Challenge 1: Implementing a privacy-enhancing WebExtension

1 Challenge

Design and implement a privacy-enhancing WebExtension. A WebExtension is a piece of computer code that is added to a web browser and can intercept and filter all HTTP and HTTPS requests, among many other things. By properly filtering and manipulating the HTTP(S) requests, your program will as much as possible protect your privacy, while maintaining the functionality of the websites visited.

2 Details

You will be provided by code for a simple but fully functional WebExtension called `pewe` in which you yourself can add lines of programming code to process outgoing requests.

By default, it simply displays each outgoing the GET request into your browser's debugging output, the so-called console, so you can see what is going on. You will have to modify this code such that it takes appropriate actions that enhance the user's privacy, e.g. preventing some HTTP requests, or removing privacy-sensitive data from the request to the server.

The language in which WebExtensions are written, is *JavaScript*. You may not yet have encountered this language, so a brief introduction is given in Section 4.

3 Getting started

3.1 Installing Firefox

The WebExtension programming interface that we are going to use in this challenge, supposedly is compatible with both the Firefox and Chrome browsers, but in this instruction document we will assume you use Firefox.

So the first step is to install Firefox, if you do not have it already; you can download it from https://www.mozilla.org/ .

3.2 Making a separate profile in Firefox

Next, create a separate "profile" in Firefox. Firefox can distinguish between several "profiles", which may e.g. differ in what add-ons, such as WebExtensions, you use. By making and using a separate profile for this challenge, you can be sure that if you make an error in your WebExtension, you can still access webpages with a seperate instance of Firefox running under your "normal" profile.

On Windows, do this by first closing all instances of Firefox and then typing at a command prompt: `"C:/Program Files (x86)/Mozilla Firefox/firefox.exe" -P` or (on 64-bit versions) `"C:/Program Files/Mozilla Firefox/firefox.exe" -P` (quotes are important because of spaces) ; on Linux (and presumably MacOS), type `firefox -P`. This will bring up the profile manager, where you can make a new profile and/or select which one you want to use. Make a new profile, let's call it "challenge1".

3.3 Downloading the WebExtension and making Firefox use it

Download the zip file contaning the WebExtension from Blackboard; create a new folder, say "challenge1", and unpack the downloaded zip file into it.

Then start Firefox with the profile you created. Type `about:debugging` in the address bar, and choose "Load Temporary Add-on", navigate to the directory where you unzipped the file from Blackboard, and select the `manifest.json` file.

Now the WebExtension has been loaded, and you also have a nice "Reload" button which you can click anytime you've changed the WebExtension's code.

3.4 Test the WebExtension

To test the WebExtension, browse to a simple website, like the one used for last Monday's observation session: `http://netsys.ewi.utwente.nl/wireshark/` . If all is well, the page should appear. Press Ctrl-Shift-J to open the "console" where the browser (and extensions running in it) print debugging output: do you see the requests being printed?

Note that a lot of messages are printed in the "console", not just those from the WebExtension. The messages from the WebExtension have "background.js" at the right, since that is the filename of the core code of the WebExtension. For convenience, there's a trashcan icon at the top which empties the console window.

3.5 Look at the WebExtension's structure

The WebExtension consists of two files, which can be edited by any text editor, like notepad or better:

`manifest.json` This file describes the WebExtension, containing such information as its name and version, what permissions it needs from the browser, which file contains the background script (see below), and which script(s) to run when specific pages are loaded.

In principle, you don't need to edit this file, unless you want to do more advanced things.

`background.js` This is the background script, background meaning that it is not tied to a specific browser tab or web site. It attaches a "listener" to the "`onBeforeSendHeaders`" event; such a listener is a piece of code that will be executed every time the browser wants to send a request to a web server, but just before it actually sends it. This is the perfect place to intercept the request, perhaps to cancel it, or to remove privacy-sensitive data. The example script does no such filtering, it just prints the contents of the request.

It's this file that you'll need to modify.

Within `background.js`, you can setup your own code inside the `onrequest` function. You can start by modifying the examples that are there. The next section gives a brief overview of the JavaScript language.

4 The JavaScript language

Despite its name, JavaScript is quite different from Java, apart from superficial syntax similarities. It started out as a simple language to make web pages a bit more (inter)active. It is quite suitable for quickly cobbling together something, thanks to such features as untyped variables and automatic creation of data structures. In the course of time, it has evolved from the simple web scripting language into a powerful "multi-paradigm" language, allowing e.g. both imperative, object-oriented and functional programming styles.

For the purpose of this challenge, you don't need any advanced features of the language. Just some simple things like manipulating variables (including arrays) and simple control structures like if/then/else, for and while loops, will suffice. And fortunately, those are pretty similar to languages that you already know, like Java or C(++).

Here are some examples of manipulating variables, strings, numbers, etc:

```
var x;                    // declare a variable named x; note that you
                          // don't need to specify a type

x=3.14;                   // now x contains the number 3.14

x="abc";                  // now x contains a string

x=[10,20];                // now x contains an array with two elements
```

```
x[4]=x[0]+40;            // now x is the array [10, 20, undefined, undefined, 50]
                         // note that indices start at 0, like in Java and C(++)

x={};                            // now x contains an empty object
x.name="John"; x.age=23;         // now the object has several data fields
x.hobbies=["chess","swimming"];  // a field can also be an array again, in this
                                 //                   case containing two strings
                                 // and more complicated structures are possible
```

Some more examples of handling arrays:

```
var a=[1, 4, 7, 7, 7, 1];      // an array with 6 numbers
a[5]=16;             // overwrite the element at position 5; now a is [1,4,7,7,7,16]
a.splice(2,3);       // remove 3 elements starting at position 2; now a is [1, 4, 16]
a.splice(2,0,9);     // insert 9 into the array at position 2, so a is [1, 4, 9, 16]
var i;
for (i=0; i<a.length; i++) {                          // loop over all elements
  console.log("At position "+i+" we find "+a[i]);    // print a line for each element
}
```

And some examples of string handling:

```
var a="hello";
a=a+" world";          // "adding" strings concatenates them, a now is "hello world"
var b=3+4;
a=a+" "+b;             // "adding" a string and a number converts the number into a
                       // string for concatenation, gives a = "hello world 7"

b=a.substr(1,4);       // chop out a substring, gives b = "ello" (note that index 1
                       // refers to the second character of the string)

b=a.replace(/world/,"planet");  // replace world by planet
```

Much more information can be found on the web, e.g. on the Mozilla website: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference

5 Exercises

In this series of exercises, we will be using the following websites. Before you start, have a quick look at each of them.

- http://netsys.ewi.utwente.nl/ch1/
- http://www.whatbrowser.org
- http://www.whatsmyip.org/
- http://www.bbc.com

5.1 Whom are we talking to?

As you have browsed to the sites listed above, a lot of output was produced by the WebExtension: a list of requested URLs.

How many hosts are contacted when accessing the BBC webpage? Can you guess what the non-bbc hosts are meant for, just judging by their URLs? And after you've loaded the BBC webpage, scroll to the bottom of the console log window, and do nothing. After a while, you'll see more requests passing by; what are they for?

5.2 What are we sending?

Modify the `onrequest` function, to also print the request headers. Have a look at what information is in there, while trying the first of the above URLs.

5.3 Hiding your browser's identity

On the first website, you are presented with the name and version of your browser, along with your IP address. Discuss what you want to hide, and how you can do it. Can you hide every piece of information shown on that site? Why (not)?

Implement your solution in `background.js` and verify that you gained some anonymity.

5.4 There it is again

Browse to the second website. You'll notice that it is still possible to determine the browser you are using. Find out how this website is doing that, and how you can prevent it.

5.5 The BBC

When accessing the BBC site, lots of requests are made to other hosts. Have a closer look at those requests. What's in them? Do you really want those "partners" of the BBC to know this?

Enhance your WebExtension in such a way, that some of this tracking is prevented.

Note: we're using the BBC (British Broadcasting Corporation) website here only as an example of a typical modern website. However, this is just an example, and there are probably websites which are even worse in terms of privacy!

5.6 How far can you go?

What more can you do to improve your online privacy? How far can you go without destroying normal browsing experiences? Do you encounter any ethical questions?

From here on, developing your WebExtension is only limited by your creativity. While enhancing it, regularly check other websites that you normally use. Do they break when using your WebExtension? Or do they gain in usability?

Note: some sites use Javascript code which is contained inside the `.html` file, rather than Javascript hosted in separate `.js` files. Unfortunately, the WebExtension cannot filter the contents of the `.html` file as it comes in, so there's little we can do about this.

Also, if you're already a bit more experienced with HTML, CSS and JavaScript, feel free to improve the userinterface of your WebExtension, e.g., adding an statusbar that shows what it has done. You can find lots of information about programming WebExtensions on https://developer.mozilla.org/en-US/Add-ons/WebExtensions, with https://developer.mozilla.org/en-US/Add-ons/WebExtensions/API/WebRequest being the starting point for documentation about modifying HTTP requests.

6 Submission

When you're finished, demonstrate your WebExtension to one of the assistants and discuss your work with him/her. After that, be sure to upload your code to Blackboard. Furthermore, every group should submit a brief report (one page maximum) describing their findings and approach.

7 Learning objectives

- Understand how HTTP works.
- Insight into how the web works nowadays, and in particular the privacy issues associated with that.
- Understand how solutions to privacy issues can be implemented, and the drawbacks associated.