Writing Secure Chrome Apps and Extensions

Keeping your users safe

Jorge Lucángeli Obes
Software Engineer

Google 12 I/O

Keeping users safe

- A lot of work going into making browsers more secure
- What about users' data?

... means dealing with untrusted content safely

- New platform and browser features designed to handle untrusted content
- **Security features still not in widespread use!**

Focus on the client side

- Showcase security features all client code can leverage
- Chrome platform enables rich but secure apps, even offline

Web platform

- Important security features:
  - Content-Security-Policy (CSP)
  - HTML5 sandboxed iframes

Same-origin policy

- Objects can interact directly if and only if they belong to the same web (security) origin
- schema://host:port

Common vulnerabilities

- Untrusted content/script gaining authority of an origin: XSS
- Origin gaining network access privileges of another origin: CSRF

Extensions | ManifestLint (unsafe)

ManifestLint (unsafe)

Original manifest


```
{
  "name": "ManifestLint (unsafe)",
  "description": "",
  "version": "1",
  "app": {
    "launch": {
      "local_path": "main.html"
    }
  },
  "manifest_version": 1
}
```


Lint Clear

- Original manifest [x]
- Manifest 0 [x]

Original manifest


```
{
  name: ManifestLint (unsafe),
  description: ,
  version: 1,
  app: {
    launch: {
      local_path: main.html
    }
  },
  manifest_version: 1
}
```


No Content Security Policy setting

ManifestLint (unsafe)

ManifestLint (unsafe)

Original manifest

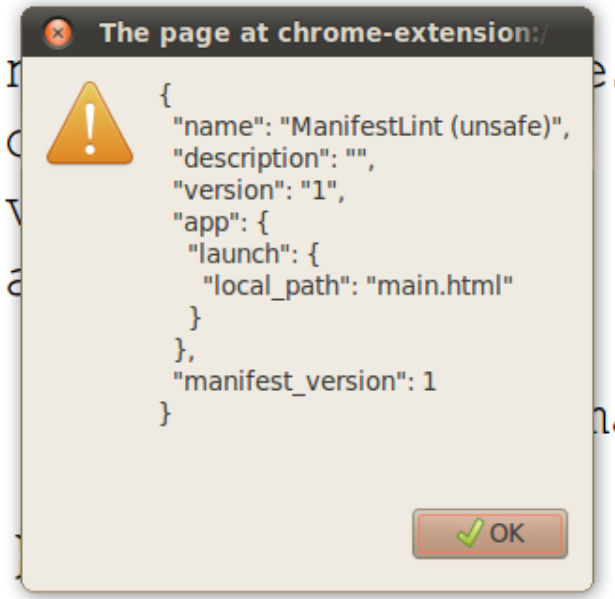
```
{
  "name": "Safe manifest =P",
  "description": "</span><img
onmouseover='alert(localStorage.getItem(localStor
age.key(0)))' width=20px height=20px><span>",
  "version": "1",
  "app": {
    "launch": {
      "local_path": "main.html"
    }
  },
  "manifest_version": 2
}
```

Lint Clear

- Original manifest [x]
- Original manifest [x]
- Manifest 0 [x]

Original manifest

```
{
  n                                    est =P,
  c
  V
  a
                                       ain.html
  manifest_version: 2
}
```



No Content Security Policy setting

What's the problem?

main.html

JS

```
<script>
  // ...
  var rawJson = $('#contentta')[0].value;
  // ...
  var parsedJson = eval('(' + rawJson + ')');
  var pp = prettyPrint(parsedJson);
  $('#contentdiv')[0].innerHTML = pp;
  // ...
```

Content Security Policy (CSP)

- CSP allows whitelisting trusted scripts/resources
- Prevents most XSS attacks

Separating scripts from content

- The browser needs to know the origin of scripts/resources
- The only way to do this is to forbid inline script and `eval()`

How do we use CSP?

manifest.json

JSON

```
{
  "name": "ManifestLint",
  "version": "1",
  "app": { "launch": {
      "local_path": "main.html"
    } },
"content_security_policy": "default-src 'self';
     script-src 'self' https://ajax.googleapis.com;
     style-src 'self' 'unsafe-inline'"
}
```

Nothing works!

Refused to load script from 'http://code.jquery.com/jquery-latest.js' because of Content-Security-Policy.

Refused to execute inline script because of Content-Security-Policy.

Refused to execute inline event handler because of Content-Security-Policy.

Need to:

- Load external scripts over HTTPS
- Extract own scripts to `.js` files
- Remove inline event handlers

CSP-ready extension

main.html

HTML

```
<!DOCTYPE html>
<html>
<head><title>ManifestLint</title>
<script src="https://ajax.googleapis.com/.../jquery.min.
js"
 type="text/javascript"></script>
<script src="main.js" type="text/javascript"></script>
</head>
<body>
...
```

CSP-ready extension

main.js

JS

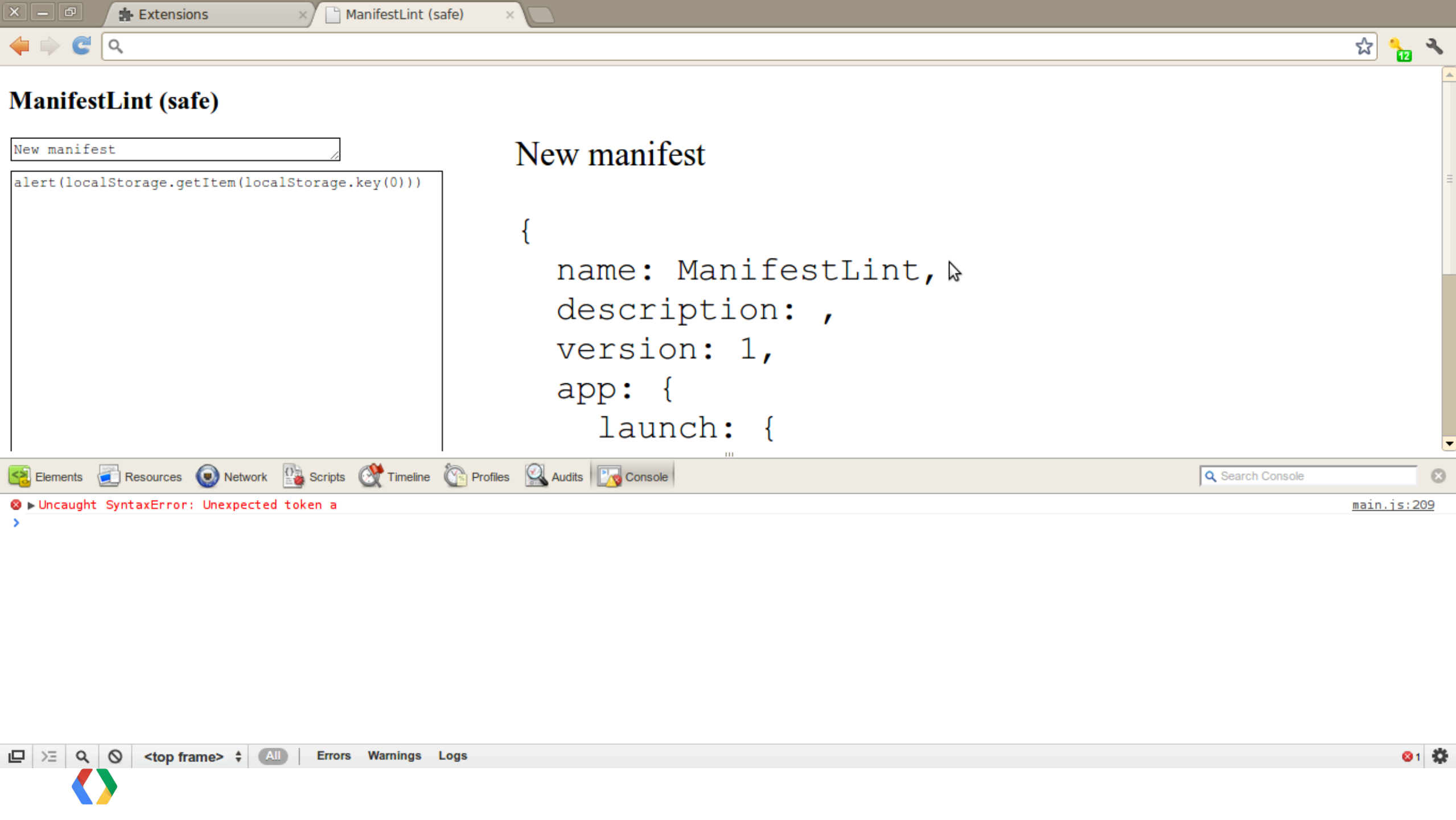
```
function lint() {
// ...
}

$(document).ready(function() {
  // Set up event handling.
  $("#lint").click(lint);
});
```

CSP-ready extension

main.js

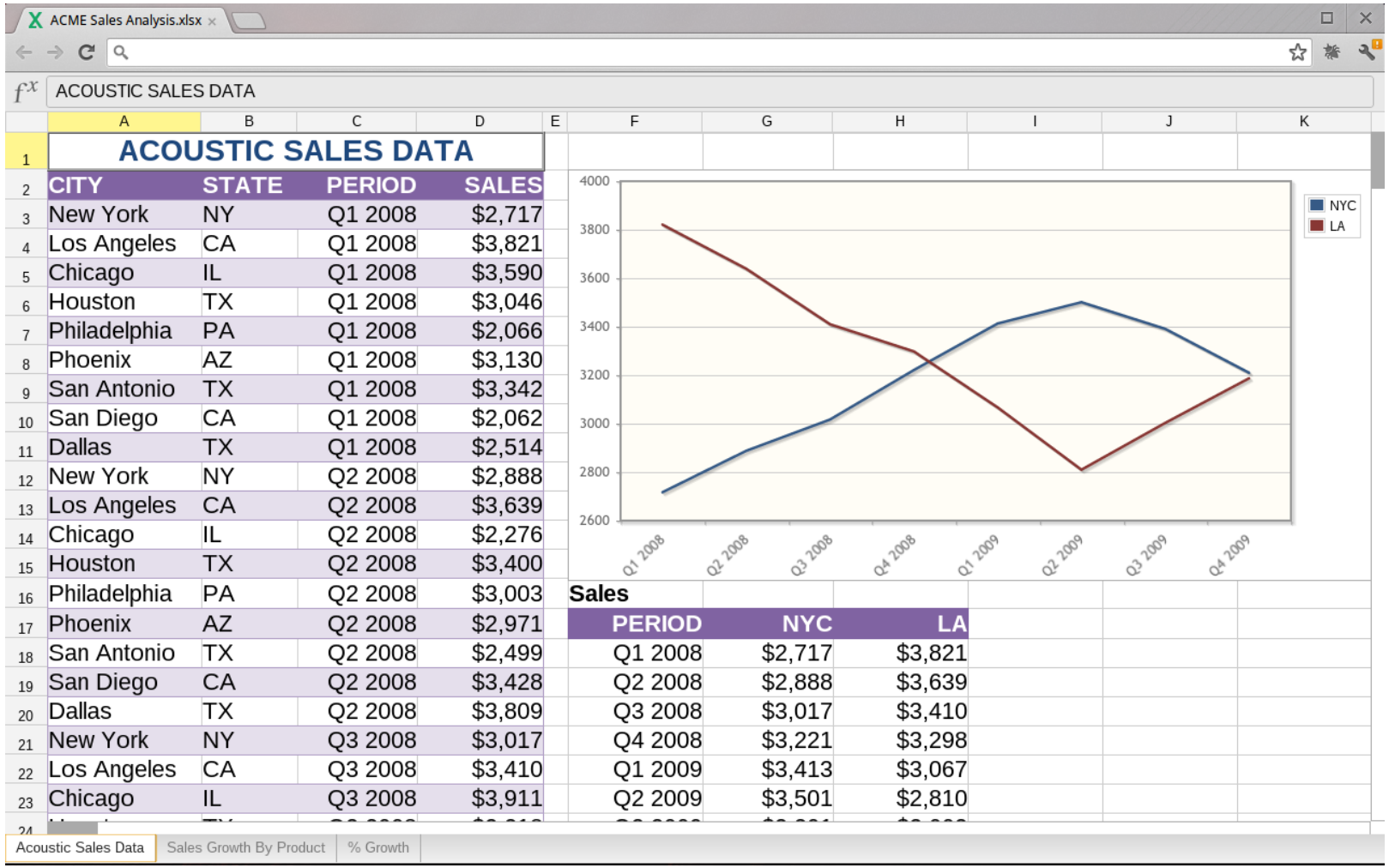
JS

```
function lint() {
  // ...
  var rawJson = $('#contentta')[0].value;
  // ...
  var parsedJson = JSON.parse(rawJson);
  var pp = prettyPrint(parsedJson);
  $('#contentdiv')[0].innerHTML = pp;
// ...
```


Extensions
ManifestLint (safe)
ManifestLint (safe)
New manifest
alert(localStorage.getItem(localStorage.key(0)))
New manifest
{
name: ManifestLint,
description: ,
version: 1,
app: {
launch: {
Elements
Resources
Network
Scripts
Timeline
Profiles
Audits
Console
Search Console
Uncaught SyntaxError: Unexpected token a
main.js:209
<top frame>
All
Errors
Warnings
Logs

More involved CSP examples

Chrome OS document viewer



More involved CSP examples

Chrome OS document viewer

JSON

```
...
"content_security_policy":
"default-src 'self';
 script-src 'self';
 style-src 'self' 'unsafe-inline';
 img-src 'self' blob:;
 frame-src 'self'",
...
```

CSP in the Chrome Web Store

Manifest version 2

JSON

```
"script-src 'self'; object-src 'self'"

"default-src 'self';
 script-src 'self' https:;
 object-src 'self' https:;
 connect-src <XHR hosts>"
```

Manifest version 2 timeline

- Chrome 21 (mid August): Block new Chrome Web Store items that use manifest v1
- Chrome 23 (early November)
  - Block Chrome Web Store updates that use manifest v1
  - Chrome stops packaging manifest v1 items
- Q1 2013: Remove manifest v1 items from the Chrome Web Store search results
- Q2 2013: Unpublish all manifest v1 items from the Chrome Web Store
- Q3 2013: Chrome stops loading or running manifest v1 items

CSP in the drive-by web

- Supported by Firefox and Chrome
- `X-Content-Security-Policy` and `X-WebKit-CSP` headers

CSP with web frameworks

- Bringing up a web framework with CSP
- Ember.js Starter Kit as a Chrome App

Ember.js Starter Kit with CSP

manifest.json

JSON

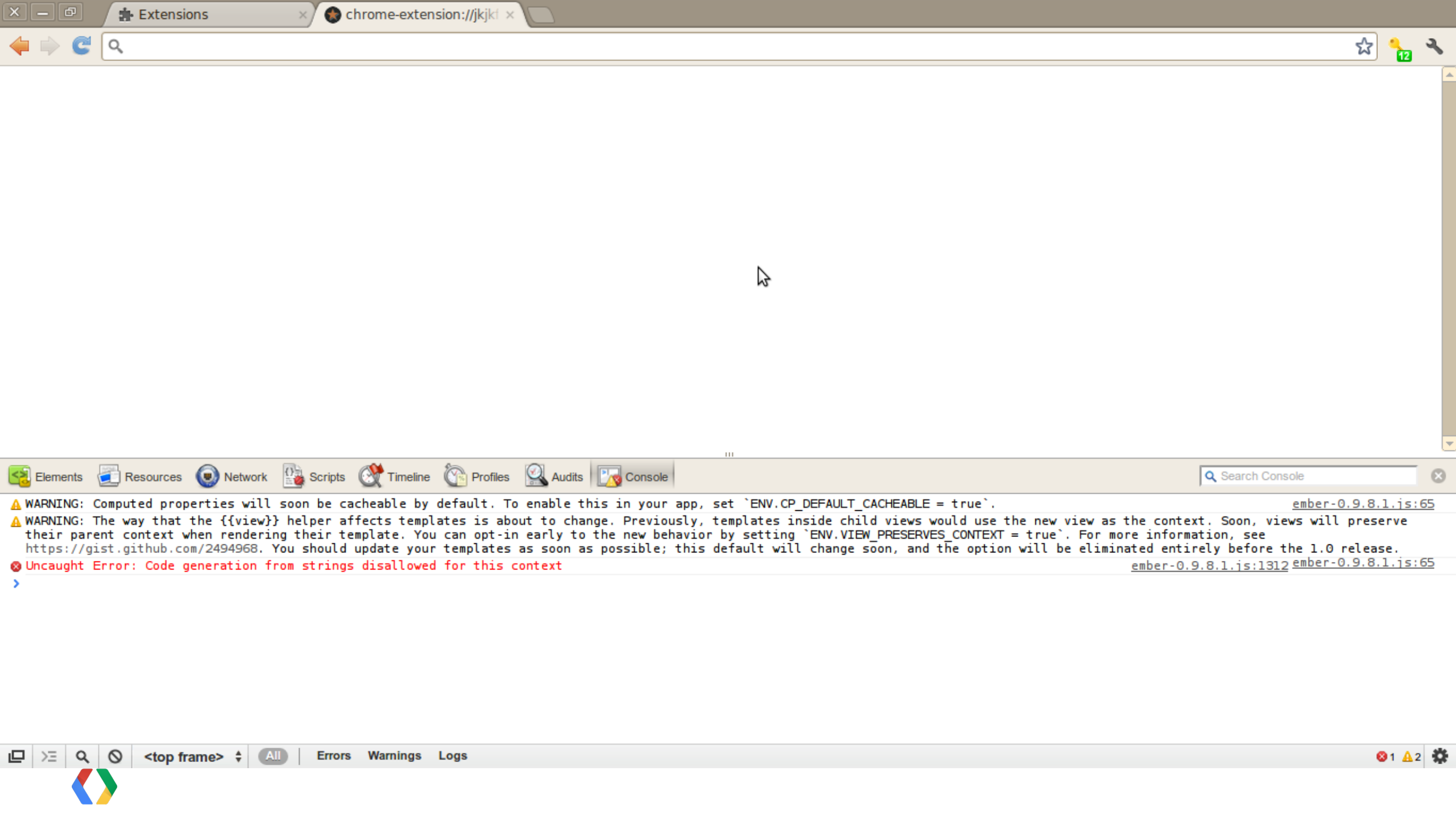
```
{
  "name": "Ember Starter Kit",
  ...
  "app": { "launch": {
      "local_path": "index.html"
    }
  },
  "content_security_policy": "default-src 'self';
      script-src 'self' https://ajax.googleapis.com;
      style-src 'self' 'unsafe-inline'"
}
```

Ember.js Starter Kit with CSP

index.html

HTML

```
<body>
  <script type="text/x-handlebars">
    {{#view App.MyView}}<h1>Hello world!</h1>{{/view}}</script>
  <script
   src="https://ajax.googleapis.com/.../jquery.min.js">
  </script>

  <script src="js/libs/ember.js"></script>
  <script src="js/app.js"></script>
</body>
```


Extensions
chrome-extension://jkjkf
Elements
Resources
Network
Scripts
Timeline
Profiles
Audits
Console
Search Console
WARNING: Computed properties will soon be cacheable by default. To enable this in your app, set `ENV.CP_DEFAULT_CACHEABLE = true`.
ember-0.9.8.1.js:65
WARNING: The way that the {{view}} helper affects templates is about to change. Previously, templates inside child views would use the new view as the context. Soon, views will preserve their parent context when rendering their template. You can opt-in early to the new behavior by setting `ENV.VIEW_PRESERVES_CONTEXT = true`. For more information, see https://gist.github.com/2494968. You should update your templates as soon as possible; this default will change soon, and the option will be eliminated entirely before the 1.0 release.
Uncaught Error: Code generation from strings disallowed for this context
ember-0.9.8.1.js:1312
ember-0.9.8.1.js:65
<top frame>
All
Errors
Warnings
Logs
1
2

Ember.js Starter Kit

Uncaught Error: Code generation from strings disallowed for this context

ember-0.9.8.1.js:1312

Ember.js Starter Kit

ember.js

JS

```
// ...
return Function.apply(this, params);
// ...
```

CSP don'ts

- `'unsafe-inline'` and `'unsafe-eval'` for `script-src`

Including untrusted content

- Untrusted content is OK if it doesn't interact directly with our app
- Make it live in an **unique** origin

HTML5 iframe `sandbox` attribute

- Include untrusted content/code in your app **safely**
- Sandboxed iframes can restrict the behaviour of the HTML/JS inside the iframe

HTML5 iframe `sandbox` attribute

HTML

```
<html>
...
<iframe sandbox id='sandbox' src='sandbox.html'>
</iframe>
...
```

Sandboxed iframe restrictions

- Unique origin
- No scripts
- No forms
- No top navigation
- No popups
- No plugins

Can be disabled individually (except plugins)

Back to code

Chrome OS document viewer app (`appViewer.html`)

HTML

```
<body>
<iframe id="sandbox" webkitallowfullscreen
 sandbox="allow-scripts allow-popups"
 src="viewer.html">
</iframe>
</body>
```

Sandboxed iframe

Chrome OS document viewer app (`viewer.html`)

HTML

```
<html><head>
<link rel="stylesheet" type="text/css" href="..." />
<script type="text/javascript" src="..."></script>
</head>
<body>
  <div id="app"></div>
</body>
</html>
```

Inter-frame communication

`postMessage()`

JS

```
function handleMessage(event) {
  if (event.origin !== <expected source>)
    return;

  // ...
}

window.addEventListener("message",
                        receiveMessage,
                        false);
```

Inter-frame communication

`postMessage()`

```
var parentWindow = parent.window;
parentWindow.postMessage("Sandbox reporting",
                         <target origin>);
```

JS

Inter-frame communication

`postMessage()`

```JS
var sandbox = document.getElementById('sandbox');
sandbox.contentWindow.postMessage("Ack", "*");
```

Sandboxed iframes don't s

- `allow-scripts` together with `allow-same-origin` lets sandboxed code remove the `sandbox` attribute

Common issues

- CSP restrictions
- Communication between different origins: `postMessage()`
- CSP and sandboxed iframes
- Unique origins are very restricted

CSP and sandboxed iframes

manifest.json

```
"web_accessible_resources": [
  "viewer.html"
],
"content_security_policy":
"default-src 'self';
 script-src 'self' chrome-extension://<id>;
 style-src 'self' 'unsafe-inline' chrome-extension:;
 img-src 'self' blob:"
```

JS

Restrictions for unique origins

- No HTML5 File API access
- No access to `localStorage`

Typed arrays

Manipulate binary data in Javascript

JS

```
// create an 8-byte ArrayBuffer
var b = new ArrayBuffer(8);

// create a view v1 referring to b, of type Uint8.
var v1 = new Uint8Array(b);

// create a view v2 referring to b, of type Int32.
var v2 = new Int32Array(b);
```

Blob URIs

Create an object in memory and get an URI for it

JS

```
window.URL = window.URL || window.webkitURL;

function handler(e) {
  // e.data is an ArrayBuffer.
  var blob = new Blob(e.data);
  var url = window.URL.createObjectURL(blob);
  // "blob:http%3A//host%3A5001/b0eace2a-adbc-4248-9c47-bfe96576f7d6"
  var img = new Image();
  img.src = url;
  // ...
```

Native Client (NaCl)

- Use native code securely
- Same fundamentals: `postMessage()`-based API

Take aways

- Use CSP to whitelist trusted scripts/resources
- **Leverage the browser to help you prevent XSS**

Take aways (II)

- Use sandboxed iframes to restrict privileges for untrusted content

Thank You!

jorgelo@google.com
+Jorge Lucangeli Obes

Links

- Manifest versions:
  - http://code.google.com/chrome/extensions/manifestVersion.html
- Content Security Policy:
  - http://code.google.com/chrome/extensions/contentSecurityPolicy.html
- HTML5 `sandbox` attribute for iframes:
  - http://www.whatwg.org/specs/web-apps/current-work/multipage/the-iframe-element.html#attr-iframe-sandbox