SECURITY ENHANCER 1.0

Group: **SHA-3**

Satish Muddana - 112504241
Hima Upadhyay - 112516857
Andrew Marrell - 107788946

OBJECTIVE :

The chrome extension ensures the user's security. It has the following features.

1. Detect password reuse: Users, unfortunately, tend to reuse passwords across websites. Whenever one of these websites is compromised, attackers can take advantage of these passwords and use them against different services. The ideal solution to this problem is to stop users from reusing passwords in the first place. Your browser extension should be able to detect when a user is creating a new account on a website and compare the password that the user has selected against all other previously stored passwords. If the password is the same, the extension should warn the user and encourage him to choose a different password.

2. Detect the entering of passwords on the wrong website: Assuming that we have convinced users to use unique passwords, we can now detect whether the user is trying to login to a website with the password of a different website. This should allow us to protect users from falling victim to phishing attacks (e.g. detect that the user is entering her paypal.com password to the attacker.com website).

3. Modify link-clicking behaviour: Some security researchers have argued that most users would be protected if the software would stop them from visiting unpopular websites. Your extension should inspect all links in all webpages visited and for those links that are leading the user outside of the Alexa top 10K websites, the user should be warned if they click on that link. The user should have the option to dismiss the block once (i.e. be allowed to visit that website) or forever (i.e. ask the extension not to bother her next time she visits that particular website).

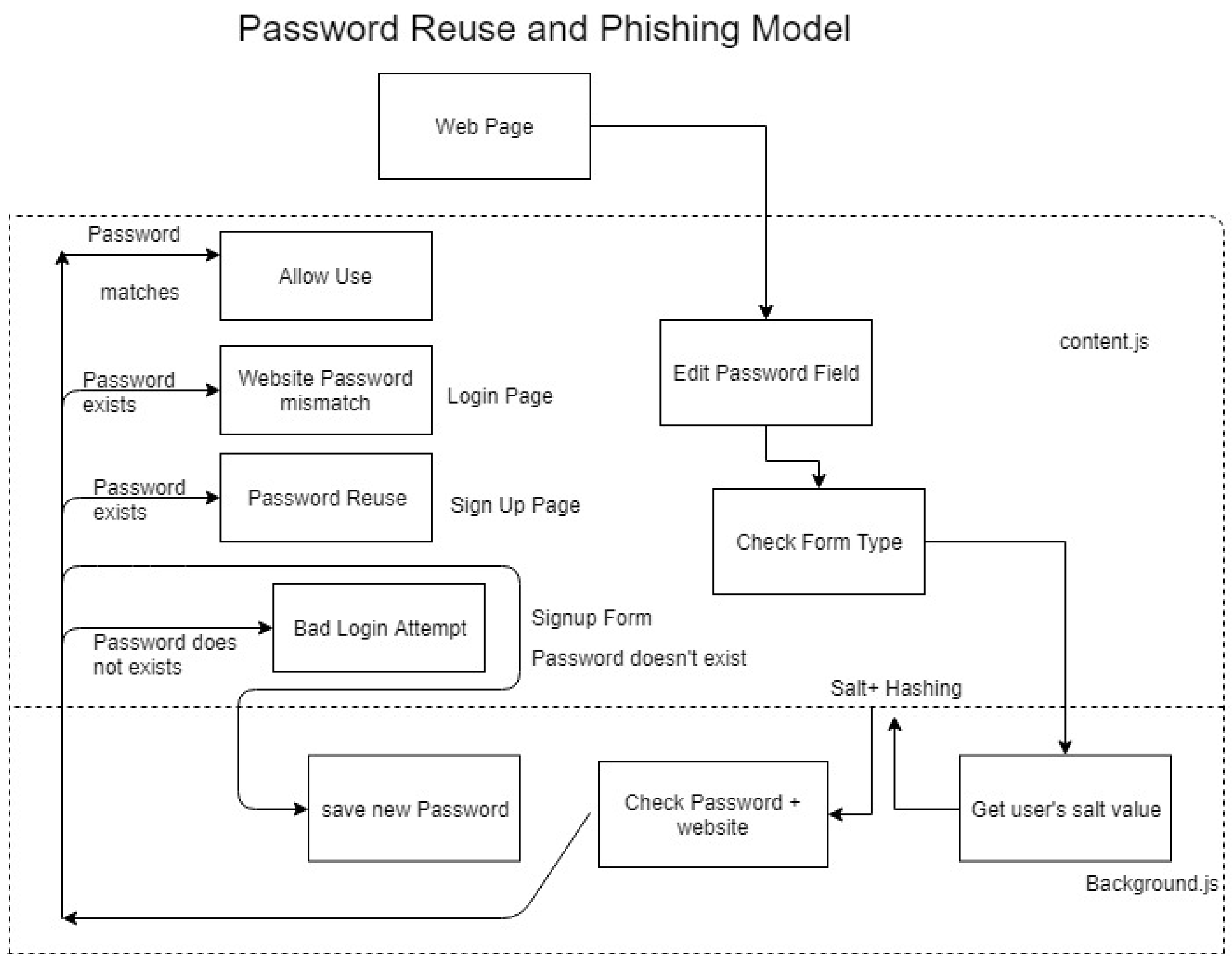
Design and Architecture :

1) Check Password reuse and phishing Attack :

Account creation and login forms are everywhere on the internet. One might think that because they are such an essential feature, there might be a standardized methodology for setting them up. Unfortunately, this is not the case; instead, everyone has their own implementation. Some account creation pages have one password field. Some have two. There are even pages without any password fields. Every form submission button has its own flavour of ‘Sign In’ or ‘Sign Up’ text. Because of this, it is very difficult to develop code that will handle every such form it comes across properly. Knowing this, we decided to take a more general approach: if a form has more than one password field (such as account creation or password change forms) or the button does not contain the word ‘Log’ or phrase ‘Sign In’, we treat it as a ‘Sign Up’ (account creation) page. If these conditions are not met, it is a login form.

With this distinction set, we attach an on lose focus event to every password field on the page. When the field loses focus, we check the type of form, and if there are multiple password fields, that the passwords contained within match each other. We then salt and hash the provided password for protection before comparing it against the passwords we are already tracking for the user, and handle the various potential cases based on these comparisons. If the user falls into one of our error states, such as password reuse, we wipe the password from the input fields to prevent the form submission from going through, trusting that there is sufficient form validation on submission to halt the attempt from going through.

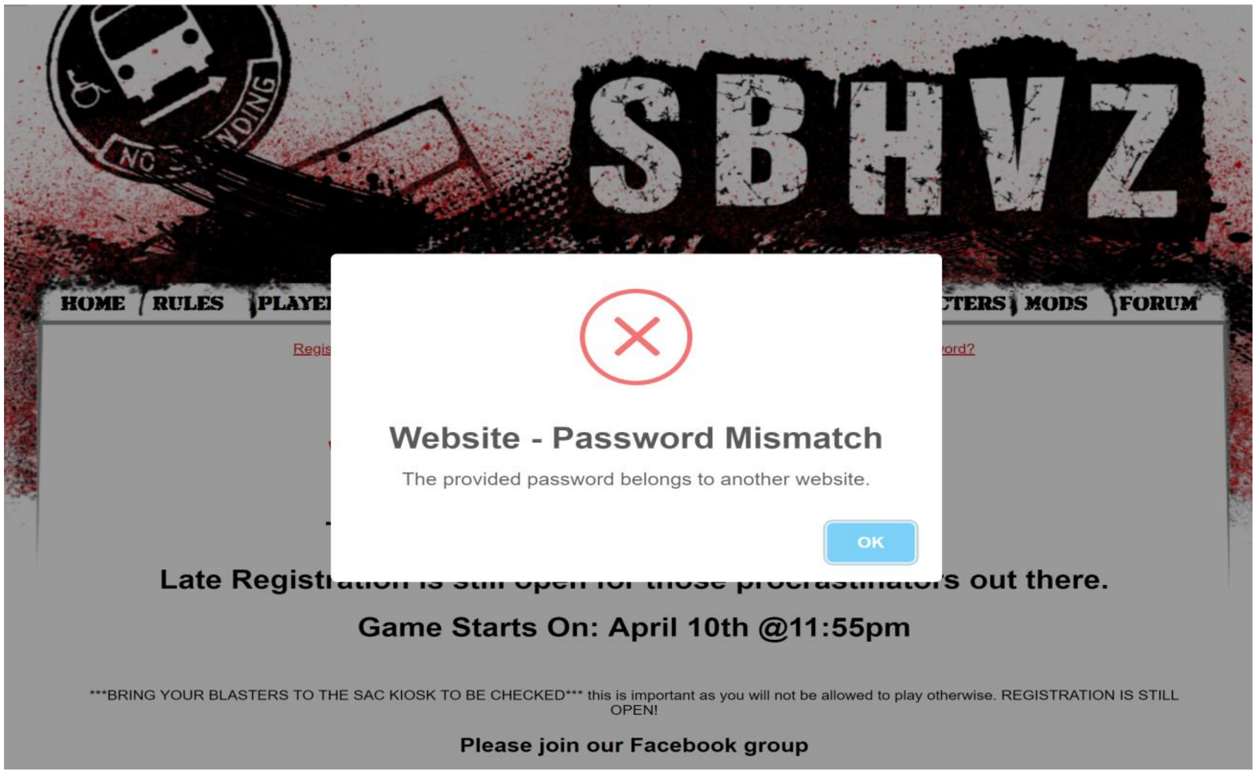
For the hashing, we initially attempted to use SubtleCrypto as a trustworthy built-in library, but discarded it when we realized that it was disabled for http; we then found and utilized implementation of SHA-256, which we have linked to in the references.



This architecture clearly shows the flow of the password reuse and phishing attack. While storing passwords in the background.js, we salt the passwords and then hash with SHA-256 to avoid password attacks on our storage.

Test cases :
When a user signs up for an account, we store that in JSON with URL and hashed password.

1) If the user tries to log in to some other website with an already existing password, alert given is :



Website - Password Mismatch

The provided password belongs to another website.

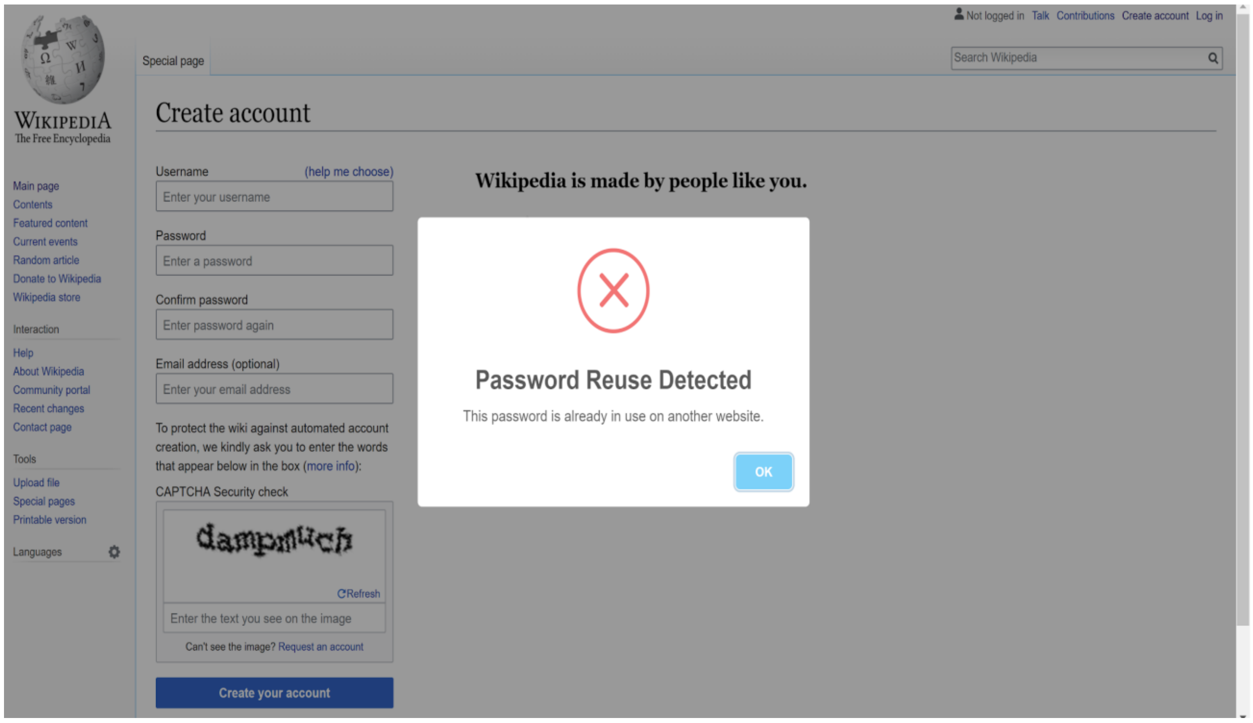
Late Registr… s out there.

Game Starts On: April 10th @11:55pm

BRING YOUR BLASTERS TO THE SAC KIOSK TO BE CHECKED this is important as you will not be allowed to play otherwise. REGISTRATION IS STILL OPEN!

Please join our Facebook group

2) If the user tries to sign up to a new website with an existing password, an alert given is :



Create account

Username (help me choose)

Enter your username

Password

Enter a password

Confirm password

Enter password again

Email address (optional)

Enter your email address

To protect the wiki against automated account creation, we kindly ask you to enter the words that appear below in the box (more info):

CAPTCHA Security check

dampmuch

Enter the text you see on the image

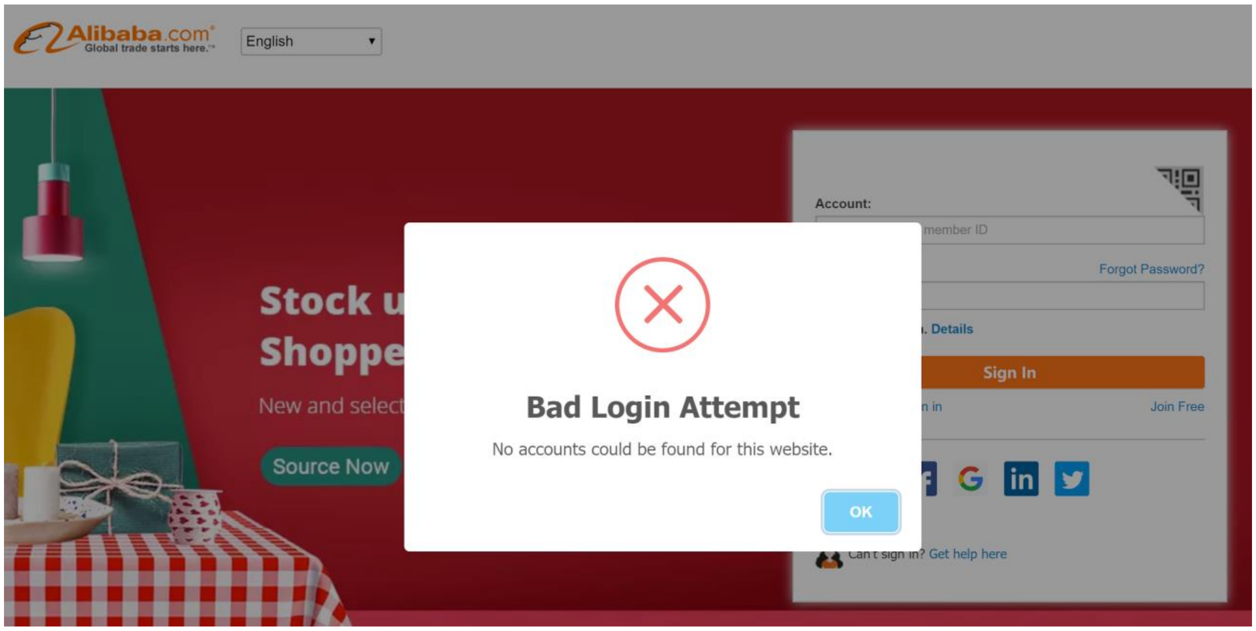
Can't see the image? Request an account

Wikipedia is made by people like you.

Password Reuse Detected

This password is already in use on another website.

3) If the user attempts to log in to a domain they have not signed up for previously, we give an error :


Bad Login Attempt

No accounts could be found for this website.

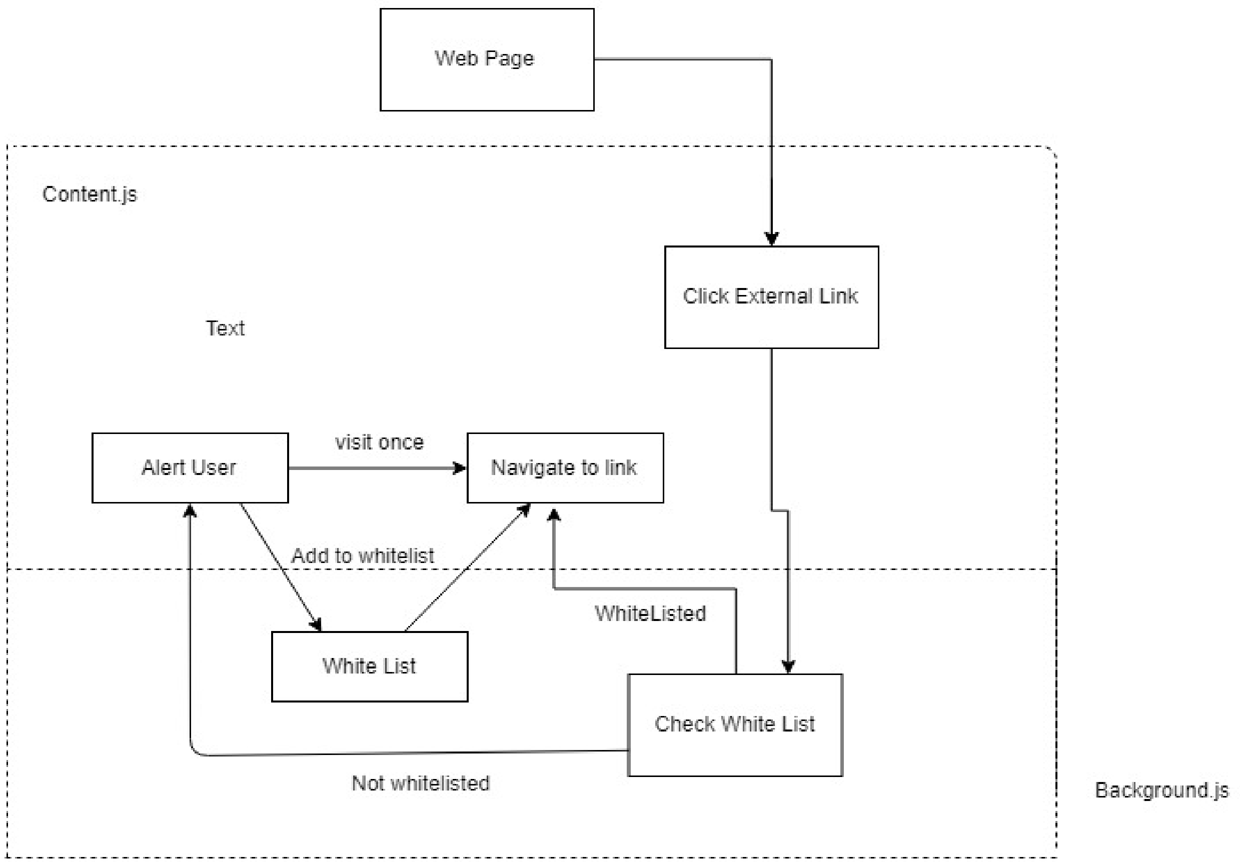
Interesting case during the testing :

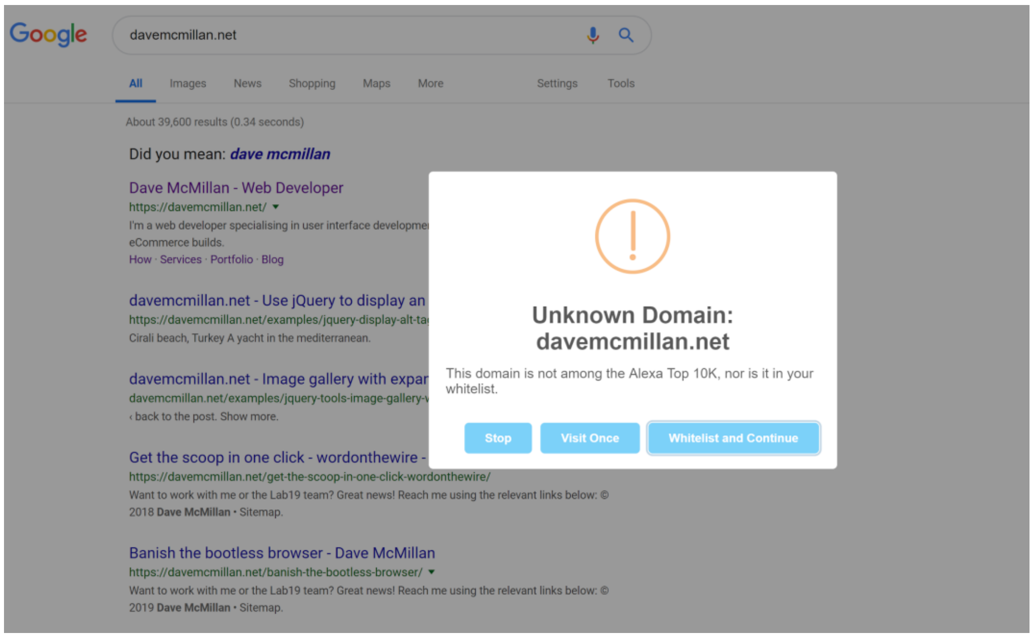
Trying to test password reuse and phishing attack, we came across the scenarios where due to the extensions like Ad Block Plus, we had been given an error like ERR_BLOCKED_BY_CLIENT, On disabling the extension, we were able to use our security extension.

2) **Modify link-clicking behaviour:**

Our approach to domain whitelist checking starts with iterating through all of the links on a page. As we iterate, we test each link to see if it is an external link or not, trusting that if the user has already navigated to a page then they trust the current website, which allows us to ignore internal, blank, and "javascript:" links. We attach a mouse-down event to each external link, which allows us to intercept all clicks; when clicked, we retrieve the hostname for the link's URL and poll our stored whitelists (the Alexa Top 10K, which is preloaded on install, and the user's personal whitelist, if present) to see if it has been whitelisted previously. If it has, we allow navigation to continue. Otherwise, we inform the user that the site has not been whitelisted and prompt them to either stop navigation, visit the site this one time, or whitelist it for future use.
It is worth mentioning that our initial approach was to use the webRequest chrome extension API, which was simple to implement but required fully synchronous code. This at the time was not feasible because we were attempting to utilize chrome storage’s synchronized format, which could only be accessed asynchronously, so we switched our methodology to one more accepting of asynchronous behaviour.

Modify link-clicking behavior





About 39,600 results (0.34 seconds)

Did you mean: ***dave mcmillan***

Dave McMillan - Web Developer
https://davemcmillan.net/
I'm a web developer specialising in user interface developme
eCommerce builds.
How · Services · Portfolio · Blog

davemcmillan.net - Use jQuery to display an
https://davemcmillan.net/examples/jquery-display-alt-ta
Cirali beach, Turkey A yacht in the mediterranean.

davemcmillan.net - Image gallery with expar
davemcmillan.net/examples/jquery-tools-image-gallery-v
‹ back to the post. Show more.

Get the scoop in one click - wordonthewire -
https://davemcmillan.net/get-the-scoop-in-one-click-wordonthewire/
Want to work with me or the Lab19 team? Great news! Reach me using the relevant links below: ©
2018 **Dave McMillan** • Sitemap.

Banish the bootless browser - Dave McMillan
https://davemcmillan.net/banish-the-bootless-browser/
Want to work with me or the Lab19 team? Great news! Reach me using the relevant links below: ©
2019 **Dave McMillan** • Sitemap.

Unknown Domain: davemcmillan.net

This domain is not among the Alexa Top 10K, nor is it in your whitelist.

This domain is not present in the Alexa Top 10K list or the user's personal whitelist, so the extensions ask them how they wish to proceed.

3) **Import Export Data :**

Initially, we wanted to use Chrome's built-in sync storage API to ensure that users would always have access to their data, but we felt that in a real-world scenario, the user might quickly amass more data than the storage quota allows to be synchronized. We, therefore, switched to local storage and made use of our extension’s pop up menu to provide the user with two buttons. The first button reveals a file chooser control which allows users to select the data file they wish to import, and the second button downloads a file containing their currently stored data within the extension. With this menu in place, the user can freely install our extension on as many machines as they wish, and always have access to their up-to-date data.

4) **Storage:**
To overcome the limitation of storage of chrome extension, we opted for the local storage and gave the user an import/export facility by which he can transfer his stored data from one machine to another.

Contribution :

Three of us worked together for higher level architecture and storage choices. Andrew designed and implemented task 3 to Modify link-clicking behaviour and import-export data. Hima did the groundwork of designing and initial coding of the password checking, sign in and log in differentiation, which Satish polished and checked corner cases as well as optimized the code to a great extent. We all did thorough testing and helped each other where we felt stuck. From JSON design, Hash function and event point which needs to be triggered, everyone gave their inputs and we used the best options possible. We thank the professor for his prompt answer to all of our questions related to project work.

SET UP:

1) Download the code from github https://github.com/satish-muddana/security-enhancer
2) Extract the zip file contents
3) Open chrome://extensions and select load unpacked
4) Select the directory containing the extracted files

REFERENCES:

The chrome extension API documentation

Although it functions mostly as an introductory overview of the many available APIs, we learned much from this documentation as we developed our extension.

https://dev.to/aussieguy/reading-files-in-a-chrome-extension--2c03
https://developer.mozilla.org/en-US/docs/Web/API/Body

We found these resources while researching file importing for the Alexa Top 10K whitelist.

https://sweetalert.js.org/guides/
https://unpkg.com/sweetalert@2.1.2/dist/sweetalert.min.js

The Sweetalert framework and minified javascript file. We used sweetalerts to provide informative, customizable messages to the user.

https://stackoverflow.com/questions/21012580/is-it-possible-to-write-data-to-file-using-only-javascript

When we were researching how to export the user data, we came across Niraj's answer, which was very informative.

https://stackoverflow.com/questions/26884140/open-import-file-in-a-chrome-extension

Similarly, Ivan's answer here helped us with writing the code to import the user data file.

https://jquery.com/download/

The jQuery website, where we downloaded the minified script file from. We used jQuery to select and perform operations on links, password fields, buttons, and forms.

https://geraintluff.github.io/sha256/

The SHA-256 implementation we ended up using. Initially, we wanted to use the built-in SubtleCrypto functionality but quickly found that in their desire to ensure proper encryption they disabled all of its functionality on http. Knowing this, we looked for and found what seems to be a trustworthy implementation of the SHA-256 algorithm, figuring it would be less secure to try implementing our own version of the algorithm.