HOMEWORK II

waituckw@andrew.cmu.edu

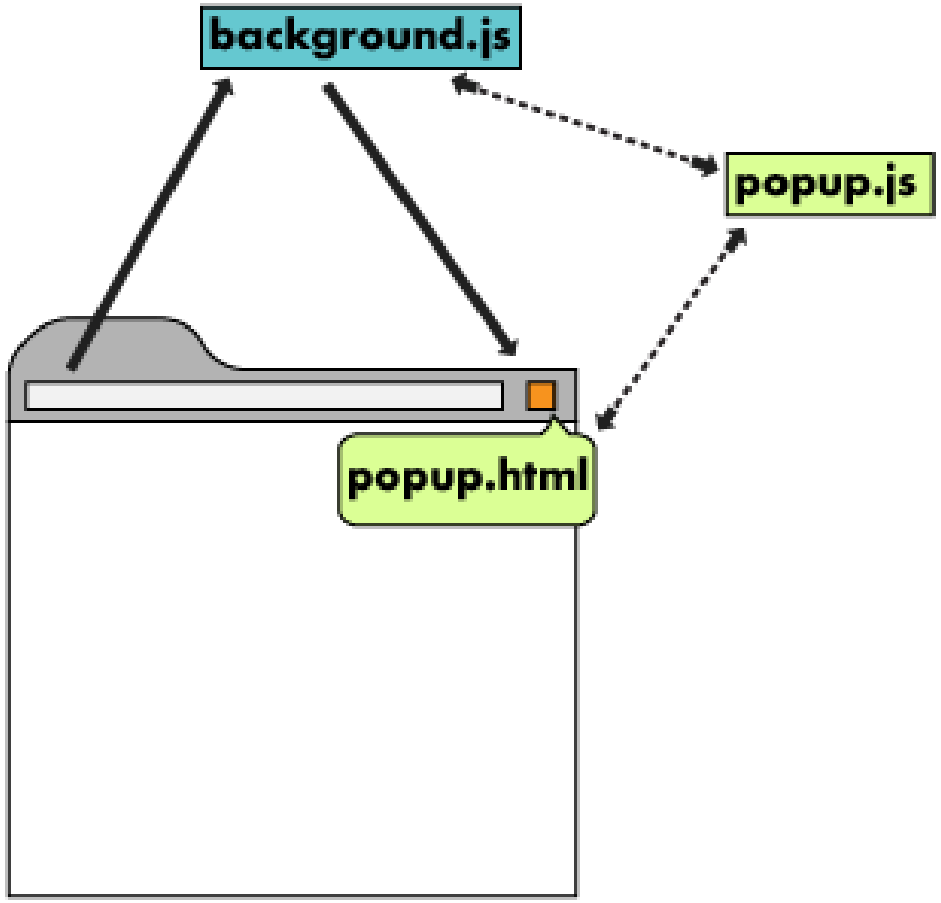
OUTLINE

1. Chrome Extensions Development
   1. Browser Action Extension
   2. Content Scripts and Including External Libraries
   3. Background Scripts and Permissions
2. Extension Exploitation
3. Build Your Own “Bad” Extension

CHROME EXTENSIONS

1. Web applications that act on web applications
2. HTML, CSS, JS, Manifest.json
3. Packaged as a .crx file (which is .zip with special headers)

CHROME EXTENSIONS



BROWSER/PAGE ACTION

This **Google Mail Checker extension** uses a **browser action**.

This **Mappy extension** uses a **page action** and **content script**.

On all pages, you can activate the button

On certain pages, you can activate the button

CREATING A UI FOR YOUR EXTENSION

```
{
    "name": "Getting Started Example",
    "version": "1.0",
    "description": "Build an Extension!",
    "permissions": ["storage"],
    "background": {
      "scripts": ["background.js"],
      "persistent": false
    },
    "browser_action": {
      "default_popup": "popup.html"
    },
    "manifest_version": 2
  }
```

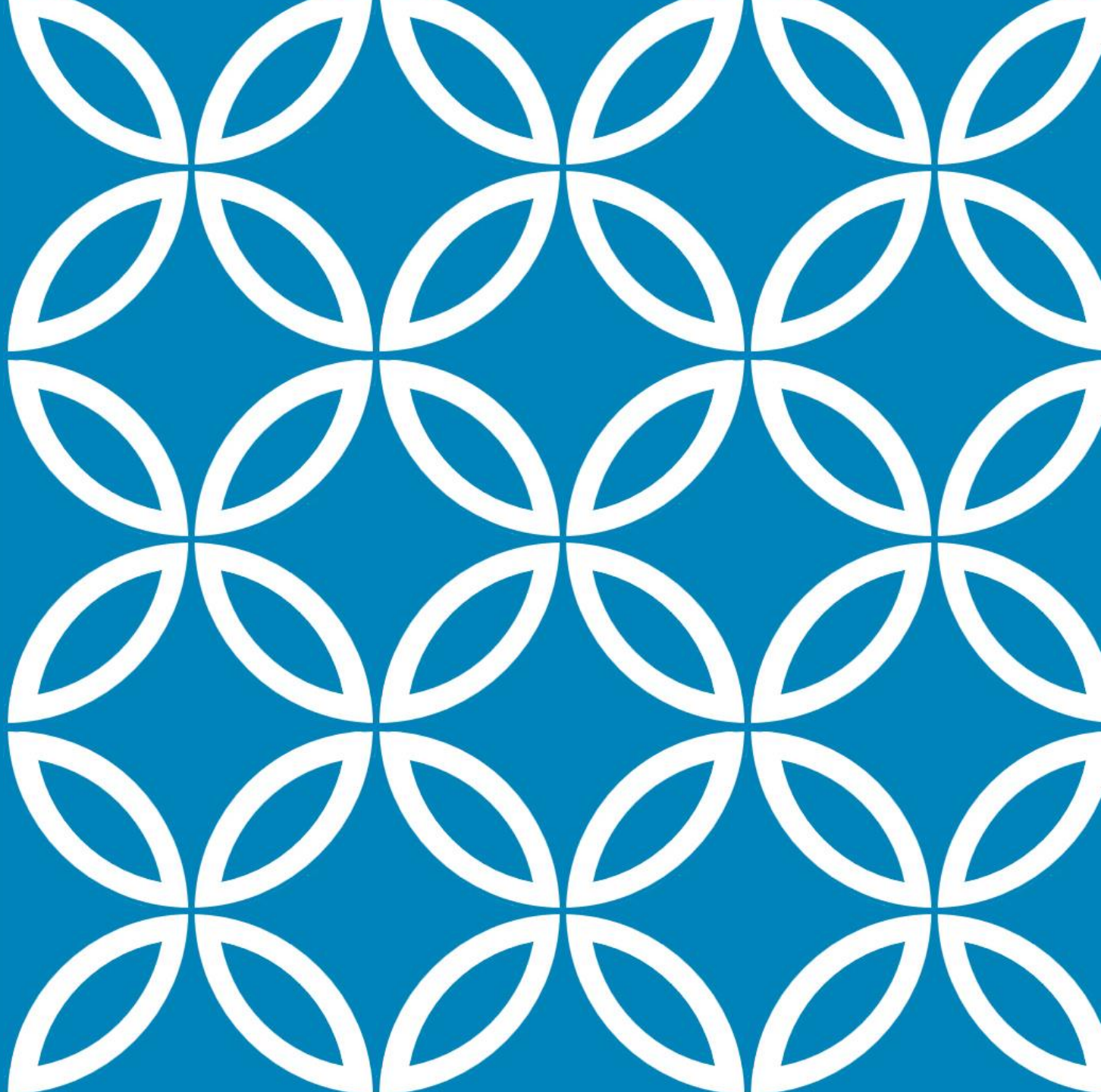
Manifest.json

```
<html>
    <body>
        <h1>Hello world!</h1>
    </body>
</html>
```

Popup.html

DEMO

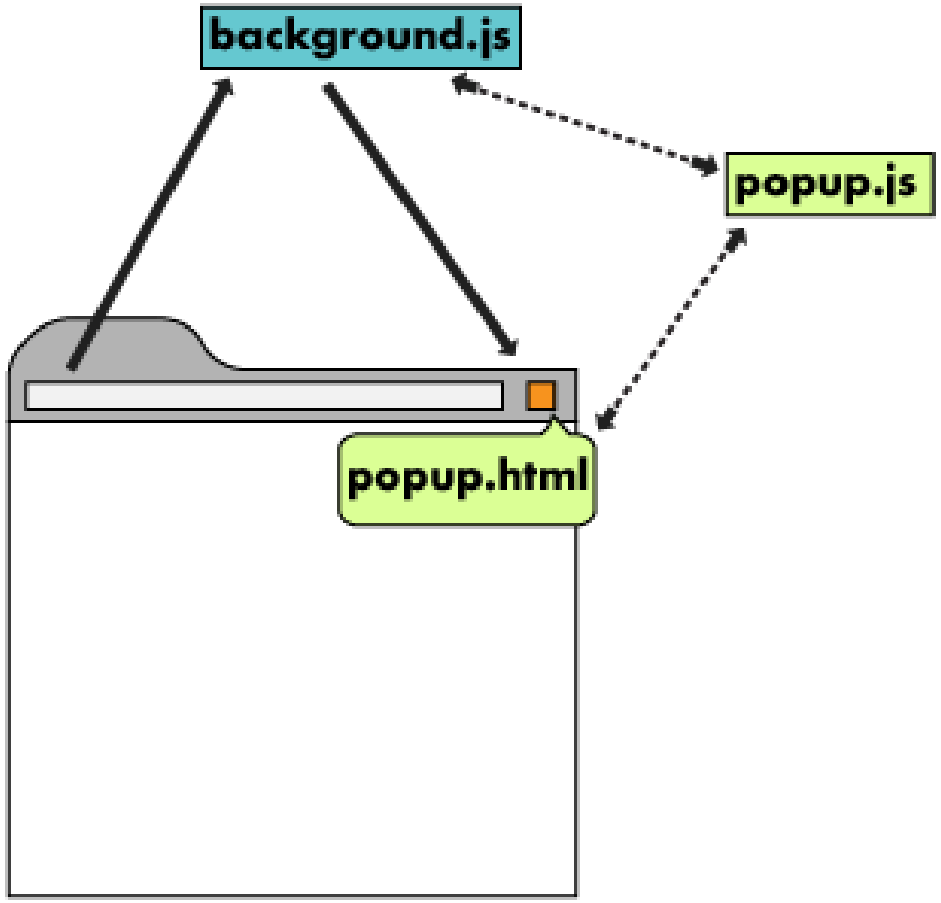
Installing and testing your first browser extension



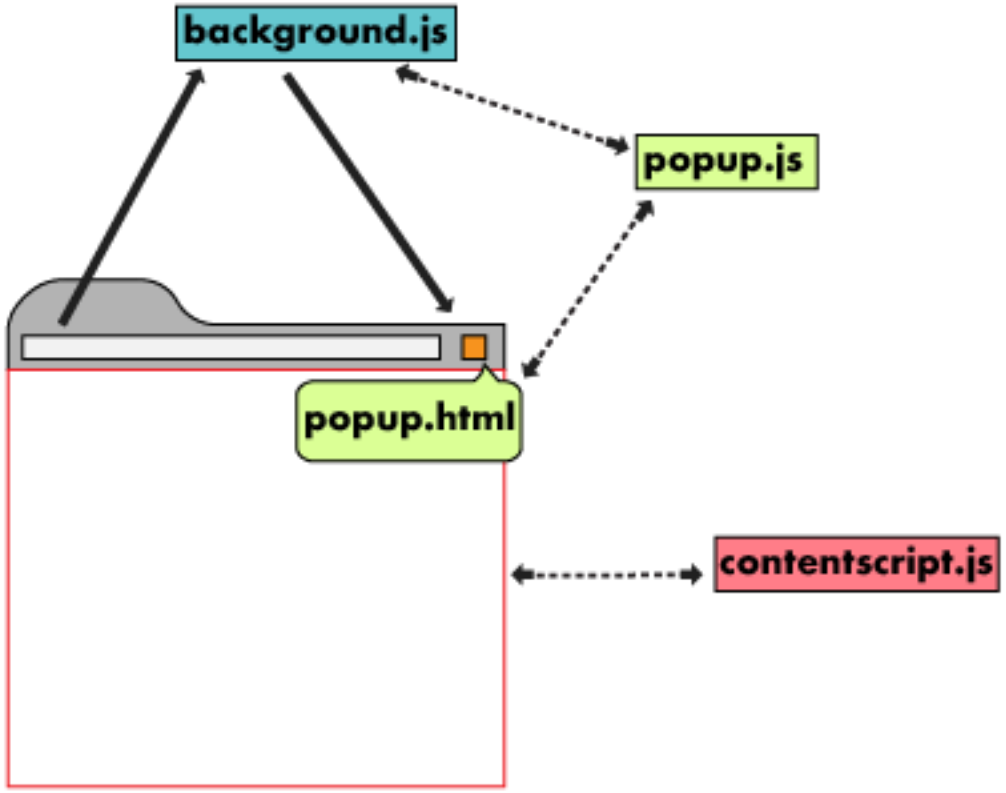
Q1.1 SUMMARY

1. Learn about browser actions
2. Learn how to create your own user interface in a browser extension through popups

CHROME EXTENSIONS



CHROME EXTENSIONS



CONTENT SCRIPTS

```
{
    "name": "My extension",
    ...
    "content_scripts": [
      {
        "matches": ["http://*.nytimes.com/*"],
        "include_globs": ["*nytimes.com/???s/*"],
        "js": ["contentScript.js"]
      }
    ],
    ...
  }
```

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

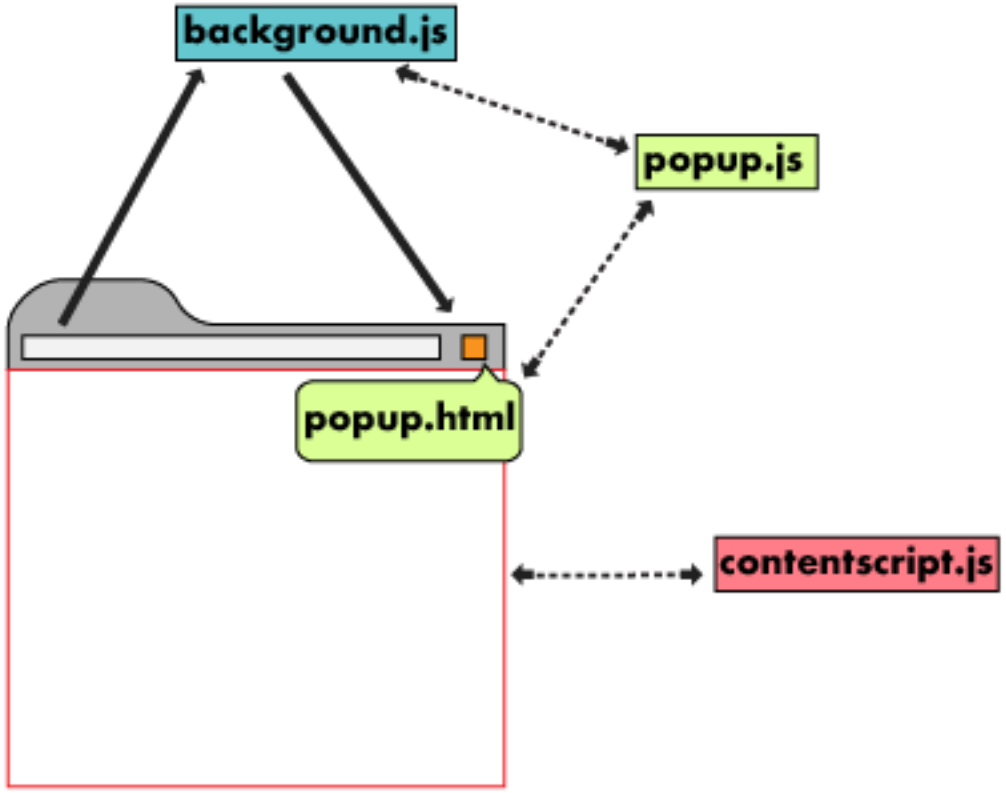
CONTENT SCRIPTS ISOLATED WORLDS

1. The web page cannot access variables declared in your web extension
2. The web page also cannot call functions declared in your web extension

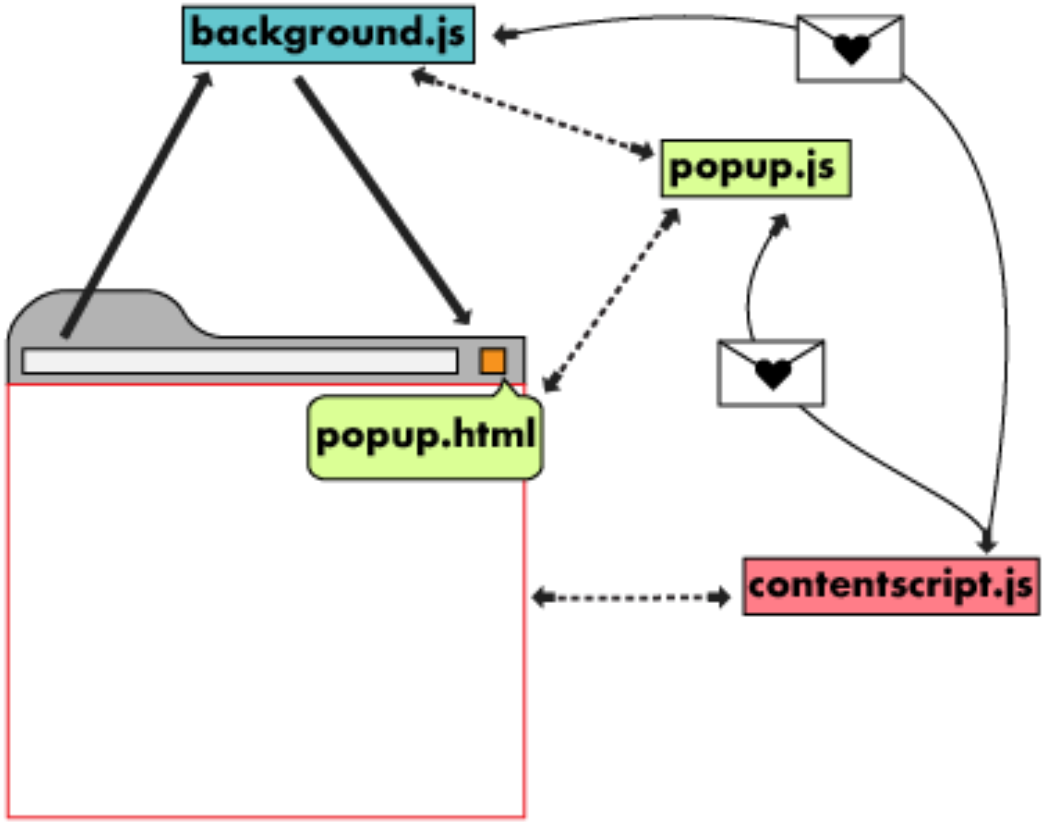
Q1.2 SUMMARY

1. Learn about content scripts
2. Learn how to use libraries in your content scripts
3. Learn how to use content scripts to modify the DOM

CHROME EXTENSIONS



CHROME EXTENSIONS



MESSAGING

popup.js

```
chrome.tabs.query({active: true, currentWindow: true}, function(tabs) {
    chrome.tabs.sendMessage(tabs[0].id, {greeting: "hello"}, function(response) {
      console.log(response.farewell);
    });
  });
```

Contentscript.js

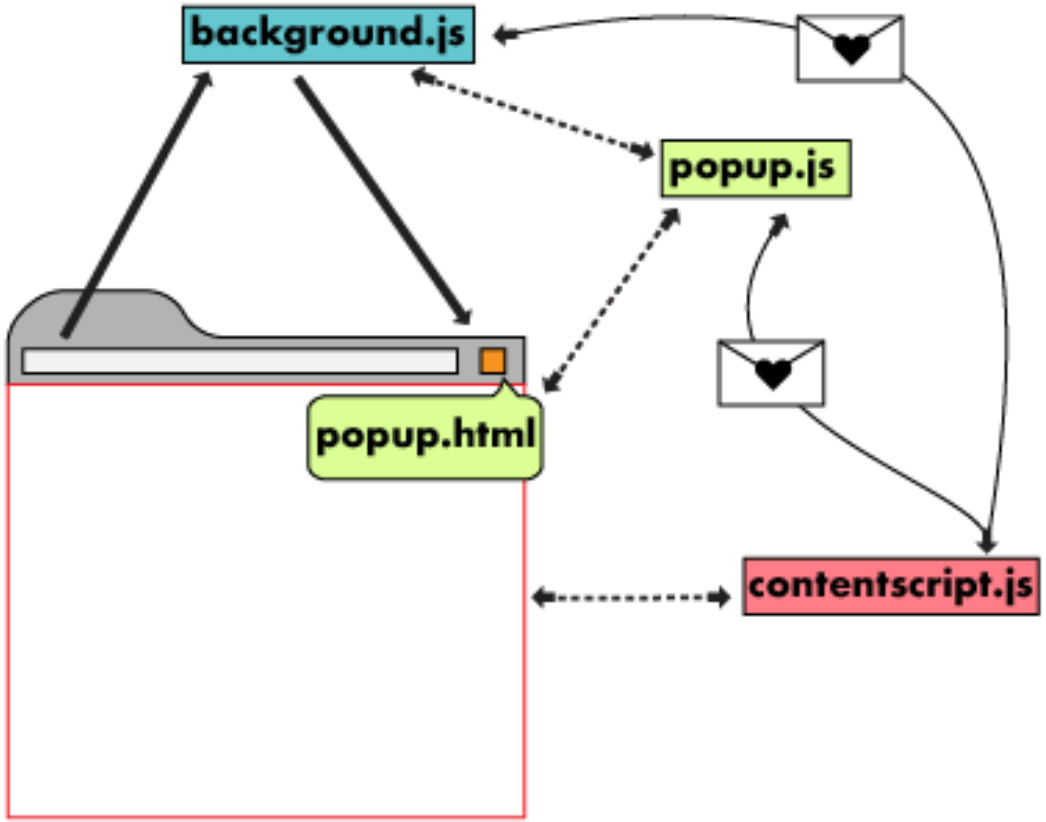
```
chrome.runtime.onMessage.addListener(
    function(request, sender, sendResponse) {
      console.log(sender.tab ?
                  "from a content script:" + sender.tab.url :
                  "from the extension");
      if (request.greeting == "hello")
        sendResponse({farewell: "goodbye"});
    });
```

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

Q1.3 SUMMARY

1. Learn how to interact with the content script from browser/page action
2. Learn about the messaging system in Chrome

CHROME EXTENSIONS

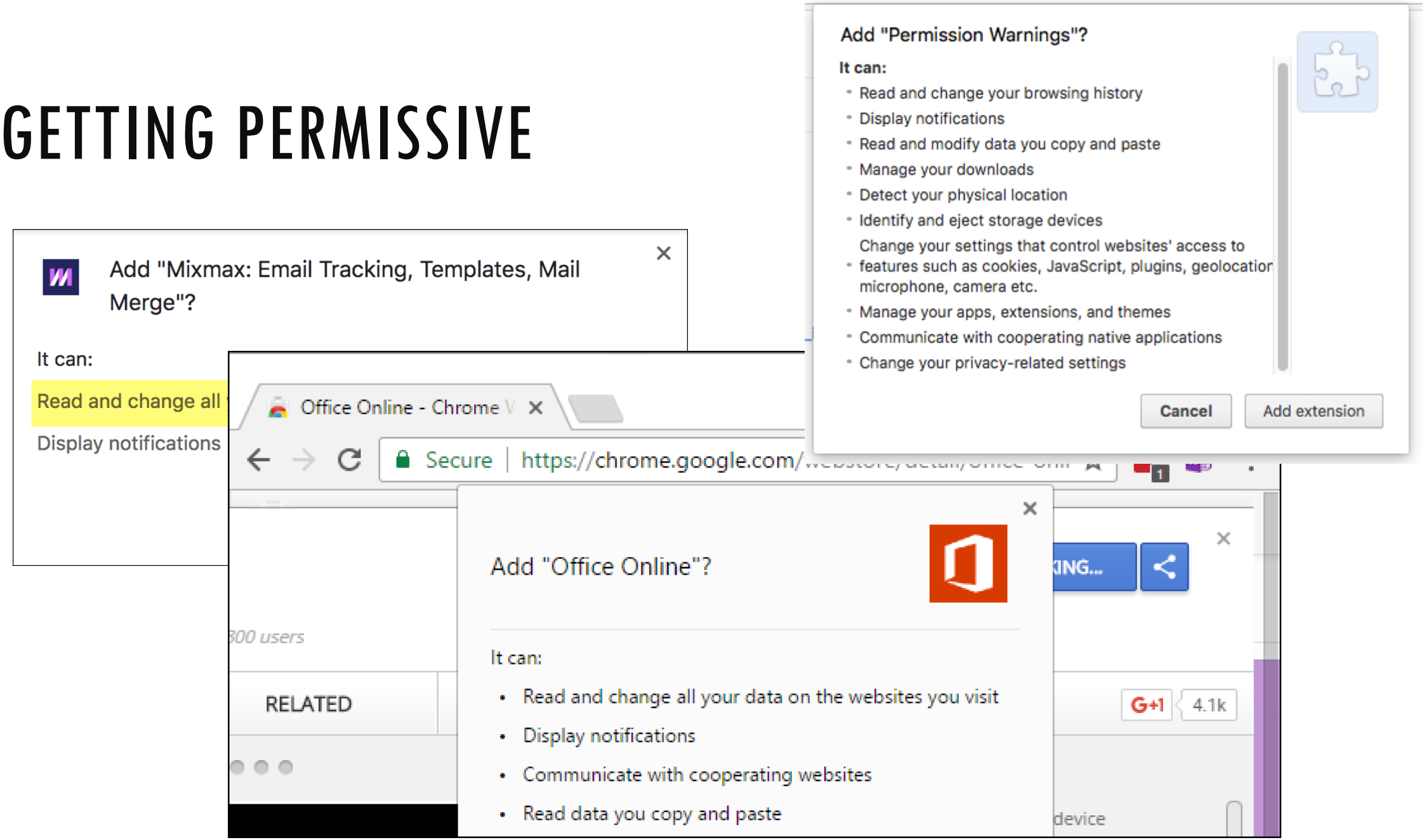


BACKGROUND SCRIPTS

1. Secret page with just the Javascript code loaded

```
{
    …
    "background": {
      "scripts": ["background.js"],
      "persistent": false
    },
    …
  }
```

GETTING PERMISSIVE



ADDING PERMISSIONS

```
{
   …
     "permissions": ["storage"],
   …
}
```

Q1.4 SUMMARY

1. **Track the user!**
2. Explore how to add permissions to enable your extensions

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

3. Learn how to make requests from extensions themselves
4. Learn how to use exotic APIs

OTHER TIPS

1. Developing – Use Visual Studio Code!
2. Debugging – Look out for useful error messages!
3. You might have to reload the extension…

DEMO | Debugging

BADNESS WITH EXTENSIONS

Even with the most benign of extensions, bad things can happen…

Universal XSS

- Achieve XSS on any web page

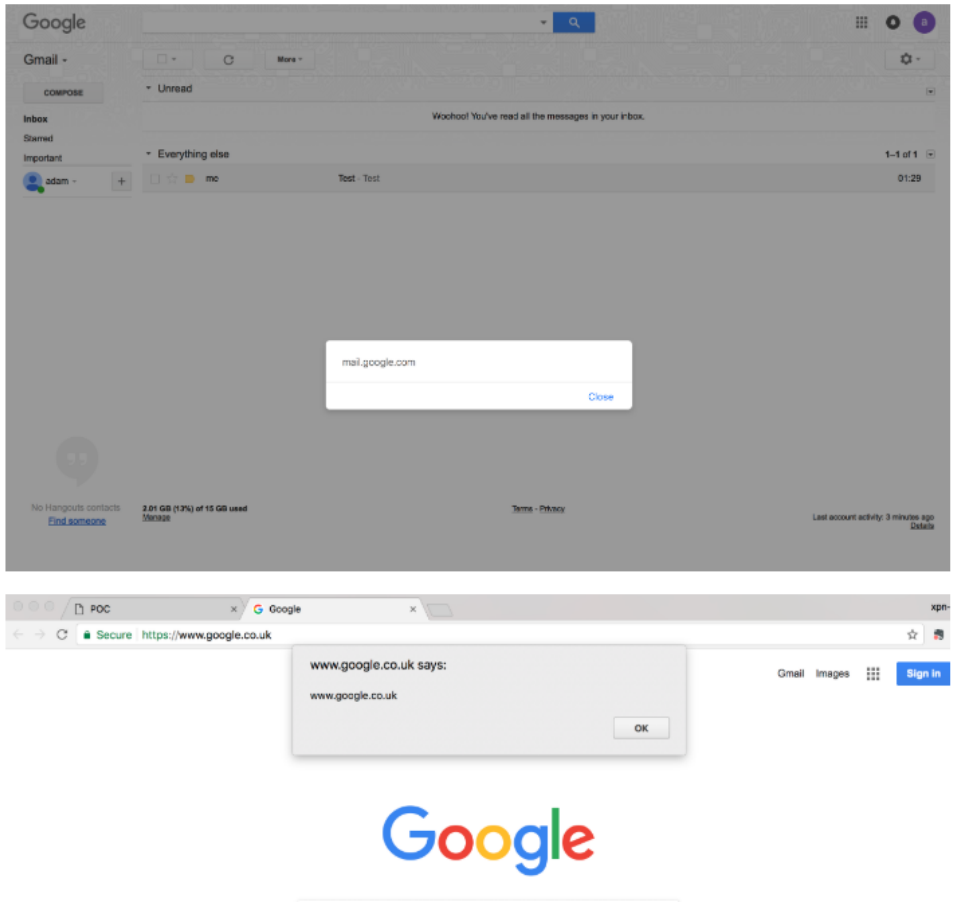
Inspiration:

https://blog.xpnsec.com/evernote-webclipper-uxss/

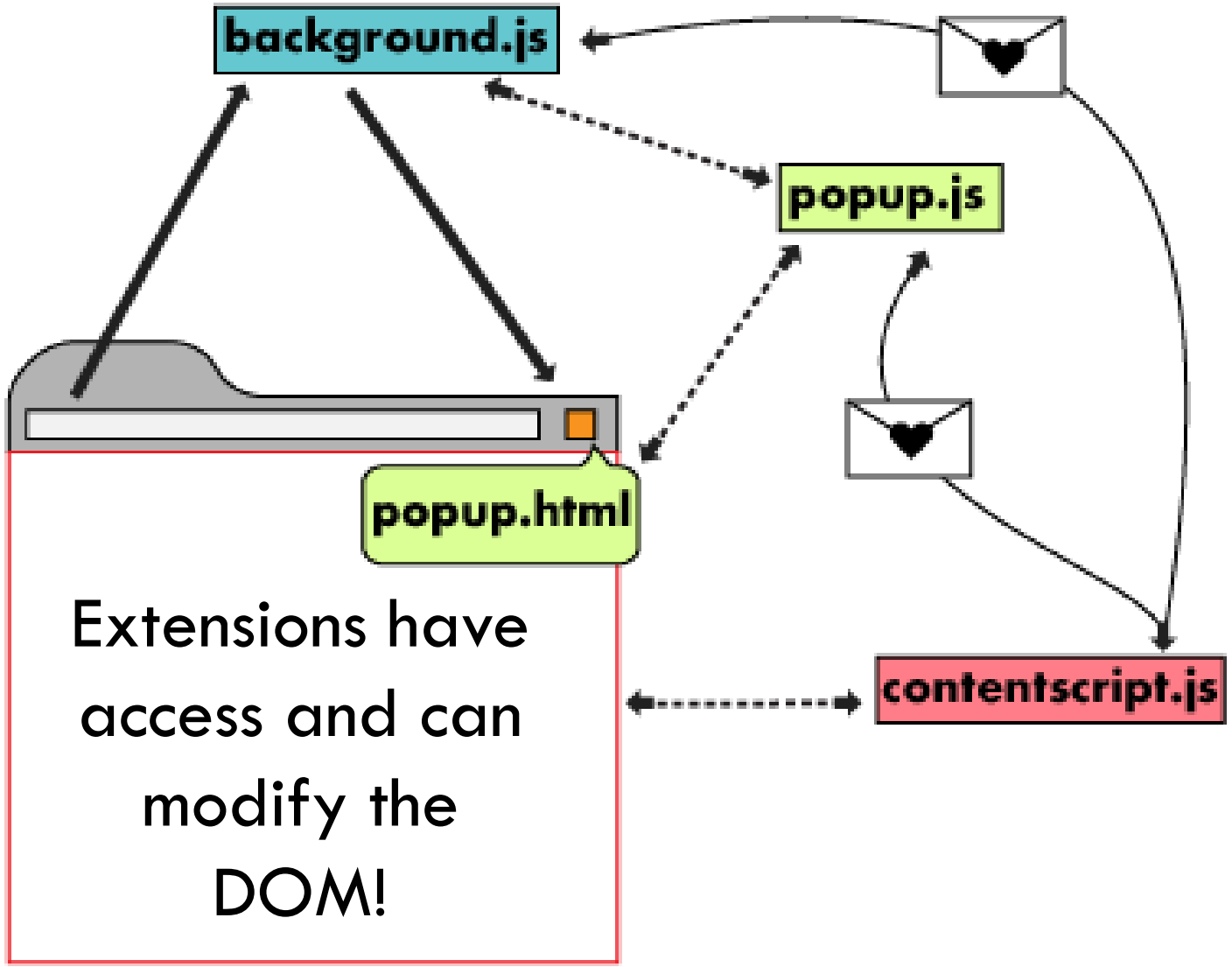
Universal XSS via Evernote WebClipper

Posted on 2018-01-18 Tagged in exploit, web, xss

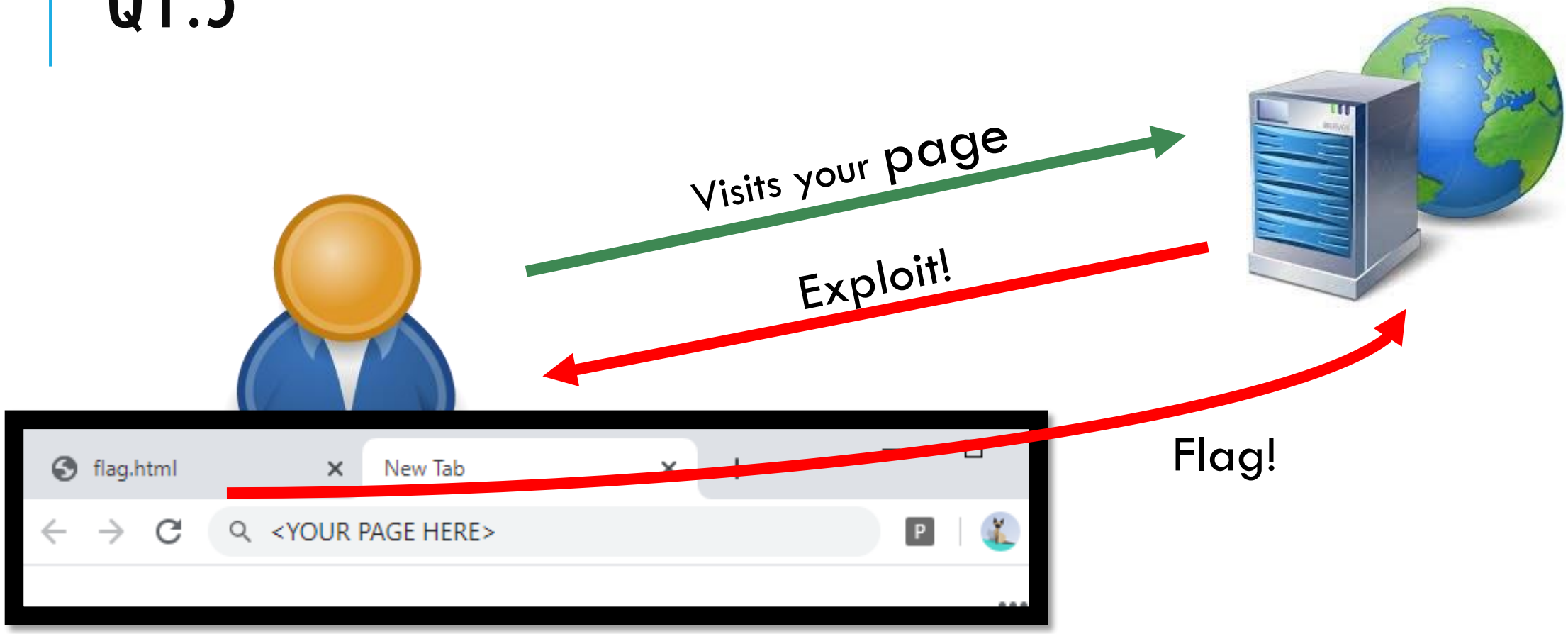
During an evening of bug hunting, I found a cool issue in Evernote's WebClipper tool. The results:



HOW CAN THIS BE ALLOW?



Q1.5


Visits your page
Exploit!
Flag!
flag.html
New Tab
<YOUR PAGE HERE>

Q1.5 SUMMARY

1. Learn to exploit extensions
2. Understand security implications of installing Chrome extensions
3. Exploit a novel class of vulnerabilities

DEMO

Debugging Universal XSS

HOW DO WE SECURE EXTENSIONS THEN?

1. Don't use HTTP
2. Use CSP
3. Limit external connections
4. Limit injecting to the DOM
5. Limit permissions
6. Avoid bad functions (document.write, innerHTML, eval)
7. Sanitize all user input
8. Check out other tips in https://developer.chrome.com/extensions/security

Q2 - MORE BADNESS!

1. We ask you to create a extension that tracks user activity on the web (URLs visited)
2. We want you to prove that you can track them by providing us with the server in which the data exfiltrated can be retrieved (via a CSV file – a comma separated row of URLs visited)
   1. Your server should run on localhost
   2. You may use a node.js application or have any other application server, provided you give us a Dockerfile.
3. You will need to create a Proof-of-Concept video to show that the exfiltration works
4. Basic Test Suite coming soon!

Q2.2 – EVEN MORE BADNESS (BONUS)

1. Come up with more interesting things to exfiltrate!
2. Come up with better ways of exfiltrating data (using less permissions, less obvious exfiltration traffic)
3. We understand you don't have infinite time, so this is a bonus
4. Graded on creativity, effort, technicality

SUBMISSION

1. Submit the writeup to Gradescope
   1. Each sub question requires a short writeup on what you did (not more than one page)
   2. Q2 should have a proper writeup, showing us why you did what you did
2. Submit the extension code to Canvas
   1. Each subquestion should have its own folder
   2. In a .zip file

END | **Questions?**