manifest.JSON: master file viewed by the chrome browser. Includes the title, icon, background scripts/pages, contents scripts, etc.

background.js: waits for the browserAction event, which is when the user clicks on the Chrome extension

htmlParser.js: logic that performs the operations of the chrome extension. Called in background.js

syntaxChecker.js: Ensures a web page's scripts and forms are suitable for use with the Enclave by checking for signature and secure tags. Called in background.js

EnclaveManager: C++ program that waits to receive messages from the browser extension. Messages arrive in the following format: first 4 bytes are the message length, the subsequent bytes form arguments delimited by '\n'. The first argument is a number corresponding to the name of the command to be called.

Installation/Usage
Clone this repository into home/.config/google-chrome/NativeMessagingHosts/ (The .config folder may be hidden. Press cltr+h to reveal hidden folders when viewing files on Linux).

Open the terminal within home/.config/google-chrome/NativeMessagingHosts/sgx-browser, make EnclaveManager.cpp as outlined in make.sh. Type "chmod +x EnclaveManager" in order to enable this program.

Delete debug_log.txt for now. This is the file that should be produced after running the extension and EnclaveManager

Move "com.google.chrome.example.echo" out of the sgx-browser folder and up one level; its path should now be home/.config/google-chrome/NativeMessagingHosts/com.google.chrome.example.echo.

Load the extension by opening home/.config/google-chrome/NativeMessagingHosts/sgx-browser when loading an extension in Chrome

Go to home/.config/google-chrome/NativeMessagingHosts/examples and run index_sgx.html.

Click on the extension and go to home/.config/google-chrome/NativeMessagingHosts/sgx-browser. You should now have debug_log.txt again with the scripts and forms of index_sgx.html

Overview

This is a Google Chrome extension that obtains and passes the form of secure websites to an SGX enclave. (See Google's Extension Tutorial for a more comprehensive overview for the architecture of an extension).

These files are located in the .config/google-chrome/NativeMessagingHosts folder, in accordance to Native Messaging Documentation provided by Google. (See https://developer.chrome.com/extensions/nativeMessaging for further reference).

Every chrome extension has an associated manifest file; manifest.JSON specifies the title, the icon, background scripts, and permissions of our extension.

The code for the actions of the browser extension is located in background.js, a script which is triggered by the browserAction event. (In this implementation, the browserAction occurs when the user clicks on the extension button in the Chrome Browser).

background.js initializes a Chrome Native messaging port with a Native Messaging host file named "com.google.chrome.example.echo". (This name is from Google's Native Messaging example and can be readily changed in the future). This file specifies, among other things, the path of the native program that our extension will communication with via Native Messaging. In this implementation, the native program is Enclave Manager.

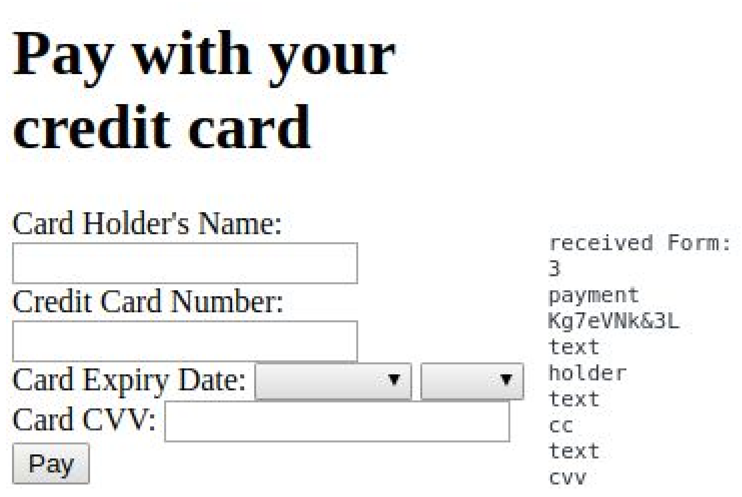
Upon the browserAction event, background.js creates a port with htmlParser.js and waits for a message (a webpage's scripts/forms) to relay to the Enclave Manger.

background.js then calls syntaxChecker.js, which ensures that a web page's script is suitable for use with the Enclave. It gets all of the script and form tags of a webpage to check that they are secure, have a signature, and in the case of forms, are of an accepted type. An error is thrown if any of these checks fail.

Background.js then calls htmlParser.js to obtain the script and forms to pass along to the Enclave Manager. htmlParser.js initializes two ports, one for the script of the page and one for the forms of the page. The extension will then iterate through all form tags, parse them, and write their contents to a string. Every form will have the following format:

- Numeric code that corresponds to the ADD_FORM command in the Enclave Manager
- Name of the Form
- Signature of the Form
- The type of attribute and name for each input of the form

PICTURE/EXAMPLE TO ILLUSTRATE FORMAT OF FORM



```
received Form:
3
payment
Kg7eVNk&3L
text
holder
text
cc
text
cvv
```

This program additionally listens to the form for on and off blur events in order to get the mouse coordinates for on and off focus events.

After parsing for the form, htmlParser.js obtains the script of the page.Every script will have the following format depending on whether it has a signature and whether it has a src embedded in its script. If it has a signature, the format will be:

- Numeric code that corresponds to the ADD_SCRIPT command in the Enclave Manager
- Signature of the script
- #EOF#

If it has a src embedded, the program retrieves the script from the src. The format will be a webpage script.

Both the forms and the scripts are passed to background.js through message passing. Once received, background.js passes these to the Enclave Manager.

The messages passed to the Enclave Manager have a numeric code at the beginning indicating the type of action that is required. At the current iteration, the messages are written to debug_log.txt. In future iterations, these messages will be passed to the Enclave.

```
received JS:
4
238&3fkYk23#@
#EOF#
/*
 * Credit card data validation JS API
 *
 * from https://github.com/juspay/card-validator/
 */

var __indexOf = [].indexOf;

var cardTypes = [
  {
    name: 'amex',
    pattern: /^3[47]/,
    valid_length: [15],
    cvv_length: [4]
  }, {
    name: 'diners_club_carte_blanche',
    pattern: /^30[0-5]/,
    valid_length: [14],
    cvv_length: [3]
  }, {
    name: 'diners_club_international',
    pattern: /^3([689]|09)/,
    valid_length: [14],
    cvv_length: [3]
  }, {
    name: 'jcb',
    pattern: /^35(2[89]|[3-8][0-9])/,
    valid_length: [16],
    cvv_length: [3]
  }, {
    name: 'laser',
    pattern: /^(6304|670[69]|6771)/,
    valid_length: [16, 17, 18, 19]
  }, {
    name: 'visa_electron',
    pattern: /^(4026|417500|4508|4844|491(3|7))/,
    valid_length: [16],
    cvv_length: [3]
  }, {
    name: 'visa',
    pattern: /^4/,
```