Electronics and Computer Science
Faculty of Engineering and Physical Sciences
University of Southampton

Thomas Hoad - tdh1g19
September 2021 - May 2022

Web Browser Security Plugin

Project supervisor: Erisa Karafili
Second examiner: Tim Norman

A project report submitted for the award of:
Computer Science MEng

Abstract

How safe do you feel whilst browsing the internet? We have such a reliance on it for almost every part of our daily lives, from chatting with friends to paying for anything we could wish for, that we could go our entire lives with only a phone to hand. Maybe we are vaguely aware of the severity and frequency of cyber attacks, yet we rarely think to spend our time or money on protecting ourselves. In a world full of security concerns, we are all too unaware of the risks that exist and are too reckless when it comes to our browsing activity. My project aimed to create a plugin that helps provide users with security information about their current activity that is pertinent to them and comes from a trustworthy source. This plugin took the form of a Google Chrome extension that was built as a mix of first-party analysis and third-party information. I believe my solution has revealed a gap for such an accessible and informative tool, but there are many more attachments that could improve it.

Statement of Originality

- I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.
- I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of the programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

We expect you to acknowledge all sources of information (e.g. ideas, algorithms, data) using citations. You must also put quotation marks around any sections of text that you have copied without paraphrasing. If any figures or tables have been taken or modified from another source, you must explain this in the caption and cite the source.

I have acknowledged all sources and identified any content taken from elsewhere.

If you have used any code (e.g. open-source code), reference designs, or similar resources that have been produced by anyone else, you must list them in the box below. In the report, you must explain what was used and how it relates to the work you have done.

I have not used any resources produced by anyone else.

You can consult with module teaching staff/demonstrators, but you should not show anyone else your work (this includes uploading your work to publicly-accessible repositories e.g. Github unless expressly permitted by the module leader), or help them to do theirs. For individual assignments, we expect you to work on your own. For group assignments, we expect that you work only with your allocated group. You must get permission in writing from the module teaching staff before you seek outside assistance, e.g. a proofreading service, and declare it here.

I did all the work myself and have not helped anyone else.

We expect that you have not fabricated, modified or distorted any data, evidence, references, experimental results, or other material used or presented in the

report. You must clearly describe your experiments and how the results were obtained and include all data, source code and/or designs (either in the report or submitted as a separate file) so that your results could be reproduced.

The material in the report is genuine, and I have included all my data/code/designs.

We expect that you have not previously submitted any part of this work for another assessment. You must get permission in writing from the module teaching staff before re-using any of your previously submitted work for this assessment.

I have not submitted any part of this work for another assessment.

If your work involved research/studies (including surveys) on human participants, their cells or data, or animals, you must have been granted ethical approval before the work was carried out, and any experiments must have followed these requirements. You must give details of this in the report, and list the ethical approval reference number(s) in the box below.

My work did involve human participants in an online survey. I was granted ethical approval with reference number 72120.

In addition to this, the following software was used for the development of my project that was not developed by myself: IntelliJ, DB Browser for SQLite, Google Chrome, Chrome Developer API, Google Safe Browsing API, IPQualityScore API.

Contents

Chapter 1

Introduction

1.1 Project Motivations

Cyber security is a major problem in the modern age of technology and can feel overly complicated for those who don't understand it or know where to look for information. Browsing the internet is such a necessity for most people for both work and free time that it makes sense to use some kind of protection to keep us safe. Given the risks on some websites, it is surprising to see most people not just not using tools, but also being completely unaware that they can protect themselves. Ad blockers have become a common tool for some, but are not quite as widespread as we might hope. Even if we use ad blockers, how do we know that the websites we are browsing are safe? Security scanners exist but have such low usage that we do not have enough awareness that they can benefit us or little to no cost. We need a better way to stay aware of threats and make informed decisions about our browsing habits. We need a way to question the security of websites that may appear safe so that we can collectively learn more about cyber security.

1.2 Project Description

My project will take the form of a Google Chrome extension that communicates with a Node.js server running in the background of the user's computer. The extension will provide security information to the user when they use their Chrome browser. The server will be used to compute the information from the extension and collect a database of results. The extension uses the Chrome Developer API to provide a suite of data collection tools to curate many first-party security tests, corroborated with third-party tests from the Google Safe Browsing API and the IPQualityScore API.

1.3 Project Goals

My goals are split into three tiers. I consider the primary goals essential, secondary goals are very much achievable and the tertiary goals could be possible with enough time. If not completed, this final tier will become future extensions of the project.

Primary Goals

- The extension provides a simple overall security rating for the website.
- The extension also provides detailed security information.
- The extension uses both first-party and third-party tools.
- The server uses web sockets for communication
- The server performs the computation of the security information for the plugin.
- The server uses a database to store this information.

Secondary Goals

- The extension collects information such as the source code, protocols, network traffic and more to send to the server.
- The server uses code review techniques on the information collected by the extension to look for vulnerabilities, malware, phishing attempts, exploits, etc.
- The server communicates with third-party APIs to collect security information about the current website.
- The extension provides an ad blocker to prevent pop-ups and stop trackers to provide a base level of security.

Tertiary Goals

- The server analyses the security of the browser itself, not just the current website.
- The plugin performs analysis on website login systems, looking for CAPTCHAs and attempting to perform crude SQL injections.
- The plugin can be expanded to provide MITM detection, VPN integration and more.

Chapter 2

Background Research and Literature Search

2.1 Research of Papers

A Framework for Website Security

This paper [1] identifies a framework used for identifying vulnerabilities by using a collection of scanning plugins to analyse the source code that links up to a vulnerability database. The user can choose the plugins available and receive a log file of the findings at the end. The paper identifies ten common vulnerabilities (only eight are named) that these scanners appear to be looking for the most. A custom program called VulScanner implements these plugins using a mixture of white box and black box testing that covers a wide range of languages, vulnerabilities and more. The result of the paper demonstrates how different plugins were able to find a large variety of vulnerabilities but at the cost of a long time. Arachini, one of the plugins used, identified almost 300 vulnerabilities on its own but it took around five and a half hours to work. The paper concludes that whilst its tool is not optimal, and takes quite a long time, the main purpose is that using a combination of scanning tools helps to improve the number of vulnerabilities found, and different plugins can cover the shortcomings of others.

The key weakness identified is that to find a lot of vulnerabilities, there is a cost in time. Security information needs to be timely when given to users, so this approach can only be used by the website owners well ahead of time. Constant scanning like this cannot be done by a user-hosted tool so this isn't an approach I can fully adapt to my solution. However, the level of information provided by these scanners is important. Third-party tools should be able to provide this information from an accessible platform using an API which could provide the more technical

vulnerability information I would like to provide users.

Detecting Visible Security Flaws

This second paper [2] introduces visible security flaws and how they can often be missed when securing a website. The process used here was to look at over 200 different financial websites to look for at least one of five visible vulnerabilities. This paper describes how 76% of the websites had at least one of these vulnerabilities. They looked at financial websites due to the growing use of online banking and hence the increased security pressure that is demanded of these websites. These figures are quite scary as it means that even the best-protected sites still have their openings for a cyber attack. Whilst this paper is relatively old, it still gives a good insight into the oversight that some companies may have over their security.

From this paper, I took inspiration from looking for similar visual security openings. If most websites undergo rigorous vulnerability testing, they may be more likely to miss these kinds of vulnerabilities than something more technical yet well-known. Third-party tools will check for a large number of those and a seal of approval for a website will be enough to guarantee safety from those. However, these tools may miss surface level vulnerabilities that an extension at a high level may have an easier time detecting automatically. The first three vulnerabilities identified here are breaks in the chain of trust, secure login options on insecure pages and security information on insecure pages. Detecting these could be a strong metric to showcase to users and should be easy for them to understand as well.

The Effectiveness of the CAPTCHA

This paper [3] looked at CAPTCHAs and how effective they can be. There are several varieties with different developers attempting to make one more suited for their individual needs. Whilst most CAPTCHAs take no more than 15 seconds to solve, they can have varying levels of utility and user experience satisfaction. For example, the invisible captcha was instant but very little user safety satisfaction because no one could see it, compared to the Google CAPTCHA which was perceived to be the strongest despite it being the quickest to verify. The users seemed to like to know a CAPTCHA was in place but didn't like spending any time unlocking it. In my plugin, I could add a detector that could look for any on the page ahead of time to inform the user of their presence. This would mean that the use of invisible CAPTCHA would be preferred because my solution can relay that it is present whilst the strong CAPTCHA works in the background without interrupting the user experience.

The Effectiveness of Ad Blockers

The next paper [4] was investigating the prevalence and impact of ad blockers on a global scale and a better breakdown in the US. The first section of this paper is setting up some equations to calculate the estimated number of users using an ad blocker, the second shows the results of these calculations. One of the most interesting parts of these results was a heat map showing the percentage of users with an ad block in different US states. A lot of the west coast states had an average of 25% prevalence compared to a lot of the southern states with roughly 10% prevalence. Thus, the more educated states/those with the best access to technology were more aware of their security online. I want to make a tool that is accessible to people and explains the importance of using such tools as an Ad Blocker. I could also try to implement my ad blocker in the plugin to make it a more general-purpose tool rather than having users download multiple different extensions. Creating an ad blocker may be a diversion from the vulnerability detection I had previously intended but it could also be a showcase of how this solution could be expanded with a wealth of security detection and prevention tools.

2.2 Research of Existing Solutions

Avast Security Scanner

The most popular solution I found on the Chrome extension store was the Avast Online Security Checker [5]. It had the highest number of downloads, the highest number of reviews and a very high average rating. Avast is a somewhat recognisable company by name so I had high hopes for their solution. It was very easy to install and can be opened quickly from an icon in the top right of the browser. This level of accessibility is what I believe most users will look for. The plugin works on two fronts: running in the background when using a search engine and in the forefront when opened from the icon. When browsing Google, it will show a shield icon for secure links in the search results. By far the easiest way to However, Google itself will background screen these websites so it was incredibly difficult to find a link it didn't trust. The extension can also block trackers and ad collection in the background, which I didn't notice much whilst using it. The second part of this solution was especially disappointing. It would give a very simple website rating and say either good or bad with a single line of text and a unique thumbs-up graphic. There is no further feedback as to how this is calculated which was not reassuring.

This tool should have been very useful but it lacked any amount of information.

I was hoping that from a reputable company, a more extensive report would be given. Whilst the accessibility was very strong and it felt like it was working constantly by showing results in search engines, it just didn't deliver on the main page of the extension. From this research, I knew I wanted to make my plugin a Chrome Extension to make it very accessible. There is no other platform that could've given me this level of accessibility. I also know that to compete with this solution and others, I need to provide more than the minimum amount of security information to back up my results.

WOT Security Scanner

The second most popular Chrome extension I could find was the WOT security scanner [6]. I was hoping that this solution may provide a more comprehensive security report than the Avast solution as it was the only other extension on the store that could compete with the popularity of Avast. It provided much of the same functionality as its counterpart but with a rating system as well. This rating system is based on user reviews and most websites didn't have enough up-to-date reviews to make this rating accurate or reliable. Once again, there was no sign of technical security information but did provide the same levels of accessibility as the Avast security scanner. This existing solution reinforced the takeaways I had from the Avast scanner.

SSL Trust

The first website based tool I researched was SSL Trust [8]. This company is an SSL/TLS certificate provider and so must test the security of a lot of websites thoroughly. Part of their website is a URL scanner and this is free for anyone to access. The most noticeable downside of this solution is the time it takes to scan a single website. For my test case, I used the Southampton university website [7] and this test took just over 2 minutes to perform the scan. However, the trade-off is worth it given the level of information in return. The tool covers multiple areas of security that seem to be discovered from a mostly network-based approach. The following are the main areas that the report provided from a scan covers.

- Malware and antivirus checks from roughly 90 third-party sources.
- Secure connection certificate checks.
- Network protocol checks.
- Vulnerability detection.

The results were comprehensive and are generally understandable for people at any level of knowledge. The third-party source checks were very useful and seemed to be one of the fastest parts to calculate. The slowest part was vulnerability detection. Some of the vulnerabilities found were Heartbleed, LOGJAM and more which are all found at the transport layer in the TLS protocols. These kinds of vulnerabilities may be better checked from a third-party database rather than being implemented into my vulnerability scanner as trying to detect these may either be inaccessible or may be seen as an attack.

Google Safe Browsing

Whilst SSL Trust seems like a reputable source, it wasn't a company that I had heard of before. My next line of research led me to discover how Google checks its websites to make sure it is hard for a user to find dangerous websites. Google Safe Browsing appears to be a feature embedded into Chrome and can be accessed through the inspect menu. This can tell you if the certificate, resources and connections are secure. If a user isn't using Chrome, then they won't benefit from this feature. There is also a website version of this called the Google Transparency Report [9]. This solution gave instant feedback, but much like the extension solutions, gave only a single indicator of whether it was safe or not. I was hoping that more information would be given by this given how it's unlikely people use it for regular browsing habits leaving it without many purposes compared to its competitors.

I also discovered during my research that this transparency report result can also be acquired using the Google Safe Browsing API. This can give the same information, plus a bit extra if the website has recently been attacked. Given the reputable nature of the name Google, this would be a good inclusion in my solution as a third-party source.

Pentest Tools

Pentest Tools [10] became very useful for the solution when comparing the results of my testing. The site could analyse a URL and find named vulnerabilities, and security information hidden in the browser and gave a useful overall security assessment for those that may not understand the technical jargon. You could also download a PDF report of the results to save for later. This solution also provides a more comprehensive scan, but for a cost and this was this solution's biggest downfall. The free scans were less informative that the full scan and you

could only perform a couple a day for free. Overall, this solution provided the best information but didn't have any accessibility.

Browser Audit

I also investigated a solution called Browser Audit [11] that gave an overview of the actual browsers' security information. No other popular solution could find the same issues as Browser Audit and this made it a very unique and potentially useful solution. Everything seemed based on the security of the browser itself rather than just the current website. As a website, it's slightly less accessible and recognised, but this tool could be integrated with a plugin to solve this.

IPQualityScore

The other third-party API source I investigated was IPQualityScore [13]. This API was important to me as it provided information about recent malware, phishing and spamming attacks. Security vulnerabilities can be patched so quickly after first being recognised, that a website may only look insecure for a short amount of time. If I could use this tool to relay when a recent attack occurred, this may give users an indication of how frequent or likely an attack will come. If a website is secure and trustworthy, there shouldn't have been any recent attacks of significance. As most users do not know when an attack occurs unless it attracts large scale media attention, this would be a very useful metric to relay.

Chapter 3

Final Project Specification

3.1 Solution Architecture

The project development will be split into two halves: the client-side Chrome extension and the back-end server. I decided on an extension for Chrome because this platform is incredibly popular and the research into the Chrome Developer API showed how useful it can be. As per this API, the front end will consist of a JSON manifest, a background JavaScript file and an HTML file for the display. The client-side will communicate with the server via web socket, sending JSON messages to each other. The server will be written in JavaScript using Node.js to run it as a server away from a web browser. I considered using Java here, but Node.js allowed me all the functionality I needed for using web sockets. This server will also use a web socket connection to communicate with an SQL database that stores the security information collected.

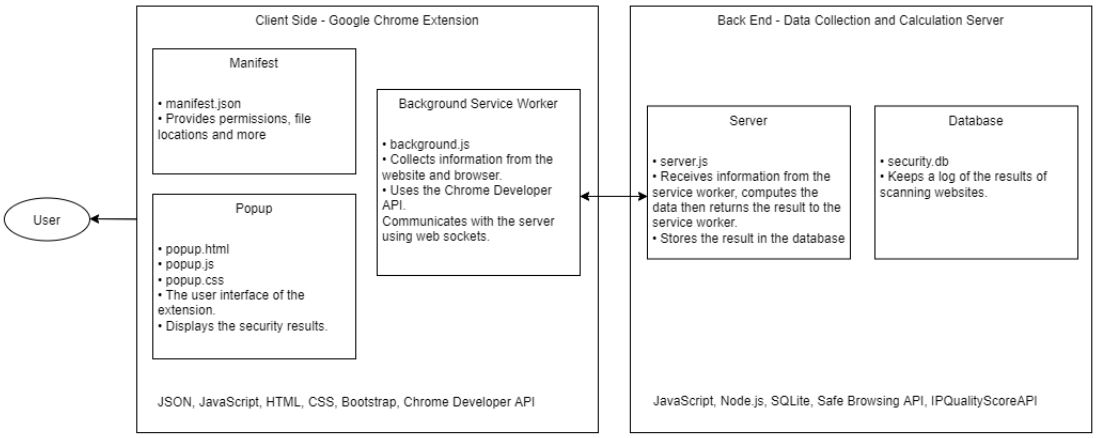

Figure 3.1: Solution Architecture

3.2 Information Flow

Shown below is a sequence diagram showing the different systems and how they communicate with each other in an example scan of a website. All communication, except for displaying the results, is done via web sockets over a prearranged port. An API key is required for communication with third-party APIs. For every new website scanned, the sequence diagram will repeat itself, although the server will remain connected to the database.

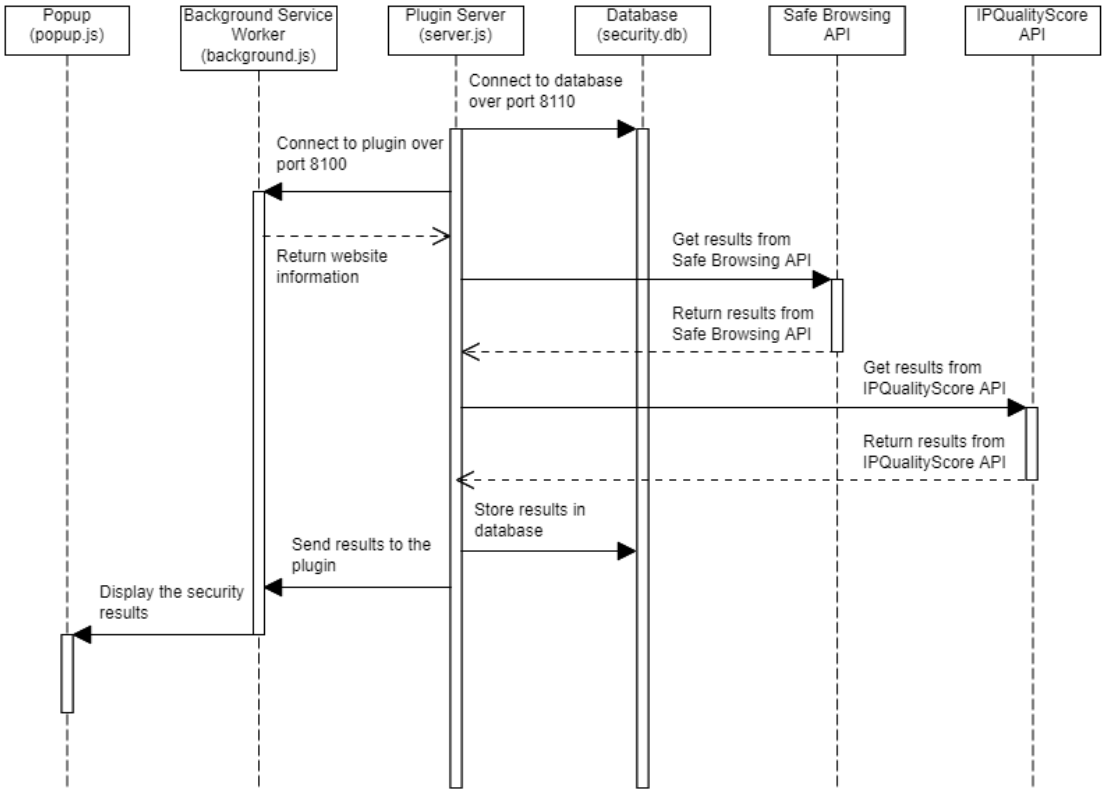

Figure 3.2: Plugin Sequence Diagram

3.3 Database Design

The database is shown below represented as an entity-relationship diagram. The diagram is constructed using IntelliJ to automatically recognise the attributes and display the tables automatically.

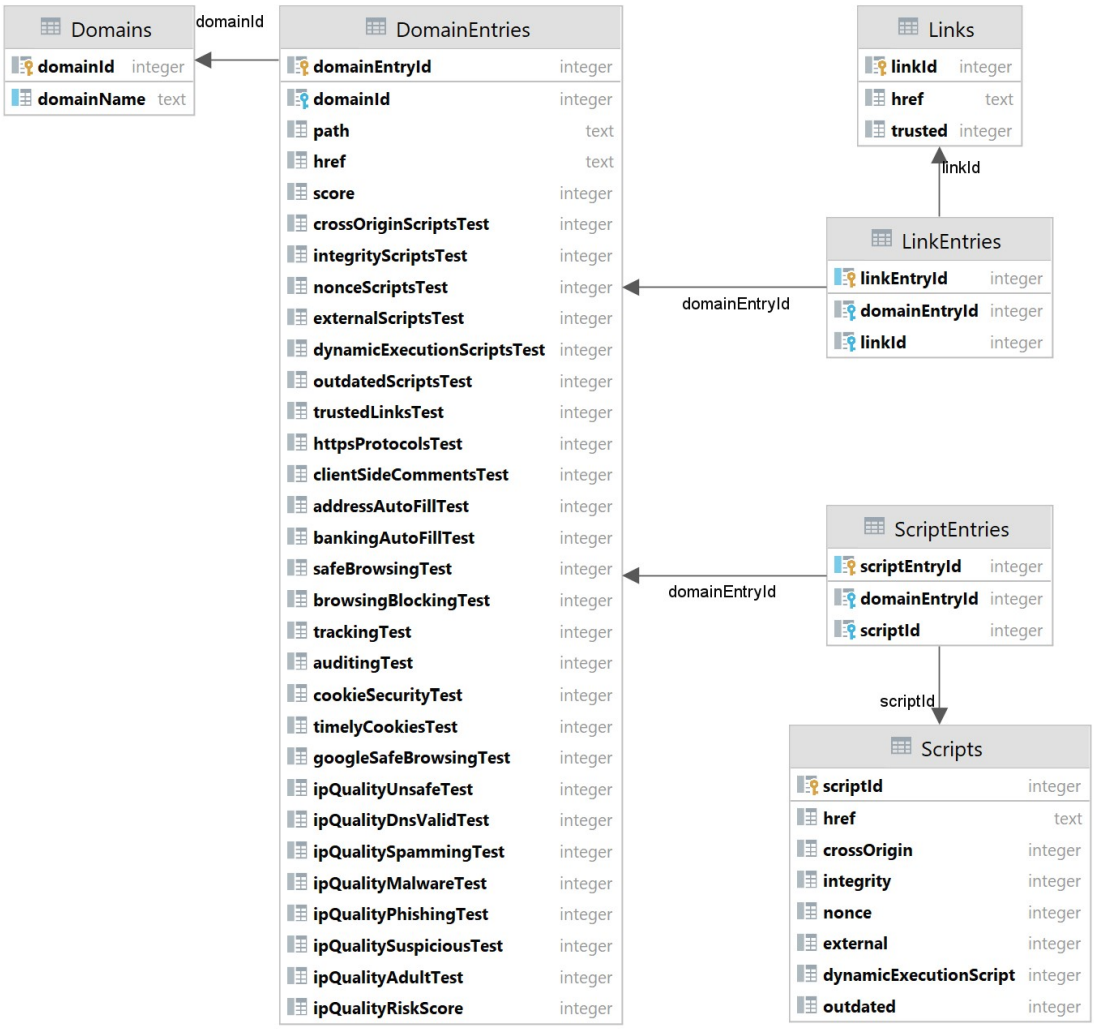

Figure 3.3: Database Final Structure

When a new scan of the website is completed, the following records are added to the database.

- A new domain into Domains if the domain does not already exist in the table.
- A new link into Links if that link path does not already exist in the table. This record also contains the test results for that specific link.
- A new script into Scripts if that script path does not already exist in the table. This record also contains the test results for that specific script.

- A domain entry into DomainEntries with all the test results and a bit of background information.
- A link entry into LinkEntries with the associated domain entry id and link id.
- A script entry into ScriptEntries with the associated domain entry id and script id.

3.4 Project Setup

IntelliJ	• My IDE of choice. • Ultimate version provides web development tools. • Easy to organise code structure. • Integrates databases, server running and more.
Google Chrome	• Very popular web browser and good extension store. • Used to run the extension. • Makes use of the Chrome Developer API.
DB Browser for SQLite	• Used to manage and update the database easily. • Designed to work with SQLite, has a very readable UI.

GitHub Version Control	• Integrated into IntelliJ to push and pull code directly. • Visible online to review commits that mark progress through development.
Third-Party APIs	• Chrome Developer API used for extension development. • Google Safe Browsing API and IPQualityScore API used for security scores.

3.5 User Interface Design

Shown below is the design for my user interface. The layout distinctly separates the first-party and third-party results. The top of the extension gives a visible security rating that can be recognised at a glance as well as a brief overview so that the user knows the results correspond correctly to the current website.

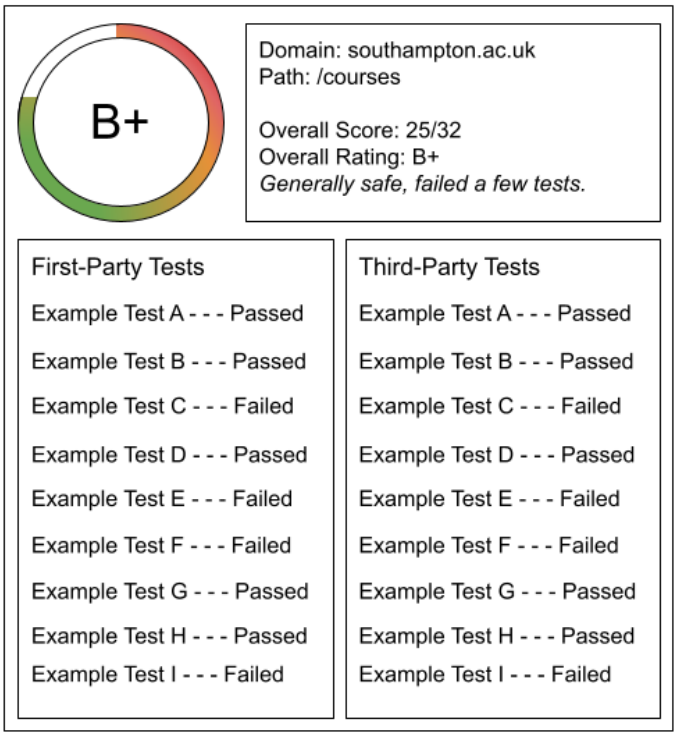

Figure 3.4: User Interface Design

3.6 Gantt Charts

The Gantt chart constructed below shows the final time management of my project. To create this chart I used my GitHub commit history to find significant moments when I moved onto a distinct stage of my project. When compared to the draft Gantt chart, goals have either been split into smaller components or removed from the chart if they were not achieved. Several goals set were attempted during the 'scripts and links' stage in late December.

Task	Start Date	Duration	End Date
Basic chrome extension	29/11/21	7	6/12/21
Web scrapper	6/12/21	7	13/12/21
Code review	13/12/21	14	27/12/21
Vulnerability detection	27/12/21	14	10/1/22
Security rating	10/1/22	7	17/1/22
Ad blocker	17/1/22	14	31/1/22
3rd party integrations	31/1/22	7	7/2/22
Additional detection functionality	7/2/22	14	21/2/22
Cookies and caching	21/2/22	7	28/2/22
Extension settings	28/2/22	7	7/3/22

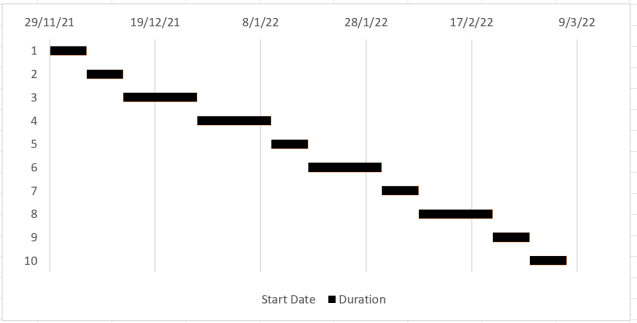

Figure 3.5: Planned Gantt Chart

Stages	Start Date	End Date	Duration
Basic Chrome Extension	29/11/2021	30/11/2021	1.00
Data Collection	30/11/2021	08/12/2021	8.00
Web Sockets and Server	08/12/2021	11/12/2021	3.00
Scripts and Links	11/12/2021	02/01/2022	22.00
Temporary Files	02/01/2022	08/01/2022	6.00
Improved UI	08/01/2022	21/01/2022	13.00
Browser Properties	21/01/2022	31/01/2022	10.00
XSS Attempts	31/01/2022	02/02/2022	2.00
Cookies	02/02/2022	06/02/2022	4.00
Database Setup	06/02/2022	18/02/2022	12.00
Database Logging	18/02/2022	01/03/2022	11.00
Google Safe Browsing	01/03/2022	07/03/2022	6.00
IPQualityScore	07/03/2022	08/03/2022	1.00
Improved Scoring System	08/03/2022	14/03/2022	6.00
Expanding Database	14/03/2022	18/03/2022	4.00
Bootstrap UI Update	18/03/2022	21/03/2022	3.00
Remote Code Execution	21/03/2022	03/04/2022	13.00
Final Test Additions	03/04/2022	04/04/2022	1.00
Testing Improvements	04/04/2022	20/04/2022	16.00

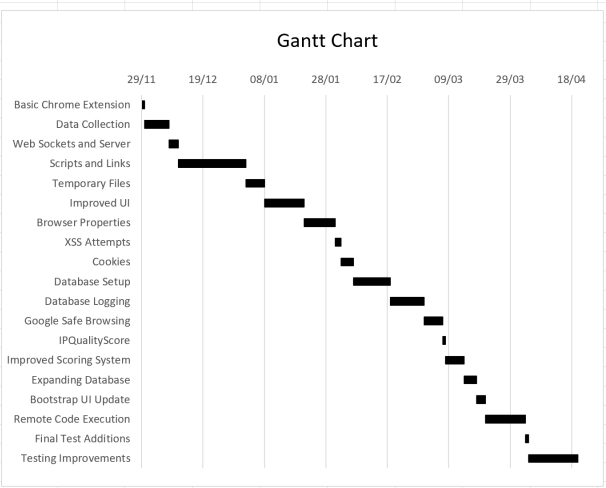

Figure 3.6: Final Gantt Chart

The goals omitted from the final product was the ad blocker and the additional functionality detection. These were all hampered by the recent updates to the Chrome Developer API [14]. A large shift in the changes to this API made that most extension developers had to completely rework large portions of their code. Google argued that this migration was intended to improve privacy, security and performance but they removed fundamental features of the API that made the development of certain extensions near impossible. The major change here was the decision to change the network request system.

The specific reference chrome.webRequest is used to allow tools to analyse, intercept, block or modify web traffic. This was incredibly powerful in allowing extensions to block certain content on pages automatically, such as ads. However, its capabilities were restricted by removing the blocking from most extensions and

removing the ability to read network requests. The API also pivoted away from background pages to service workers which will now only run for a limited amount of time.

As a result, certain features that I wanted to include like an ad blocker or network vulnerability detection would not be possible. More established ad blockers on the Chrome extension store can work but require more complicated workarounds.

3.7 Risk Assessment

Risk	**Likelihood / Impact (H/M/L)**	**Action**
No Internet Connection	L / H	Change locations
Losing programming work or report work	L / H	Use GitHub for version control and Overleaf to save report progress online
Supervisor is not available	M / L	Ensure regular updates and prepare questions beforehand
Lack of motivation	L / M	Balance the workloads of report writing and programming
Lack of time due to other modules / exams / coursework	H / M	Prioritise the correct work and ensure it is done early
Lack of time due to development time underestimation	H / M	Frequently revisit project scope to ensue the main goals are met
Lack of or outdated functionality from a section of the project	M / M	Find alternative languages / tools / APIs
Additional functionality required to continue / underestimation of the scale of a section	M / H	Scale remaining work down whilst ensuring the main project goals are met
High performance demands	L / M	Reduce the amount of computation done server side
Security concerns from the users from using my plugin	M / H	Don't ask for sensitive information and don't collect or display such information

Security concerns from the users from the results of my plugin	M / L	Report back security rating fast, clearly and recommend mitigating actions

Chapter 4

Implementation

4.1 Basic Chrome Extension

The basic template of a Chrome extension is outlined in a helpful guide [15] produced by Google. The extension requires a manifest, a user interface and a service worker. The JSON manifest file details the relative files in the extension and grants permissions for certain areas of the API. The user interface is a simple HTML file and the service worker is a JavaScript file. Setting this up gives us the basic extension which can be opened from an icon in the top right of the browser. The service worker will run in the background and the manifest shows it within the Chrome extensions page.

To background server requires Node.js to run. Like the service worker, this is just a JavaScript file to start with but requires a bit of code to be run on a web socket. IntelliJ can simplify this by setting up a configuration to remove the need to start the server from a terminal every time.

4.2 Data Collection

Data collection for the website is done using the JavaScript service worker background.js. The aim of this system is for the information to be collected every time a new website is loaded, refreshed or the active tab is changed.

To collect information from the website and the browser, I used two different avenues. Firstly I used the built-in HTML commands for the window and document objects to collect information such as the path, domain and URL protocol. These objects contain a lot of attributes that can be readily found on the website and so I used this to collect some basic identifying information about the website.

I have also used the document object to get the raw HTML of the website to be used for scanning by the server.

The second, and more significant, part of the data collection is the Chrome Developer API. This API is used to collect information about the browser and the cookie data stored in the browser. Detailed below is a breakdown of the tests used here and their related vulnerabilities.

Browser Information

Whenever the service worker is run, these results are collected immediately. These collect information about the browser related to its security. Whilst it doesn't provide the same level of results as Browser Audit, it does provide a good overview of how Chrome itself tries to reduce risk. The following identifies the parts of the API that are used to access this information. Below are the related tests with more specifics about what is being looked for and why.

- chrome.privacy.services.autofillAddressEnabled
- chrome.privacy.services.autofillCreditCardEnabled
- chrome.privacy.services.safeBrowsingEnabled
- chrome.privacy.services.safeBrowsingExtendedReportingEnabled
- chrome.privacy.services.doNotTrackEnabled
- chrome.privacy.services.hyperlinkAuditingEnabled

Address Auto Fill Test	Checks if addresses get auto-filled in forms. The following vulnerabilities exploit weaknesses CWE-287/264/200. The attacks that can exploit these vulnerabilities are most likely phishing attempts. • CVE-2012-3714 [16] • CVE-2021-21177 [17]

Banking Auto Fill Test	Checks if the related password information for a page gets auto-filled in forms. The following vulnerabilities exploit weaknesses CWE-522/200. The attacks that can exploit these vulnerabilities are most likely phishing attempts. • CVE-2021-35527 [18] • CVE-2022-0807 [19]
Safe Browsing Test	The API describes this factor as a measure of whether Google has safe browsing active. This is a feature that will try to prevent phishing or malware and is active by default.
Browsing Blocking Test	An extension of safe browsing, this will send information to Google if a page is blocked by safe browsing so they can discover malicious sites faster.
Tracking Test	If enabled, Google will ask websites not to track the user.
Auditing Test	Google will audit links on a page by pinging their requested destination.

Cookies

This section will also use the API to collect cookie data from the browser. Before this information is sent off, it is filtered to only send the cookies for the current domain to reduce the amount of data sent. The cookies have a lot of attributes that can be used to detect vulnerabilities, but this is computed on the server.

API Functionality

The API commands used for this system are detailed in the table below. Some of the functionality here can take a little while to run and certain parts of the API don't seem to work consistently which is why I have minimised the amount of the API I use.

chrome.privacy	Used for collecting browser information such as auto fill tests and safe browsing features.
chrome.webNavigation	Used to create a listener that will detect when the page refreshes.

chrome.tabs	Used to get the current active tab so that the correct website is used for getting the data.
chrome.scripting	Used to run the socket connection script to send the information back to the server.

4.3 Web Sockets

The primary communication method used within my system is via web sockets. The Node.js server establishes a web socket connection on port 8100 as soon as it starts and will listen for requests from the extension. Because of how extensions work in Chrome, a constant connection is not possible. The extension will reload quite often and service workers don't run for long so every time a webpage is reloaded it will send a new request to the server. This sequence is described in the sequence diagram in the design chapter. The data from the extension is sent as a JSON object and is received back as one.

This is designed so that there is minimal communication between the extension and the server. For each refresh, there should only be two messages sent over the web socket. The first will send the collected data and the response will send the computed results. The incoming message will contain information that is accessible in the source code and the outgoing message will merely contain the test results and no technical information on the location of vulnerabilities. This means that whilst the web socket connection may not be secure, the traffic being sent is freely available.

There is also another web socket connection with the database on port 8110. This is all handled on the server side so the extension never has to directly communicate with the database.

4.4 Server-Side Testing

The majority of the project revolves around the server that runs on the user's computer. Other extension solutions are unable to provide any detail and this server enables the computation possible to achieve this. The Node.js server will receive the data from the extension via the web socket as the JSON object to unpack and use in the computation. For each of the tests, I have tried to refer

to specific vulnerabilities related to the test that I came across whilst researching how to approach each one.

Scripts

The main area for weaknesses I believed to be in the scripts of a page. The potential vulnerabilities could be nearly endless here, especially if used from an external source. I have looked for key parts of the script tag as well as the script itself. To find the scripts, the server looks through the HTML to find script tags and then uses an XMLHttpRequest to collect the contents of the script if it is external.

Cross Origin Test	Looking for whether the script tag contains cross-origin="anonymous". If true, this means that the referenced script is executed without sending user credentials • CVE-2022-21703 [20] • CVE-2009-2816 [21]
Integrity Test	Looking for whether the script tag contains an integrity with a value. This value gets used as a checksum to make sure the script does not get manipulated. • CVE-2008-3843 [22]
Nonce Test	Looking for whether the script tag contains a nonce with a value. The nonce is a cryptographic value used to prevent attackers from accessing the content. • CVE-2022-25602 [23]
External Test	Checks whether the script comes from an external source. Internally and externally stored scripts pose their own weaknesses so it is worth knowing the destination of the scripts.
HTTPS External Test	Checks if an external script uses HTTPS protocols.

Dynamic Code Execution Test	After collecting the script source code, it will check for code that may allow for dynamic code execution. This includes methods such as eval(), window.execScript and innerHTML. • CVE-2020-15664 [24] • CVE-2017-7799 [25] • CVE-2019-16728 [26] • CVE-2019-11718 [27]
jQuery Version Test	jQuery is a very common script used by websites and hence this test checks if it is out of date. There are many vulnerabilities to be found in old version of this library so it was a very important test. • CVE-2020-11023 [28] • CVE-2020-11022 [29]

Cookies

After taking the cookies using the API from the extension, many security issues can be detected if they are insecure. There are two tests detailed below looking for the most likely problems with the cookies.

Cookie Security Test	Checks if the cookie is marked as secure. If it is marked as secure it will be sent as an encrypted request using the HTTPS protocol. • CVE-2014-3852 [30] • CVE-2015-4138 [31]

Cookie Timeliness Test	Checks if the cookie is marked as timely. Each cookie has an expiry date on it and this checks that the cookie is still pertinent. • CVE-2008-4122 [32]

Additional Tests

HTTPS Protocols Test	Checks if the website uses the HTTPS protocol. This is a very strong indication as to whether the current website is secure.
Client Side Comments Test	Checks if the website contains comments in the source code. Comments could contain sensitive or confidential information. • CVE-2007-6197 [33] • CVE-2009-2431 [34]
Trusted Links Test	Checks if the link directs to a secure location using HTTPS protocols.

4.5 Third-Party Tools

The two APIs I selected were Google Safe Browsing and IPQualityScore. Google Safe Browsing only provides me with the results for one test but comes with a reputation behind the name. IPQualityScore gives me results for tests that I was unable to compute such as malware, phishing and spamming detection.

Safe Browsing Test	Relays whether Google considers this URL to be safe. Google keeps threat lists of websites that may be unsafe from social engineering or malware and keeps a record of them with a marked threat type.

IPQualityScore Unsafe Test	A general test for whether the website is safe or not. The confidence of this test is recorded by a security score, another test I show in the extension.
IPQualityScore DNS Test	Checks if the domain has valid DNS records. This should be able to verify the validity of the URL.
IPQualityScore Spamming Test	Checks if the URL is associated with email spam attacks or an abusive email address.
IPQualityScore Malware Test	Checks if the URL is associated with malware or viruses. This should indicate a recent malware attack if it fails.
IPQualityScore Phishing Test	Checks if the URL is associated with phishing behaviour. This should indicate a recent phishing attack.
IPQualityScore Suspicious Test	Similar to the unsafe test, malware test and phishing test. Also corroborated by the security score test.
IPQualityScore Adult Test	Checks if the website contains adult content.
IPQualityScore Security Score Test	A overall score for the website. A recent attack will increase the score. Most websites will display 0 from this test.

4.6 Database Storage

The database serves as a way to keep a record of the results of scanning websites. It stores a profile on each of the unique domains, scripts and links with entries relating to all the associated files for each scan. The database can be queried to look through old results, although there is no direct user interface for this. The idea behind the database was to be able to detect how security scores for a website have changed.

The database is connected via a web socket with the server on port 8110. After the server has computed the security tests, the database is sent a series of messages to record the domain, scripts and links of a particular scan. For a comprehensive breakdown of the tables and the insert statements used, the design chapter describes the entity-relationship diagram for the database.

4.7 Displaying Results

Displayable Format

When the results are returned to the extension via the web socket response, they will return in a JSON object format. Once again, the extension makes use of the Chrome Developer API, this time using chrome.runtime messaging system. Rather than setting up another web socket connection, this allows the extension to relay messages securely to other client-side JavaScript files. The service worker will store the JSON result with chrome.storage in the browser, once the result has been received from the server. Whenever the popup is opened, popup.js will call for the results from the service worker which should send the same raw JSON that is stored in the browser. The calculation of results isn't instant so the user can keep refreshing the popup until the result arrives. Once popup.js receives the results, it will convert the JSON into HTML which then gets inserted into popup.html.

User Interface Implementations

The primary purpose of this extension is to relay security information back to the user. To do this, I needed a strong user interface to display the results. My initial design was a very basic HTML file with a bit of CSS used to create a template rating circle that will show a gradient colour and fill in more with a higher score. Shown below is the basic user interface that I used at the start of the project. This interface was mainly used as a testbed for showing that results can be displayed.

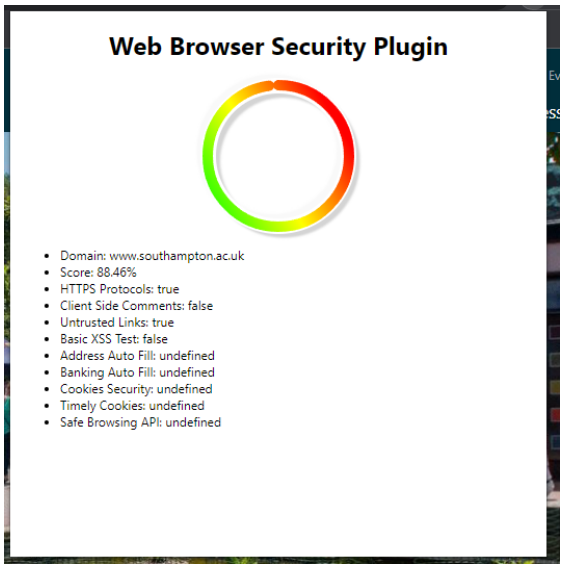

Figure 4.1: Basic User Interface

To improve the user interface, I decided to use Bootstrap to improve the quality of the design to be more like the UI design. I considered using React here but I had a familiarity with Bootstrap and was very useful for designing the layout correctly. The images below show the improved design with a working security rating ring and design. The results are now split into sections: the overall information, first-party results and third-party results. The results of each test are shown as either true or false. The score is calculated by counting up the number of correct scores, getting a percentage and then relating that to an arbitrary grade to display. The rating ring and grade were a key metric here as this part of the design is very readable and very easy to understand. This was also the design that I used in the usability testing.

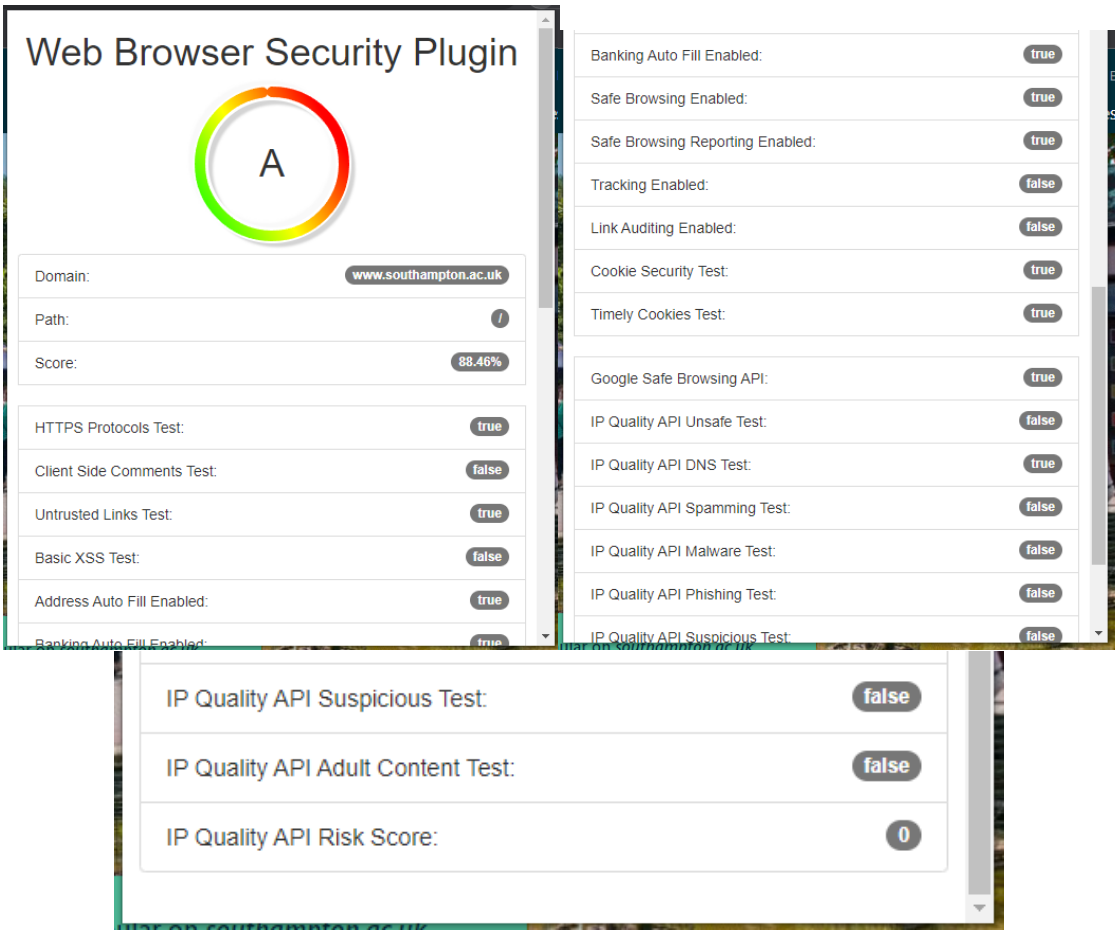

Figure 4.2: Intermediate User Interface

Chapter 5

Testing Strategy and Results

5.1 Usability Testing

The usability testing of my product was critical. This is the kind of solution that the public should be used in the background whenever they have their browser open. It will also be dealing with security which can be a very sensitive area for users. I decided to create a survey sent out to gauge people's internet usage habits and their feelings of safety whilst using their browser of choice. I want to capture people's opinions beforehand and then introduce security extensions and how they affect them. This survey will be sent to a variety of people to gauge a realistic example of the average internet user. To proceed with my survey, I got ethical approval under submission number 72120 from ERGO.

Survey Design

The survey itself was designed in Microsoft Forms because it's a platform I'm familiar with, provides accessibility options can provide special insights on responses. The following areas are what I wanted to focus on.

1. Consent and participation form.
2. Internet browsing habits.
3. Usage of existing solutions.
4. Rating for Existing Solutions compared to my solution.

Analysis of Results

The following are the rankings of the solutions given the results of the end of the survey where the user was asked to compare and contrast the presented solutions.

Individual Rating Ranking	Information Ranking
Browser Audit - 4.19/5	SSL Trust - 1st
SSL Trust - 4.0/5	Browser Audit - 2nd
Google Safe Browsing - 3.43/5	My Solution - 3rd
My Solution - 3.43/5	Avast Scanner - 4th
Avast Scanner - 3.24/5	Google Safe Browsing - 5th
WOT Scanner - 2.14/5	WOT Scanner - 6th

Clarity Ranking	Ease of Use Ranking
Avast - 1st	Avast - 1st
Browser Audit - 2nd	My Solution - 2nd
Google Safe Browsing - 3rd	WOT Scanner - 3rd
My Solution - 4th	Google Safe Browsing - 4th
SSL Trust - 5th	Browser Audit - 5th
WOT Scanner - 6th	SSL Trust - 6th

The results from these rankings showed that informative solutions are overall preferred to accessible solutions. Accessible solutions may be clear and easy to use but without information, they don't serve many purposes. Whilst they ranked highly in these categories, when it comes to the individual rating they fall significantly. Informative solutions often have to make significant trade-offs to achieve the information it requires. This can best be seen with SSL Trust which placed top in the information rankings yet 5th/6th in the accessibility rankings.

My solution ranked best of those that were also Chrome extensions and didn't rank too badly in all categories. My solution doesn't compete on levels of information with SSL Trust and Browser Audit but it can provide a strong middle ground from these results. The biggest takeaway from these tables was how well Browser Audit ranked. This was a very unique solution in the information it provided and ranked 1st overall, and 2nd in information and clarity, only losing out in the ease of use rankings.

Browsing Habits Charts

The following charts have been produced from the first set of questions gauging overall internet browsing habits. From the first two questions about time spent

using the internet, we can see most people spend many hours in a day using the internet an average of 6 hours for all devices to 4 hours for browsers. Most people require this for work and then come home and use it and spend a similar time on devices there. When asked about browsers, Chrome was by far the most popular option for respondents. This was the best platform to create the initial extension. There may be the capacity to expand to another browser later. It was concerning to see such a low proportion of people using website security scanners, but a good number using ad blockers. Except for Google Safe Browsing, not many people had heard of any of the other existing solutions.

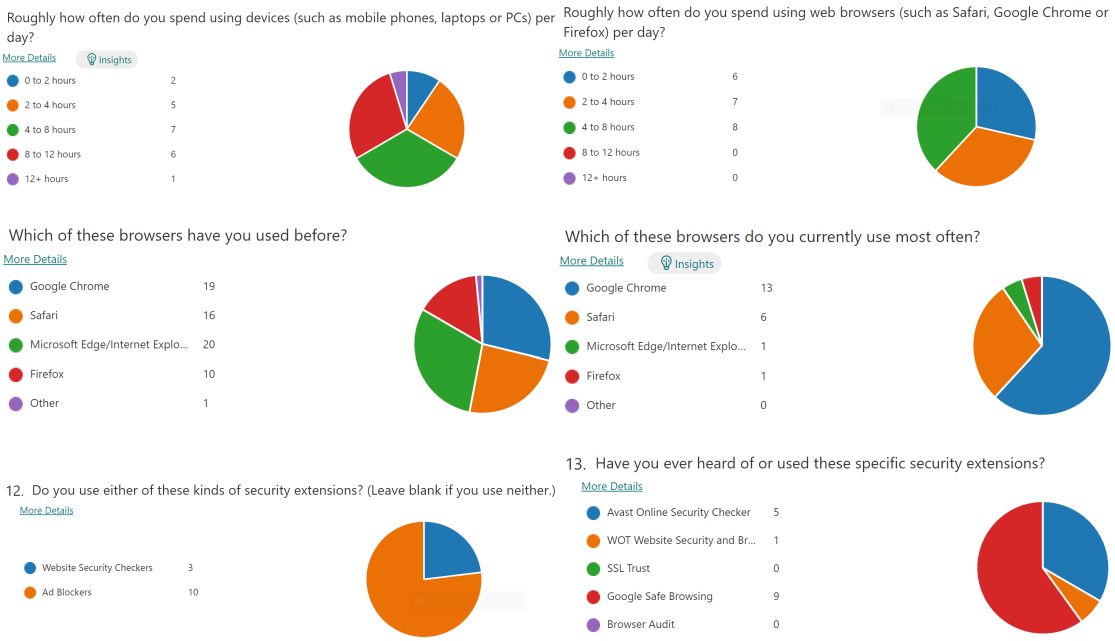

Figure 5.1: Browsing Habits Responses

Testing Feedback

I allowed respondents to leave a few comments worth of feedback for each solution so I could understand what they specifically liked or disliked about it, rather than making assumptions from their ratings. These comments touched on a couple of things: there was mostly praise for the security rating ring, but overall the information was a little over-complicated and wasn't clear what the results meant. As a result of the usability testing, a few changes were made to my user interface. These changes included adding more colour to make the test results clear, adding descriptions to the test so they are easier to understand and improving the scoring system to reflect a more accurate grade.

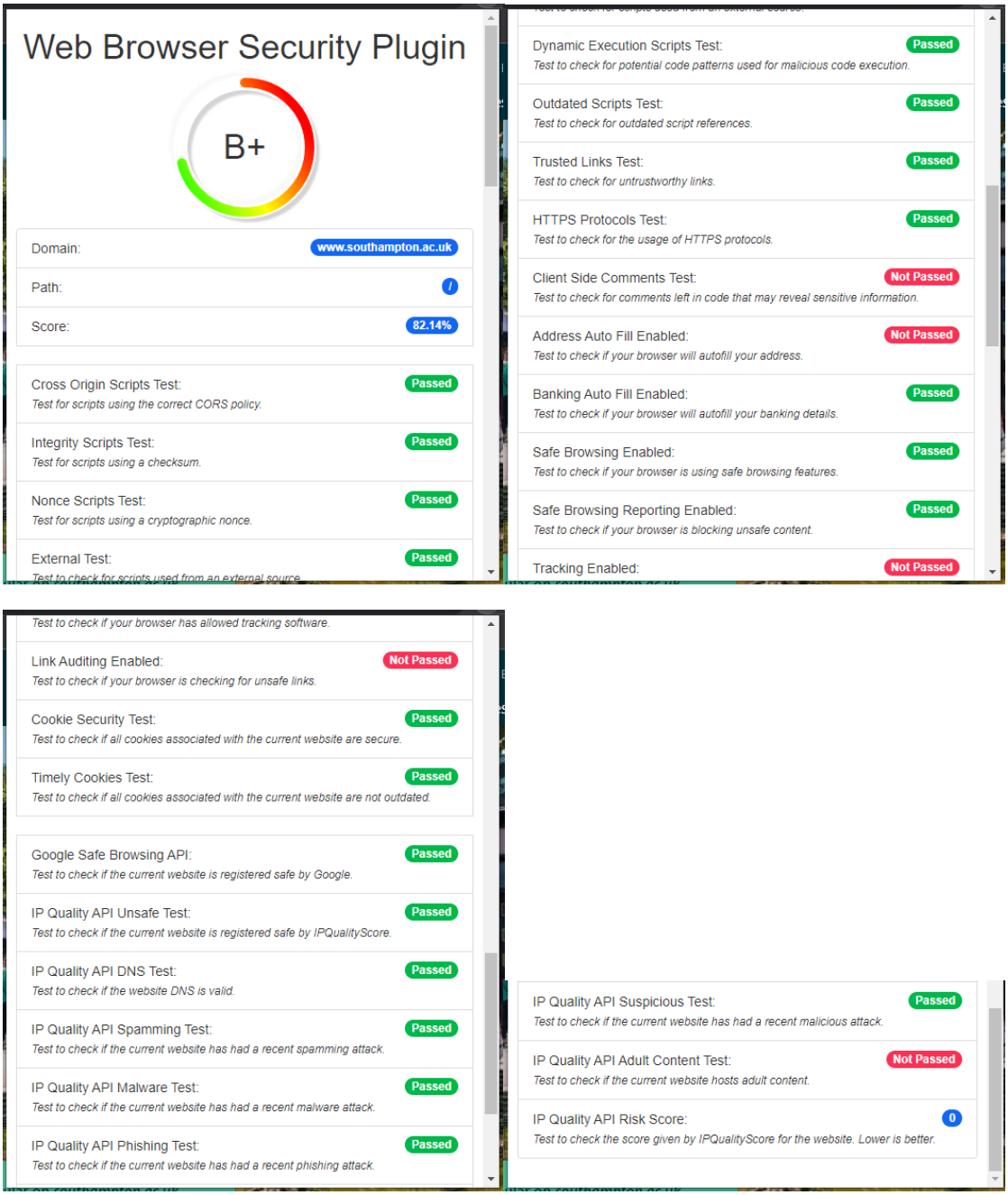

Figure 5.2: Final User Interface

5.2 Comparative Testing

The viability of my solution will depend on how strong it is in comparison to other existing solutions that I have researched during the project. In my research, I could identify a big separation in the types of security scanning tools. They would either choose between being accessible but simple or being over-informative and slow. One of the aims of my project was to fill this area of the market with a tool that was accessible but provided more information than other solutions that used a similar platform. For this reason, I am most interested in comparing my tool to Avast and WOT security extensions. They're Chrome browser extensions and as such have the same access to security information as each other and as my solution.

Comparison to Avast and WOT

The easiest comparative testing I can do was against my solutions' closest competition. Avast scanner and WOT scanner are these two such tools that both use the same platform. Whilst at one point I had planned to just create a Chrome extension, I believe it was a result of using a server on the user's computer that has helped prove its viability over these solutions.

When looking back at the results of the usability testing, my solution came out on top in several tests or second best to Avast when it came to a few other factors. After some of my adjustments to the final design to the one shown during testing, I am sure my solution would be able to outclass both of these two and perhaps many other lesser-known solutions available from the Chrome extension store.

Usability Tests	**My Plugin**	**Avast**	**WOT**
Average Rating	3.43	3.24	2.14
Information Ranking	3rd	4th	6th
Clarity Ranking	2nd	1st	6th
Accessibility Ranking	4th	1st	5th

The three factors I can compare these three plugins is the security rating, the breakdown of this score and additional tools. For the first two, my solution provides a rating that you can differentiate between websites and is understandable for anyone at any level of technological proficiency. Avast gives you a thumbs up and WOT gives an equivalent rating with outdated and biased user reviews.

Neither of these has a breakdown of how these results are met, unlike mine which has a host of tests to backup that score. I believe that in this area of the market, my solution has exceeded to competition. It would now be more important to compare it against those that provide more technical information to see whether the trade-off of accessibility was a worthwhile decision.

Comparison to SSL Trust

During this section of testing, I discovered that the differences between my solution and SSL Trust were quite large. SSL Trust was able to detect weaknesses and vulnerabilities that no other solution I had previously researched could. However, there are many things it does use that I would be very capable of implementing. SSL Trust uses a lot of other third-party tools to check websites that know of website vulnerabilities, blacklists, and spamming attacks that may all be publicly accessible. As I have shown by implementing Google Safe Browsing and IPQualityScore results, I could include these extra data sources in future without much difficulty. On the other side of the coin, SSL Trust can return a lot of information that would be impossible for my tool to collect without using a third-party tool.

SSL Trust is a company that offers SSL/TLS certificates for validated websites. To obtain a certificate, a website would have to give into rigorous checks that would not be accessible from the position my extension sits in. These tests can be specifically targeted and directed at parts of a website's systems that I would not have permission to access. As a result, SSL Trust can look for some vulnerabilities such as Heartbleed, Ticketbleed or LOGJAM. These are notorious vulnerabilities that would be a fantastic bit of information to return to the user, but this isn't feasible with a Chrome extension. Tests like this not only would look like an attack, but it wouldn't be able to return the results quickly and would make my solution slow and inaccessible. This is not the direction I want to take it in and hence it is worth the sacrifice of not providing this information, regardless of how interesting it is. However, I could provide a link to SSL Trust and other alike websites so users don't have to go searching for it on Google and potentially find less trustworthy security tools.

Comparison to Pentest Tools

A tool I discovered late on during my development was Pentest Tools. For a while, I had been searching for a solution that gave very explicit references to weaknesses and vulnerabilities and this was the best tool that I could find that showcased that information. This solution served a very similar purpose to mine but took the approach from a website based platform. Much like other solutions using this

medium, the URL scanner wasn't the only tool from the company but it was freely accessible. In theory, this was the best security tool that I have come across to date. In practice, it suffered from the same limitations as SSL Trust and Browser Audit but with a couple of additional caveats.

The main problem with Pentest Tools was that it offered a free vulnerability scanner than could only be used at most two times a day. This limited how easy it was to test against my solution. There was an extended full scan that could be done as much as you like and could find more vulnerabilities, but this tool was locked behind a paywall. When using the free scan on the Southampton University website, the following vulnerabilities were found. The last two, vulnerabilities CVE-2020-11022 and CVE-2020-11023 were both found by my solution as well. These vulnerabilities could were for identifying outdated jQuery files.

- CVE-2018-16487
- CVE-2021-23337
- CVE-2019-10744
- CVE-2020-8203
- CVE-2020-28500
- CVE-2015-9251
- CVE-2019-11358
- CVE-2020-11022
- CVE-2020-11023

As a solution, Pentest Tools could provide more information than my solution. However, this information could be found quite easily and would be easy to implement ways of scanning for these vulnerabilities. The overall information provided by this tool was the best so far, but the trade-offs in accessibility for monetary gain could not be ignored. I expect most users would find it interesting as a one of solution but would be frustrated by the paywall and move on.

Comparative Testing Takeaways

Whilst I believe I have met my aims with this section, I was disappointed with the overall takeaway of this section of testing. I understood there was a large divide between plugin solutions and program solutions but I think I have previously underestimated the perception of this divide. I believe I have made my solution to be an improvement that similar solutions but don't scratch the surface of the level of information that a solution like SSL Trust can provide. The accessibility of my tool is one of its main selling points, but the level of information it provides

isn't. I believe there may be several explanations for this, but it is apparent that other browser plugins also suffer from them.

5.3 Functional Testing

Next, I needed to test my system's actual functionality. Comparatively, my solution stands up, but it's important to understand how practical it is as a genuine solution. My aim for this section was to find many websites that I believe to be secure and another group of websites I know to be insecure and see what kind of results I get. See below a list of websites I have selected.

Testbed	I initially created a testbed.html file to deliberately add vulnerabilities to see if it could detect them. • Grade: C+ • Score: 67.86 % The testbed allowed me to test things like client-side comments and the many variations of scripts that would be considered insecure. I discovered during this stage that the outdated scripts test would fail if there was no jQuery script at all.
University of Southampton	The website I used to show off my extension during usability testing. I am fairly confident with the security of the website and is not a website you can log in to so there is a little avenue for an attack. • Grade: B+ • Score: 82.14 %
Twitter	A website with very high traffic and likely a very varied demographic. Should be incredibly secure due to its popularity and reputation. • Grade: B • Score: 78.57 % This solution was unsurprisingly quite slow to calculate which might cause frustration for users.

Facebook	Another social media platform that has a less well-renowned reputation. Quite a lot of ads are present and subject to several high-profile cyber attacks in the last few years. In testing, Facebook would not register with my solution due to protocol issues when it came to trying to collect the scripts from the website. Upon fixing it, Facebook received the same score as Twitter • Grade: B • Score: 78.57 %
bWAPP	This was a recommended known buggy web app that I wanted to test my solution on. However, the actual application required an external download so the normal version of the website wasn't as vulnerable as I expected and I was unable to test the application with my extension. • Grade: B • Score: 75.00 %
cryptwalletimport.com	A website I found by using aa419, a fraudulent websites blacklist. My solution rated it at a low C+ grade, as the third-party tools were able to detect recent phishing activity which gave it a much lower score. • Grade: C+ • Score: 67.86 %

Google	I finally wanted to test a normal Google search to check it would work given it's likely to be revisited by a user a lot. I discovered that a Google search would crash my server as it didn't like the links present on the page and failed to try to check them. This is a good warning that given the variety of links online, my solution needs to be more robust to anomalous characters, file types and more.

I noticed that most results deviated as a result of website based vulnerabilities. Browser vulnerabilities always yielded consistent results so the scores for each website will always seem worse or better based on these results. I think that this would need a better breakdown by separating the ratings for website and browser. As a result of this testing, the following tests either failed or responded incorrectly.

Outdated Scripts Test	Supposed to check for outdated jQuery scripts. If no jQuery script is detected, it will fail regardless. Could be expanded to look for other common library scripts and if they're out of date.
Dynamic Script Execution Test	The test itself didn't fail but was unable to collect the scripts for analysing due to a parsing error of the script tags. This may have likely been due to Facebook using PHP.

5.4 Solution Performance

I encountered a few performance issues whilst testing my solution. These were mostly caused by the server not computing fast enough, particularly when interacting with the database. If the user opens the popup too fast after loading a website, it can fail to load the details of the current website and will require reopening the popup after a little bit for it to update.

I also had issues with certain websites taking longer than others where they had a lot of scripts or links on the page. I believe the key solution to these problems could be to improve the database by not storing a record of every single script and link entry as this can fill up the database very quickly and takes a long time to run each of the SQL statements. I believe I also need to set up the solution to work

whilst offline to reduce the amount of information that needs to be computed by the server.

Chapter 6

Critical Evaluation

6.1 Conclusion

The key gap in the market of security solutions was for a tool that was accessible and informative. I discovered that Google Chrome extensions were the most accessible platform and the information provided can ensure users feel secure if presented correctly. My solution has aimed to deliver in both of these areas with varying levels of success with the information provided.

6.2 Successes

I think the biggest overall selling point of my solution is its accessibility. Since the changes were made after the usability testing, the extension looks very presentable and should be easy to understand for people with a range of technical knowledge or none at all. The current website gets a colour coded arbitrary rating that uses a scale that anyone can understand. The more technical detail below shows a full breakdown of how this score is achieved with details about every test used. Another key selling point of my goals is the incorporation of first-party and third-party tools. I have created my vulnerability scanning tools and I have collected the results from other sources to either backup my results or cover areas I was unable to.

When I look through my project goals, I believe I have met all of my primary goals and all but one of my secondary goals. I have also completed the tertiary goal that looks for browser vulnerabilities, an area I would like to revisit. Overall, my solution is a strong option when compared to other solutions like Avast and I believe if a person wanted to look for a similar solution, they may prefer mine over a solution like that. The trade-off for a background server should be acceptable to

get the level of information required.

6.3 Failures

When I consider the successes of this project, the key failure I come across is an unfortunate lack of tests. During the initial research and design, I overestimated how quickly it would be to create an automatic detection system for a test. There were many interesting areas of security risks that I attempted and was unable to provide any kind of result. The most notable attempt was trying to perform automatic XSS tests (cross-site scripting.) XSS is such a notable security risk and discovering these issues manually can be an achievable challenge. However, when it came to automatically testing, I ran into very non-technical problems. These were things like finding the input boxes to fill in when not all input boxes would accept automatically, then finding the corresponding search button to run this XSS was impossible. The nature of Chrome extensions means they refresh quite often so trying to do this kind of test cut it off before it could finish.

I can understand why existing solutions try to remain simple. Most people don't check how secure the current website is and hope their browser keeps them safe. The technical information can be very useful but people either assume that most websites are safe or don't stick around long enough to browse the results of the extension. The gap in the market exists but without a brand behind the product, there is likely to be little demand for this or any solution that fills it.

6.4 Future Extensions

That being said, given more time and planning, there are many areas of this project that would be feasible to improve or expand with.

- Firstly, I would create the background server plus database as an executable program to make it more viable for a shippable product.
- To fill my last secondary goal, I would create an ad blocker. These tools are very popular so would draw more attention to my tool. It may be better to set up this extension as an ad blocker first and a security rating plugin second.
- The database wasn't as significant in my project as I hoped. This could be improved in keep a record of past results better and showcasing that in the extension to identify recent changes to the score.

- Performance was a temperamental metric in my final solution and I would like to readdress how the messaging works. I would also like to set up the tool to display results even whilst online so the user doesn't have to use the background server if they don't want to.
- The project can easily be expanded by creating many more tests for vulnerabilities and this would add much more weight to my project.
- The last of my tertiary goals included MITM detection, VPN integration and more which would make for good final extensions if users needed them.

Bibliography

[1] Nguyen Duc Thai, Nguyen Huu Hieu (2019) *A Framework for Website Security*
https://doi.org/10.1145/3348445.3348456

[2] Laura Falk, Atul Prakash, Kevin Borders (2008) *Analysing Websites for User-Visible Security Design Flaws*
https://doi.org/10.1145/1408664.1408680

[3] Nitirat Tanthavech, Apichaya Nimkoompai (2019) *CAPTCHA: Impact of Website Security on User Experience*
https://doi.org/10.1145/3321454.3321459

[4] Matthew Malloy, Mark McNamara, Aaron Cahn, Paul Barford (2016) *Ad Blockers: Global Prevalence and Impact*
https://doi.org/10.1145/2987443.2987460

[5] Avast (Accessed August 2021) *Avast Online Security Plugin*
https://chrome.google.com/webstore/detail/avast-online-security/gomekmidlodglbbmalcneegieacbdmki?hl=en

[6] WOT (Accessed April 2022) *WOT Website Security and Browsing Protection*
https://chrome.google.com/webstore/detail/wot-website-security-brow/bhmmomiinigofkjcapegjjndpbikblnp

[7] University of Southampton (Accessed April 2022) *University of Southampton Homepage*
https://www.southampton.ac.uk/

[8] SSL Trust (Accessed August 2021) *SSL Trust Website Safety and Security Check Website*
https://www.ssltrust.co.uk/ssl-tools/website-security-check

[9] Google Transparency Report (Accessed April 2022) *Google Safe Browsing and Transparency Report*
https://transparencyreport.google.com/safe-browsing/search?hl=en_GB

[10] Pentest Tools (Accessed April 2022) *Pentest Tools Website Vulnerability Scanner*
https://pentest-tools.com/website-vulnerability-scanning/website-scanner

[11] Browser (Accessed April 2022) *Browser Audit*
https://browseraudit.com/

[12] Google Safe Browsing (Accessed March 2022) *Google Safe Browsing*
https://developers.google.com/safe-browsing

[13] IP Quality Score (Accessed March 2022) *IP Quality Score API*
https://www.ipqualityscore.com/threat-feeds/malicious-url-scanner

[14] Chrome Developer API V3 (Accessed December 2021) *Chrome Developer V3 Migration* https://developer.chrome.com/docs/extensions/mv3/intro/mv3-migration/

[15] Chrome Developers (Accessed November 2021) *Getting Started - Chrome Developers*
https://developer.chrome.com/docs/extensions/mv3/getstarted/

[16] CVE-2012-3714 (Accessed April 2022) *NVD Vulnerability CVE-2012-3714*
https://nvd.nist.gov/vuln/detail/CVE-2012-3714

[17] CVE-2021-21177 (Accessed April 2022) *NVD Vulnerability CVE-2021-21177*
https://nvd.nist.gov/vuln/detail/CVE-2021-21177

[18] CVE-2021-35527 (Accessed April 2022) *NVD Vulnerability CVE-2021-35527*
https://nvd.nist.gov/vuln/detail/CVE-2021-35527

[19] CVE-2022-0807 (Accessed April 2022) *NVD Vulnerability CVE-2022-0807*
https://nvd.nist.gov/vuln/detail/CVE-2022-0807

[20] CVE-2022-21703 (Accessed April 2022) *NVD Vulnerability CVE-2022-21703*
https://nvd.nist.gov/vuln/detail/CVE-2022-21703

[21] CVE-2009-2816 (Accessed April 2022) *NVD Vulnerability CVE-2009-2816*
https://nvd.nist.gov/vuln/detail/CVE-2009-2816

[22] CVE-2008-3843 (Accessed April 2022) *NVD Vulnerability CVE-2008-3843*
https://nvd.nist.gov/vuln/detail/CVE-2008-3843

[23] CVE-2022-25602 (Accessed April 2022) *NVD Vulnerability CVE-2022-25602* https://nvd.nist.gov/vuln/detail/CVE-2022-25602

[24] CVE-2020-15664 (Accessed April 2022) *NVD Vulnerability CVE-2020-15664* https://nvd.nist.gov/vuln/detail/CVE-2020-15664

[25] CVE-2017-7799 (Accessed April 2022) *NVD Vulnerability CVE-2017-7799* https://nvd.nist.gov/vuln/detail/CVE-2017-7799

[26] CVE-2019-16728 (Accessed April 2022) *NVD Vulnerability CVE-2019-16728* https://nvd.nist.gov/vuln/detail/CVE-2019-16728

[27] CVE-2019-11718 (Accessed April 2022) *NVD Vulnerability CVE-2019-11718* https://nvd.nist.gov/vuln/detail/CVE-2019-11718

[28] cve-2020-11023 (Accessed April 2022) *NVD Vulnerability cve-2020-11023* https://nvd.nist.gov/vuln/detail/cve-2020-11023

[29] cve-2020-11022 (Accessed April 2022) *NVD Vulnerability cve-2020-11022* https://nvd.nist.gov/vuln/detail/cve-2020-11022

[30] CVE-2014-3852 (Accessed April 2022) *NVD Vulnerability CVE-2014-3852* https://nvd.nist.gov/vuln/detail/CVE-2014-3852

[31] CVE-2015-4138 (Accessed April 2022) *NVD Vulnerability CVE-2015-4138* https://nvd.nist.gov/vuln/detail/CVE-2015-4138

[32] CVE-2008-4122 (Accessed April 2022) *NVD Vulnerability CVE-2008-4122* https://nvd.nist.gov/vuln/detail/CVE-2008-4122

[33] CVE-2007-6197 (Accessed April 2022) *NVD Vulnerability CVE-2007-6197* https://nvd.nist.gov/vuln/detail/CVE-2007-6197

[34] CVE-2009-2431 (Accessed April 2022) *NVD Vulnerability CVE-2009-2431* https://nvd.nist.gov/vuln/detail/CVE-2009-2431

Appendix A

Original Project Brief

Problem

In today's age, we have such a reliance on the internet, we often don't invest in protecting ourselves from it. Web browsers are one of the most used programs and act as a gateway between us and wherever we choose to visit. It's also very likely the place where users are likely to download a virus, visit an insecure website or give up details they shouldn't't. We need to do more to improve our internet browsing habits. We can only do this by having the appropriate information about website security and how we can protect ourselves.

Goals

My goal is to try to create a web browser security plugin that provides awareness to users about website security whilst also providing a package of security tools that less tech savvy people may be unaware of. This plugin will most likely take the form of a chrome extension as the browser is very common and development of such a plugin is well supported.

Scope

1. Create a basic chrome extension as a starting platform
2. Start adding modular functionality to the extension prioritising what the users feel are the most effective tools
    - Website security rating analyser (comparison against first-party score vs Avast, SSL Trust, etc.)
    - Website security vulnerability detector (code review, malware + phising detection, exploit detection using security APIs)

- Ad blocker (pop up blocker, collecting pop up statistics)

3. Add any extra secondary functionality that go beyond the initial requirements for the plugin
    - Browser security checker (must extend plugin to other browsers)
    - MITM detector
    - Captcha detection / login system security checker
    - VPN service (third-party integration)

Appendix B

Archive Contents

The important files here are popup.html (extension structure), popup.css (extension styling), popup.js (extension display functionality), background.js (extension service worker), server.js (Node.js server), security.db (database) and manifest.json (extension manifest).

- Client Folder
  - CSS Folder - popup.css
  - HTML Folder - popup.html, testbed.html
  - Icons Folder - icon16.png, icon32.png, icon48.png, icon128.png
  - JS Folder - background.js, popup.js
- Server Folder
  - security.db
  - server.js
- Node Modules Folder
- manifest.json
- package.json
- package-lock.json