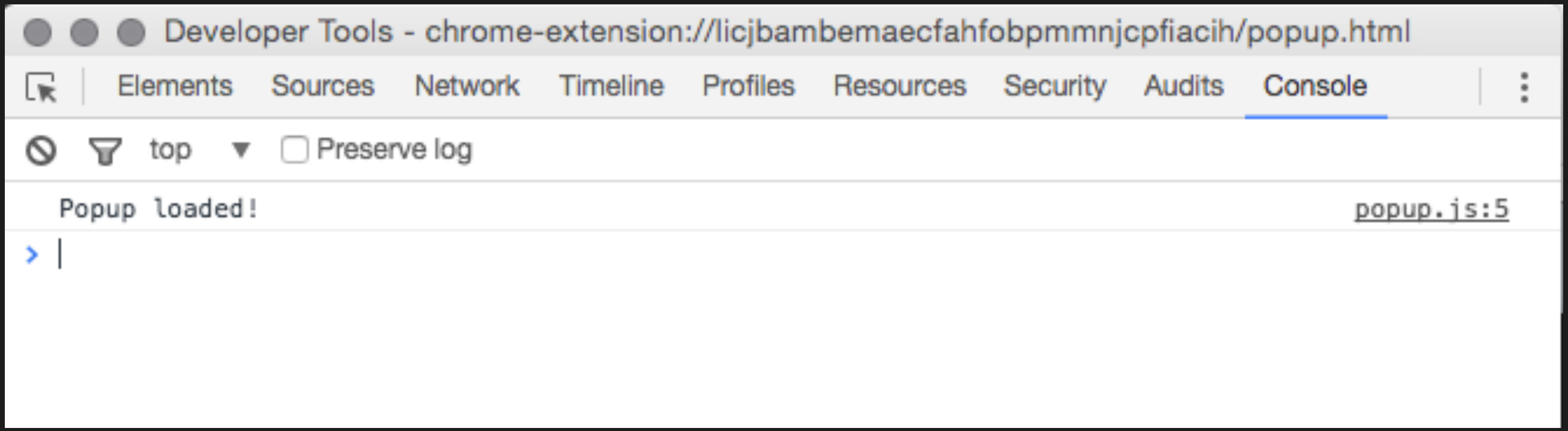


Developer Tools - chrome-extension://licjbambemaecfahfobpmmnjcpfiacih/popup.html

Elements Sources Network Timeline Profiles Resources Security Audits Console

top ▼ Preserve log

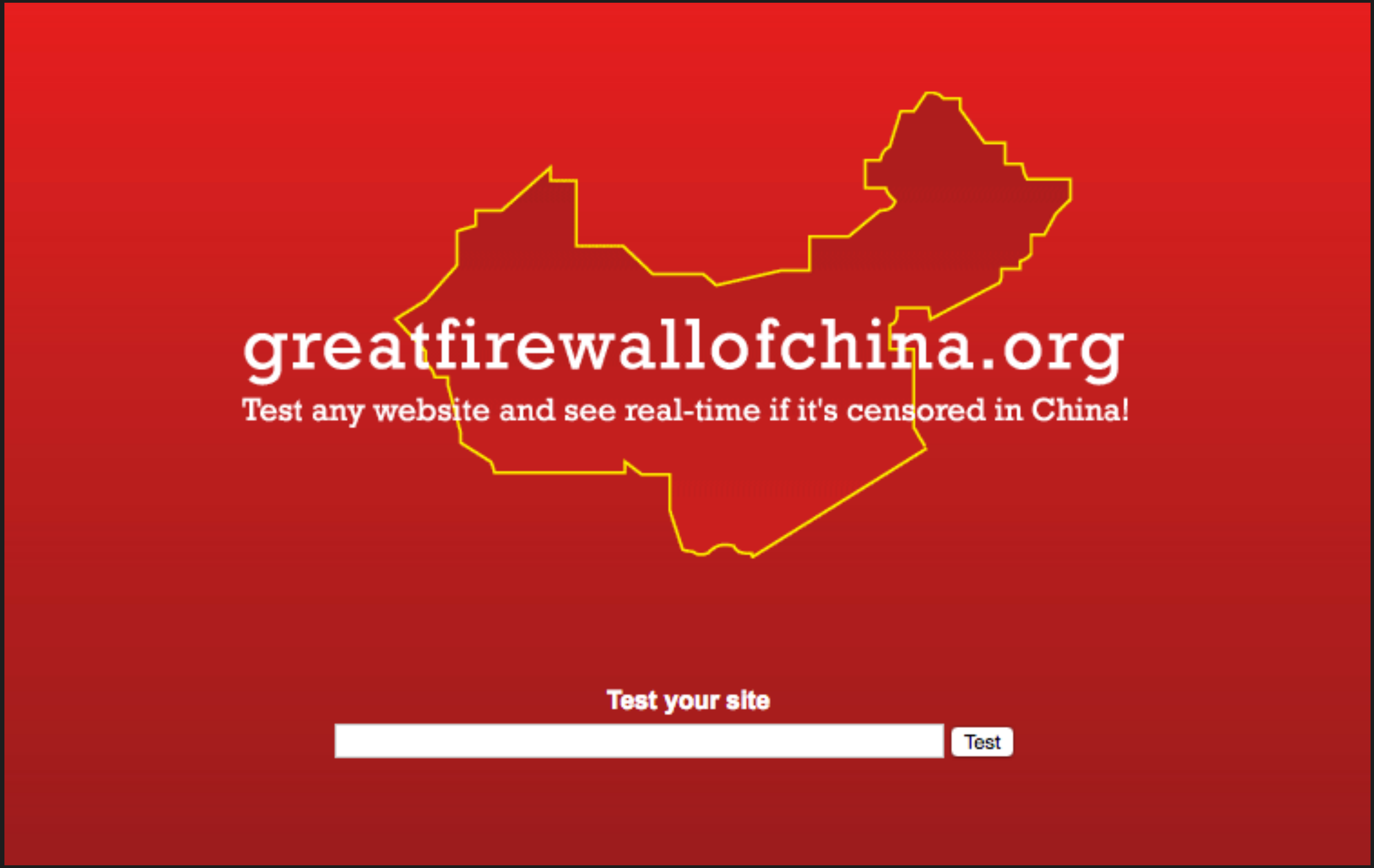
```
Popup loaded!                                                popup.js:5
>
```



```
Popup loaded!                                                  popup.js:5
>
```

Use the Browser’s built-in tools

Now let's do something useful

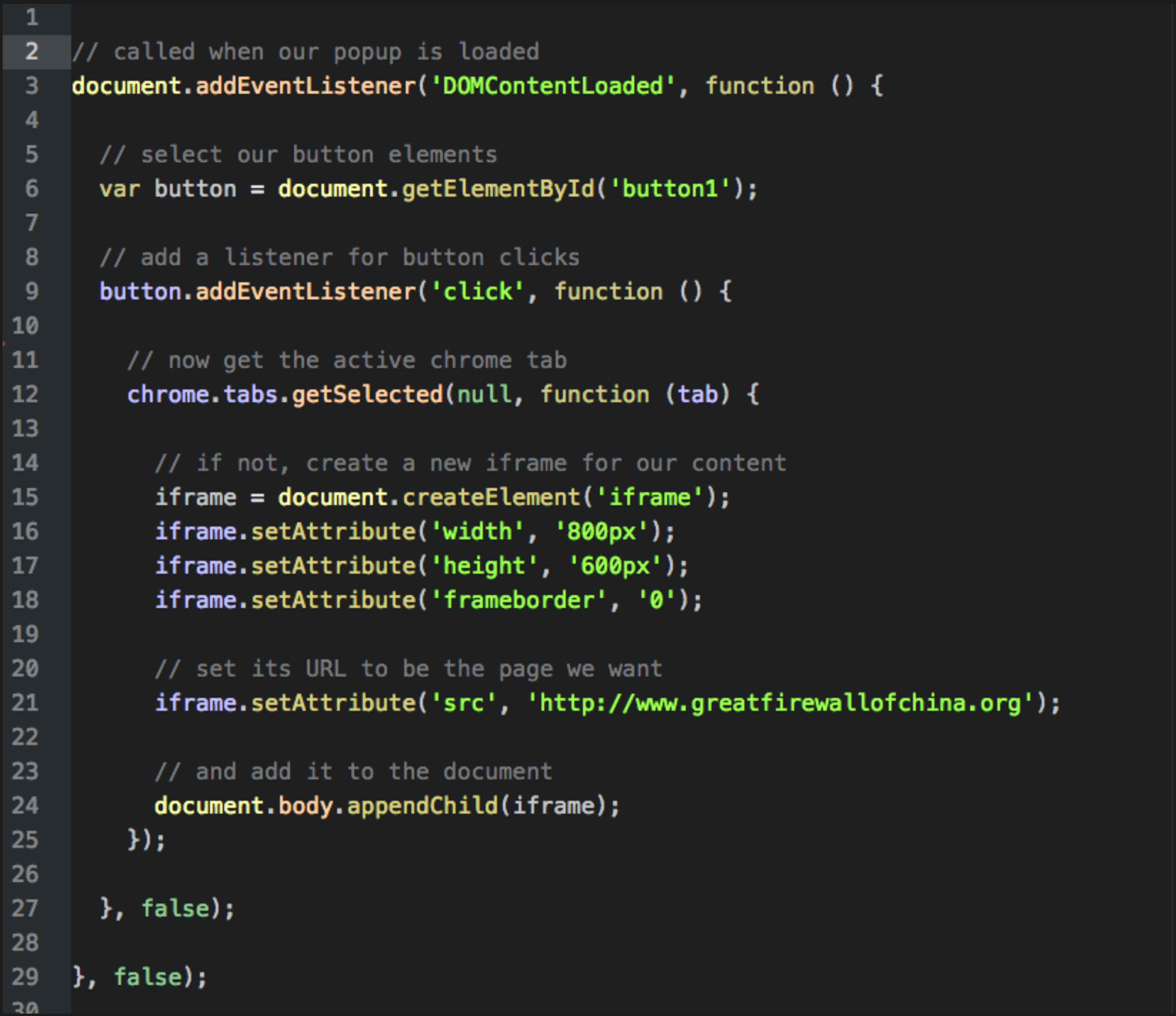

greatfirewallofchina.org

Test any website and see real-time if it's censored in China!

Now back to popup.js …

When we click our button, lets load http://www.greatfirewallofchina.org/ into our popup window…

```
See code in BrowserHacking::Example1
```



```javascript

// called when our popup is loaded
document.addEventListener('DOMContentLoaded', function () {

  // select our button elements
  var button = document.getElementById('button1');

  // add a listener for button clicks
  button.addEventListener('click', function () {

    // now get the active chrome tab
    chrome.tabs.getSelected(null, function (tab) {

      // if not, create a new iframe for our content
      iframe = document.createElement('iframe');
      iframe.setAttribute('width', '800px');
      iframe.setAttribute('height', '600px');
      iframe.setAttribute('frameborder', '0');

      // set its URL to be the page we want
      iframe.setAttribute('src', 'http://www.greatfirewallofchina.org');

      // and add it to the document
      document.body.appendChild(iframe);
    });

  }, false);

}, false);
```

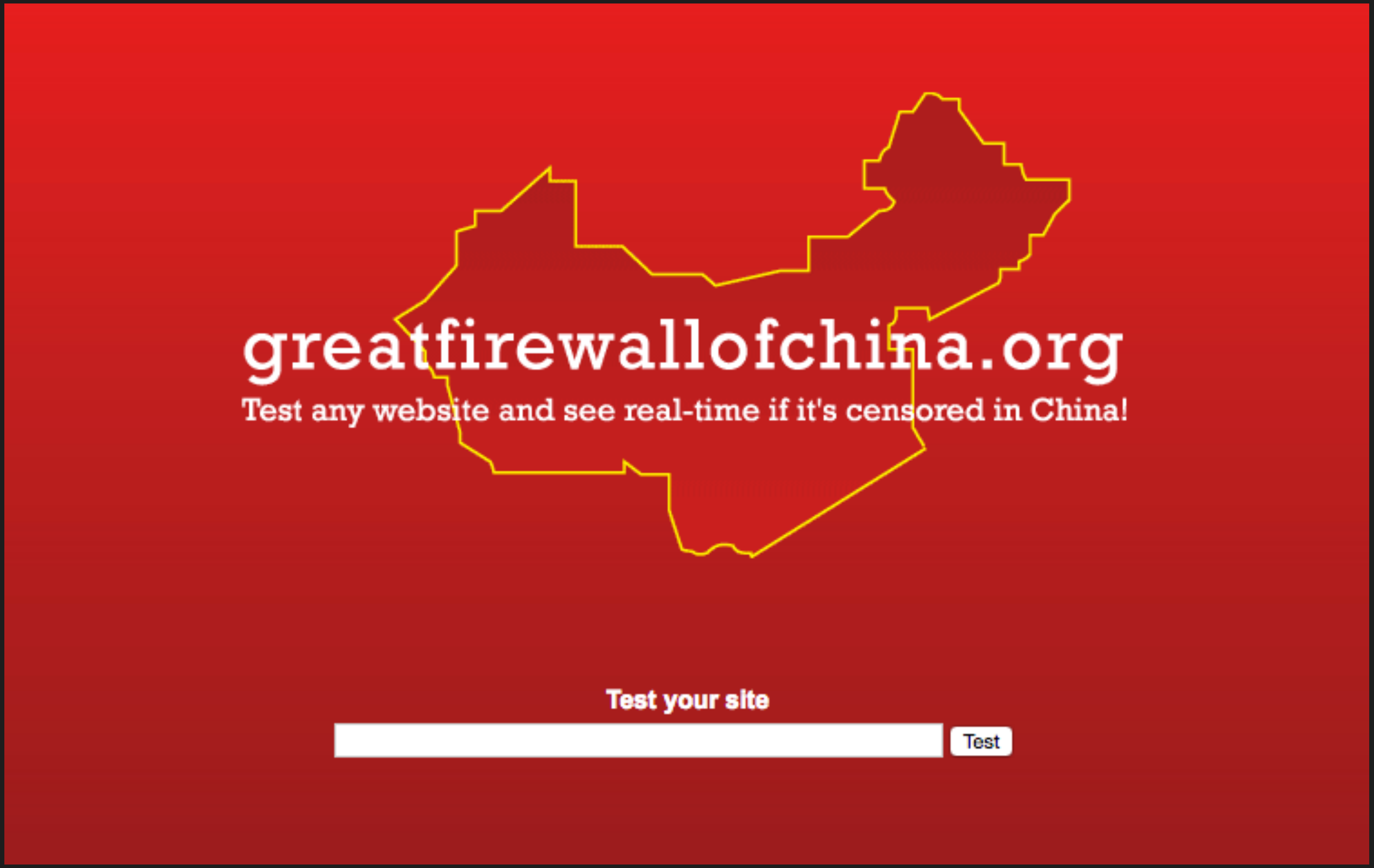
So this is OK, but what we really want is to automatically check *the current page*…

We can do this via the chrome.tabs API

```
chrome.tabs.getSelected(null, function(tab) {

    console.log(tab.url);
});
```

https://developer.chrome.com/extensions/tabs

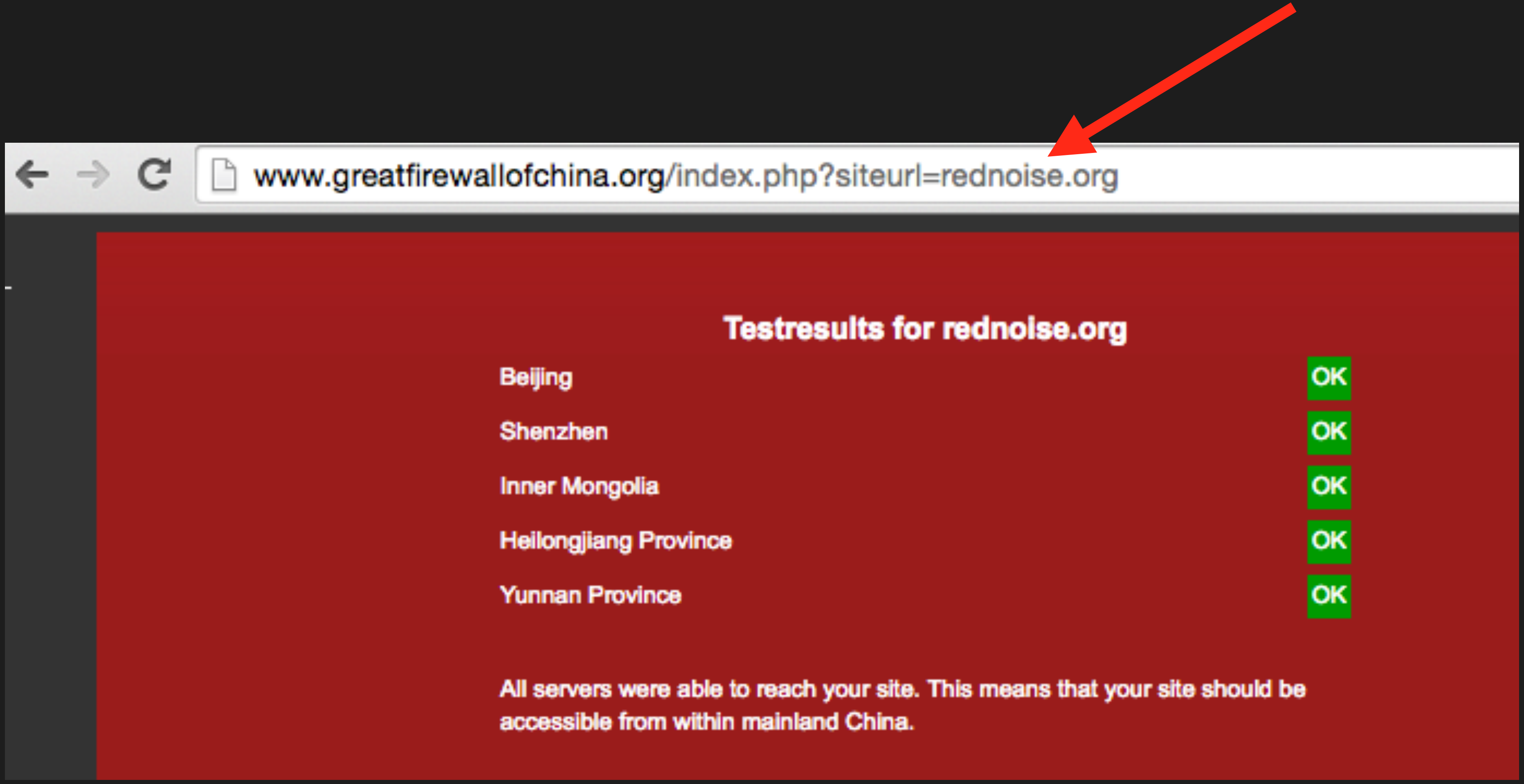
Now that we have the current tab's URL, lets take a look at how the site works…


greatfirewallofchina.org

Test any website and see real-time if it's censored in China!

Test your site

Now that we have the current tab's URL, lets take a look at how the site works…



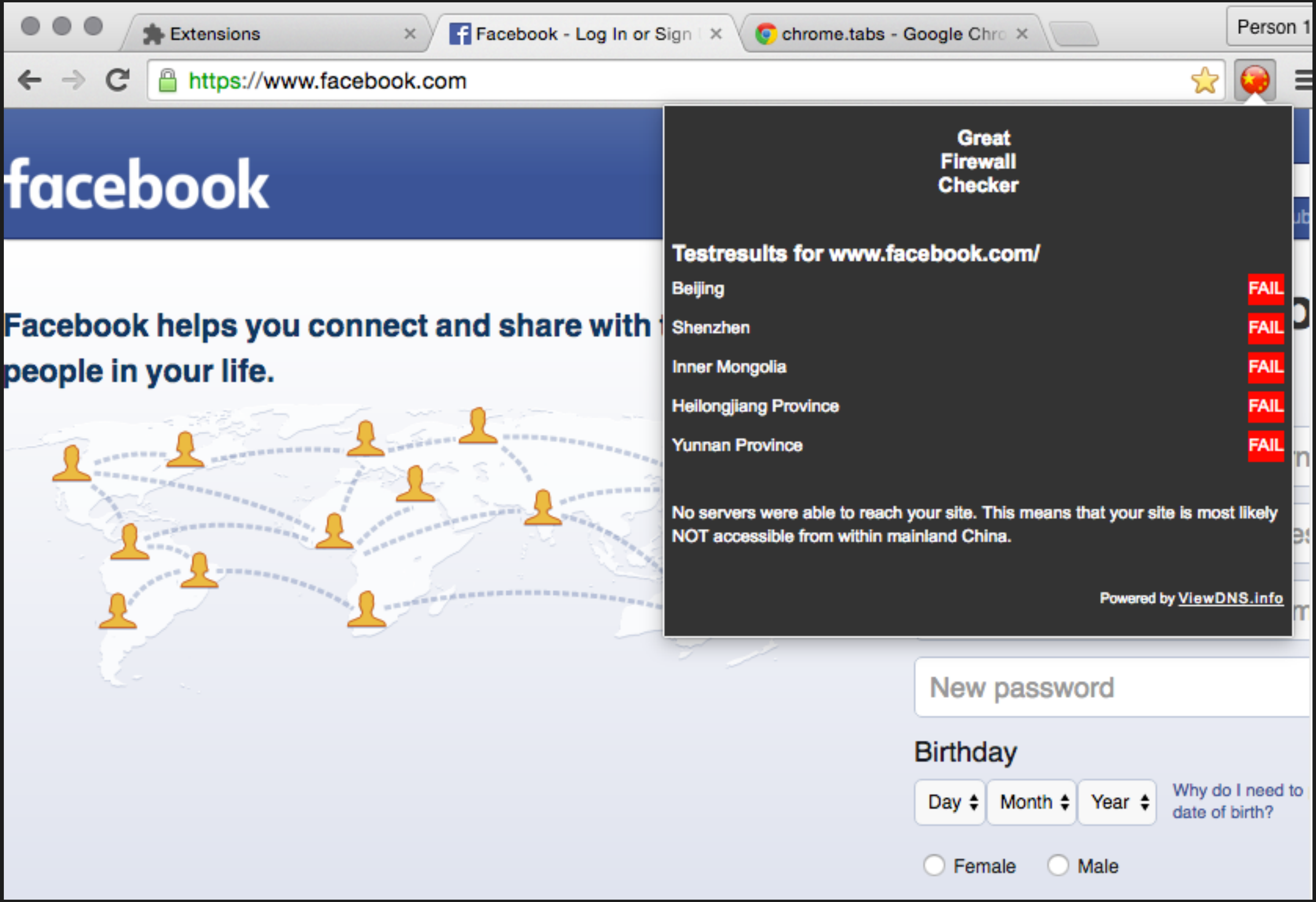
www.greatfirewallofchina.org/index.php?siteurl=rednoise.org

Testresults for rednoise.org

Beijing	OK
Shenzhen	OK
Inner Mongolia	OK
Heilongjiang Province	OK
Yunnan Province	OK

All servers were able to reach your site. This means that your site should be accessible from within mainland China.

You can see this all together in *Example 2+3*



Great
Firewall
Checker

Testresults for www.facebook.com/

Beijing	FAIL
Shenzhen	FAIL
Inner Mongolia	FAIL
Heilongjiang Province	FAIL
Yunnan Province	FAIL

No servers were able to reach your site. This means that your site is most likely NOT accessible from within mainland China.

Powered by ViewDNS.info

FIN
PARTIE I

next: content-scripts

CONTENT-SCRIPTS

Earlier I mentioned a common extension task was to inject code into some (or all) web pages that the user visits.
We can do this with *Content-Scripts*.

CONTENT-SCRIPTS

A content script is simply a JavaScript file that runs in the context of a web page. This means that a content script can interact with web pages that the browser visits. Not every JavaScript file in a Chrome extension can do this; we'll see why later…

CONTENT-SCRIPTS

Lets add a content-script to our example called content.js

And we need to list it in our manifest

```
"content_scripts": [
  {
    "matches": [
      "<all_urls>"
    ],
    "js": ["content.js"]
  }
]
```

Example4

```
"content_scripts": [
  {
    "matches": [
      "<all_urls>"
    ],
    "js": ["content.js"]
  }
]
```

This tells Chrome to inject content.js into every page we visit using the special <all_urls> URL pattern.

Note: if we want to inject the script on only some pages, we can use match patterns. Here are a few examples of values for "matches":

["https://mail.google.com/*", “http://mail.google.com/*"]
injects our script into HTTPS and HTTP Gmail.

If we have / at the end instead of /*, it matches the URLs exactly, and so would only inject into https://mail.google.com/, not https://mail.google.com/mail/u/0/#inbox. Usually that isn’t what you want.

CONTENT-SCRIPTS

so lets add some test code to content.js

```
// content.js

alert("Hello from: “+window.location.href);
```

Example4

CONTENT-SCRIPTS

Sometimes its useful to inject jQuery into the page:

```
"content_scripts": [
  {
    "matches": [
      "<all_urls>"
    ],
    "js": ["jquery-2.2.3.js","content.js"]
  }
]
```

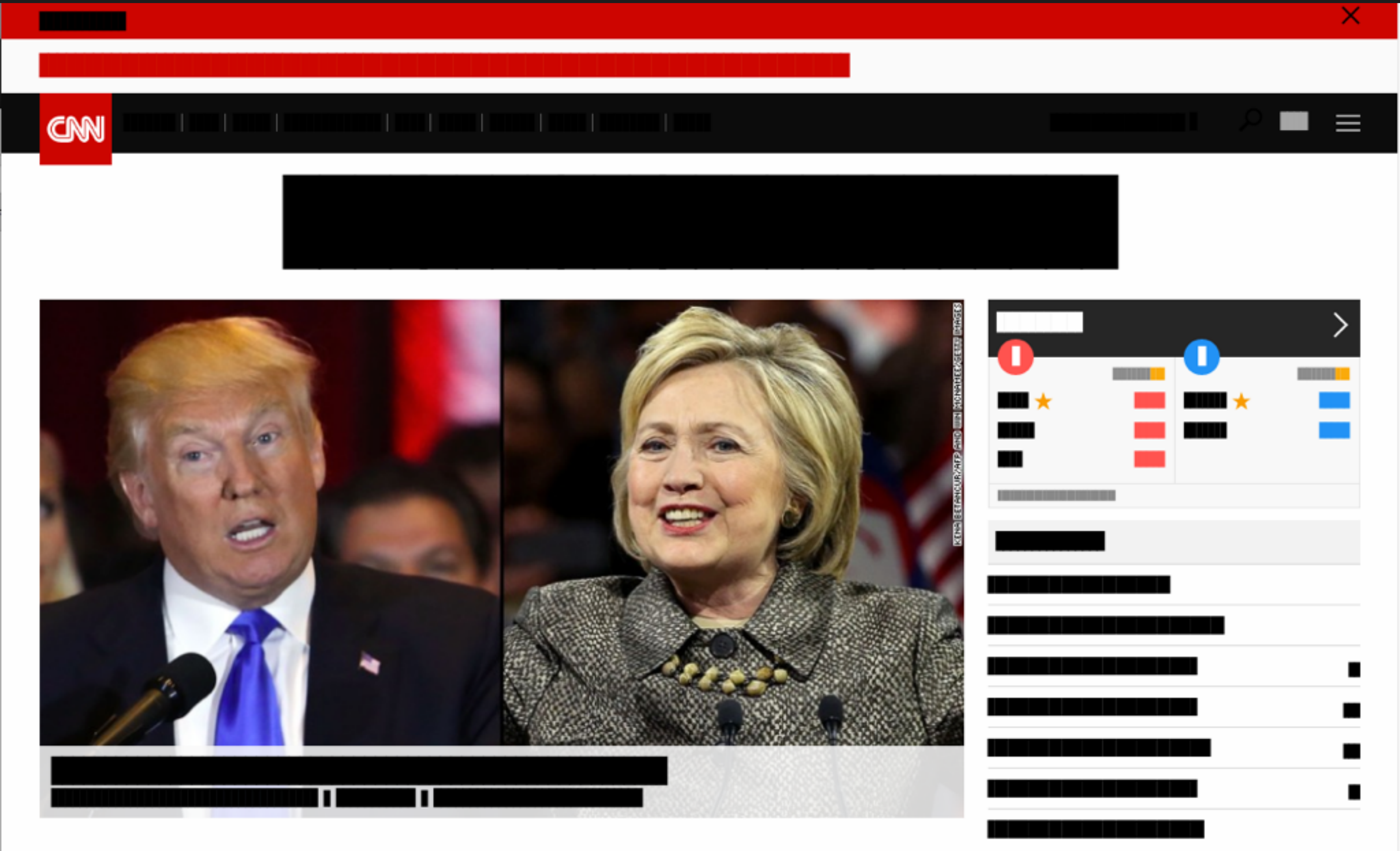
Example4

So now that we have jQuery, lets add some simple code in content.js to grab the first link on each page and print it to the console.

```
// content.js

var firstHref = $("a[href^='http']").eq(0).attr("href");

console.log(firstHref);
```

Example4

Ok, now lets try something more useful…



[redacted]

[redacted]

[redacted]

[redacted]

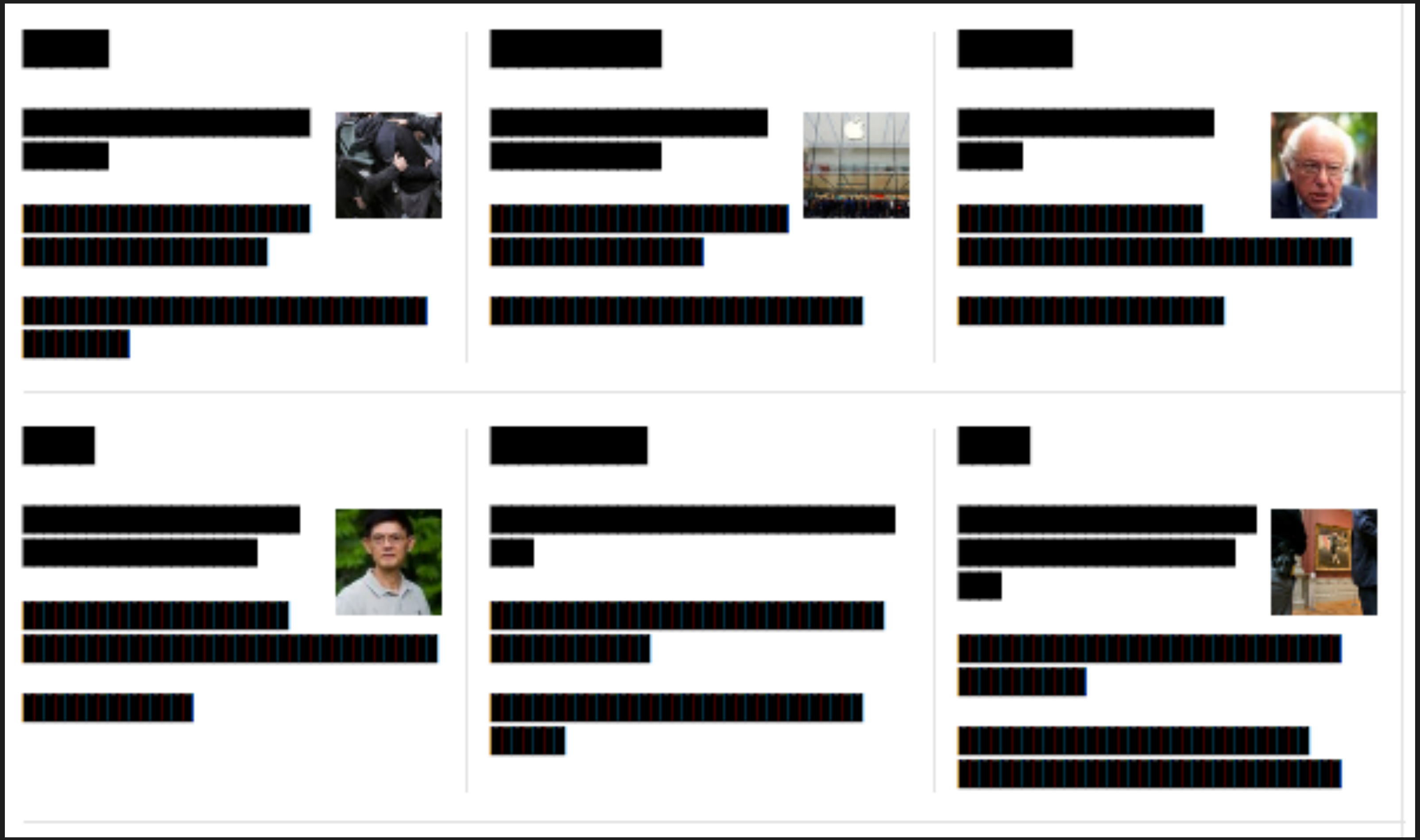
[redacted]

[redacted]

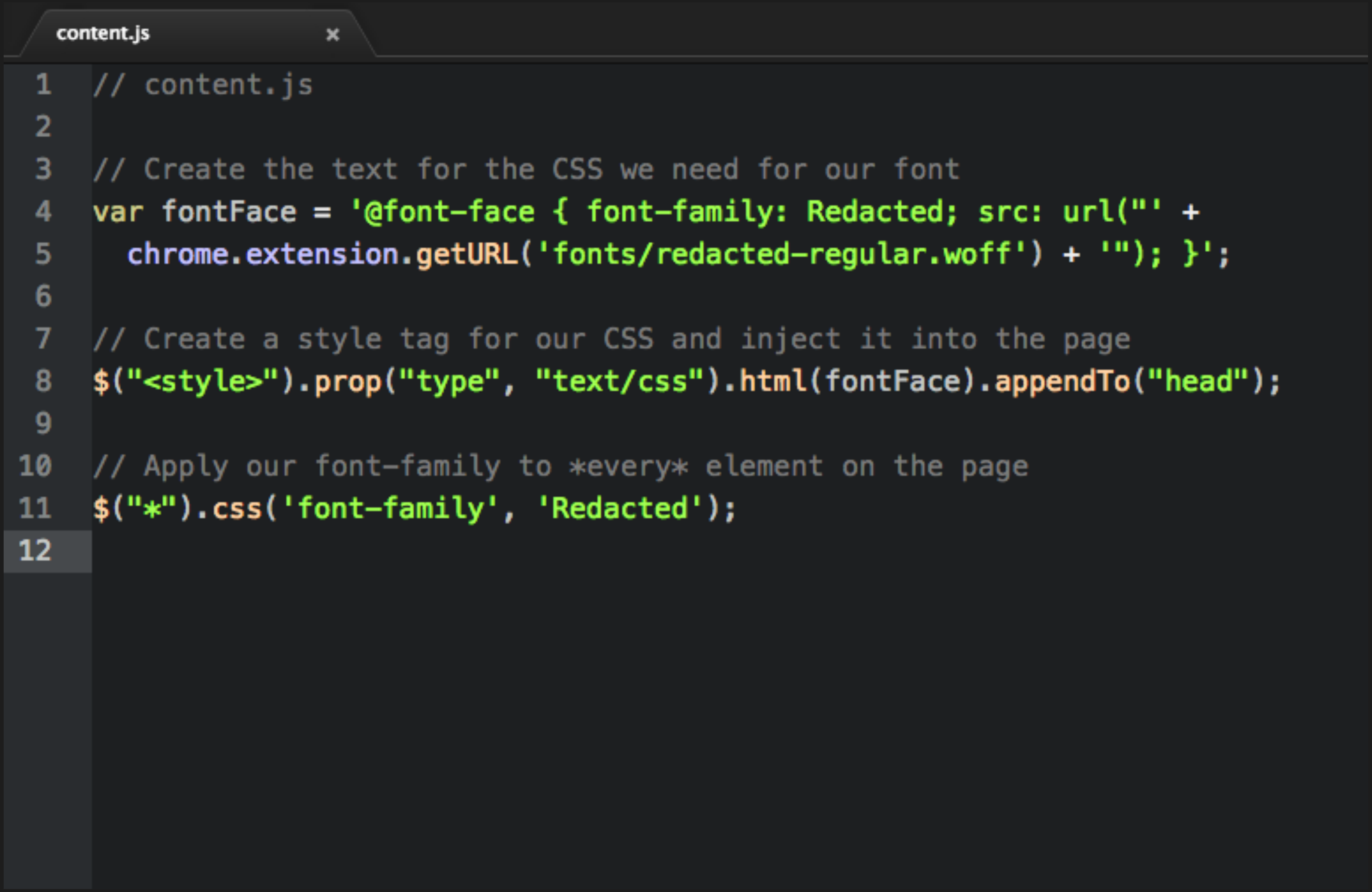
[redacted]

STEP-BY-STEP

- Download a *.woff* font file into a new fonts folder
- Use our content-script to *inject* its CSS into the page
- Then *apply* the font to our elements (jQuery)



STEP-BY-STEP

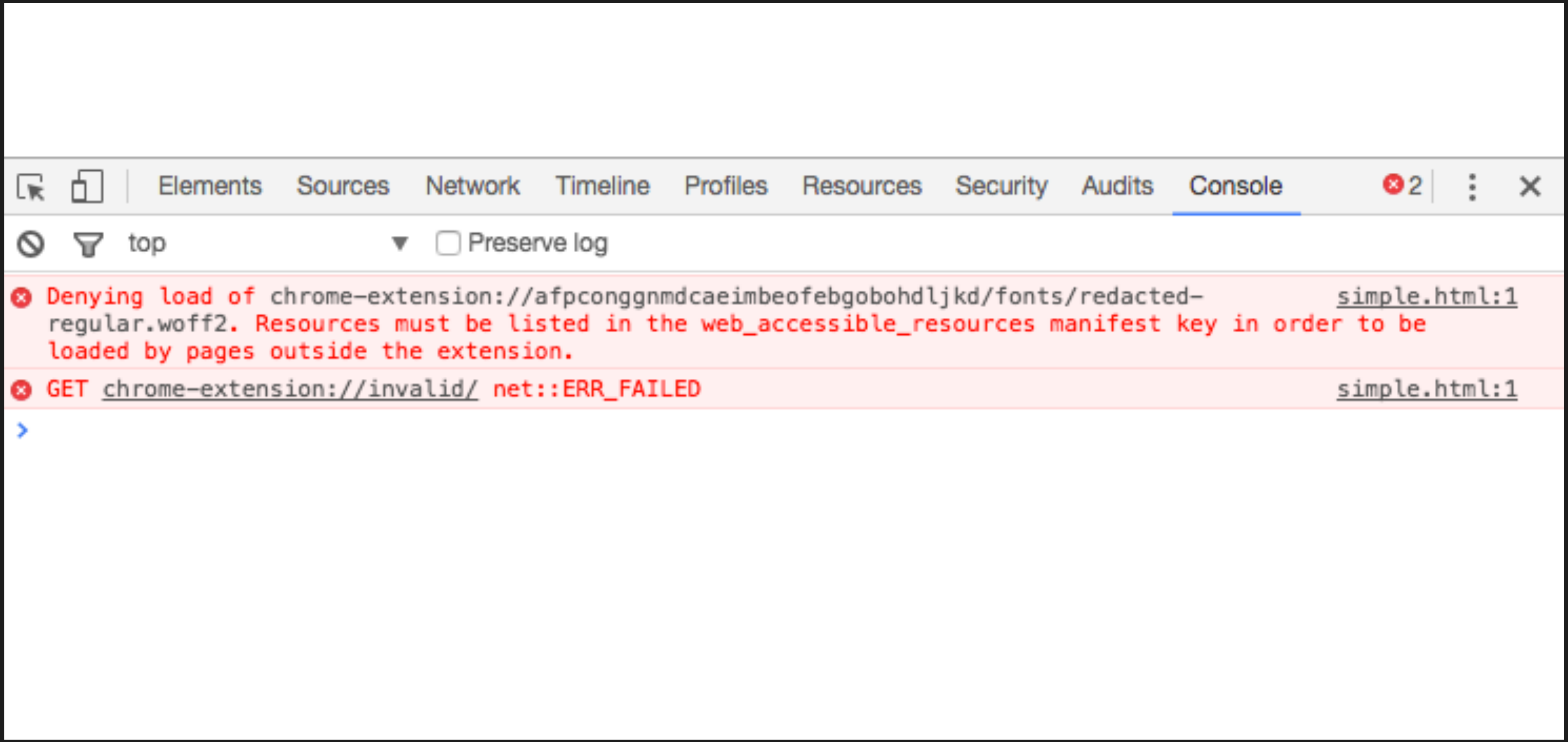


content.js ×

```
// content.js

// Create the text for the CSS we need for our font
var fontFace = '@font-face { font-family: Redacted; src: url("' +
  chrome.extension.getURL('fonts/redacted-regular.woff') + '"); }';

// Create a style tag for our CSS and inject it into the page
$("<style>").prop("type", "text/css").html(fontFace).appendTo("head");

// Apply our font-family to *every* element on the page
$("*").css('font-family', 'Redacted');

```

STEP-BY-STEP

But wait, we have a problem…



```
Denying load of chrome-extension://afpconggnmdcaeimbeofebgobohdljkd/fonts/redacted-        simple.html:1
regular.woff2. Resources must be listed in the web_accessible_resources manifest key in order to be
loaded by pages outside the extension.
GET chrome-extension://invalid/ net::ERR_FAILED                                           simple.html:1
>
```

CONTENT-SCRIPTS

We need to list the fonts in our manifest

```
"web_accessible_resources": [
  "fonts/*"
],
```

Example5

STEP-BY-STEP

Note: we can also include a CSS file to be loaded with our content-script

```
"content_scripts": [{
  "matches": [ "<all_urls>" ],
  "js": [ "jquery-2.2.3.js", "content.js" ],
  "css": [ "content.css" ]
}],
```

STEP-BY-STEP

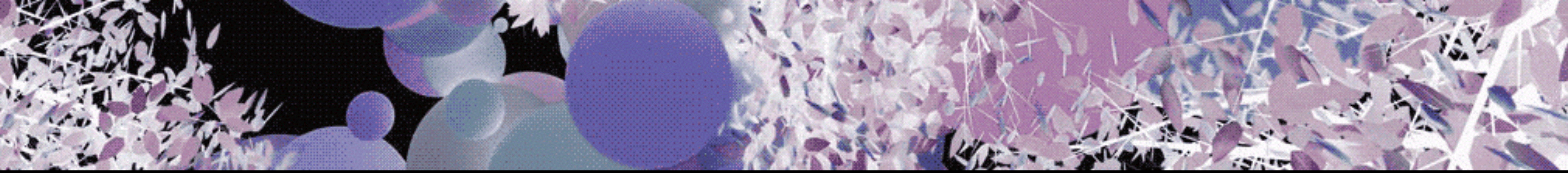
Some CSS code for contents.css

```
img,image {
    -webkit-filter: brightness(0);
}
```

Now this is the NEWS…

South China Morning Post 南華早報

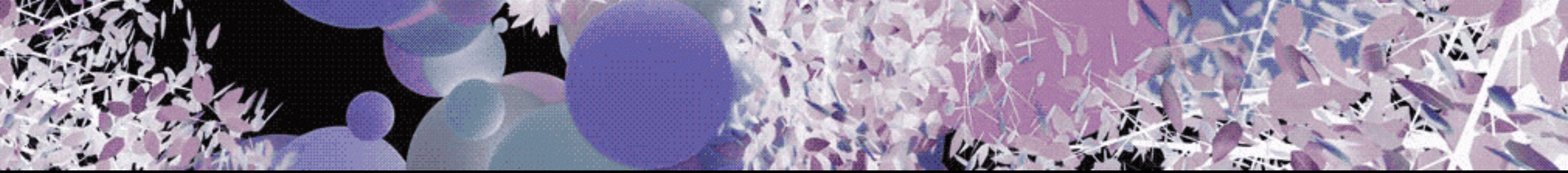
[redacted]



FIN

PARTIE II

next: adblockers



TRY AGAIN
FAIL AGAIN
FAIL BETTER

-Samuel Beckett