FUNCTIONAL SPEC

WEB BROWSER ADDON FOR DETECTING SQL AND XSS VULNERABILITIES

COLIN BROPHY
CYBERCRIME & IT SECURITY
30/04/21

Contents

INTRODUCTION

Upon doing research for my cyber-security project, I decided to build a web application vulnerability scanner. The application will work as an extension for popular web browser such as Chrome and Firefox. The purpose of the application is to run tests, both well-known attacks and manual, for SQL Injection and XSS, and give a report back to the user as to what payloads worked and what didn't. There have been few previous attempts at a solution such as this on the market, all of which seem to have been abandoned and are now out-dated. I wish to revive these into one solution, which will be thorough enough to be used in a workplace environment, but intuitive enough that a student looking to get a deeper understanding of the topics of SQL Injection and Cross Site Scripting can use the extension with ease.

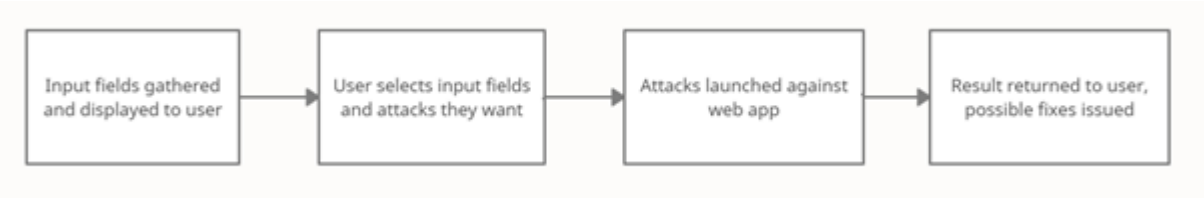
The tool will be compatible on any OS, provided the user is running Chrome or Firefox. This gives the extension a huge target market, where there are very few factors limiting the end user.

This document will outline how the extension will work. I will talk about the design of the app, the technologies used and how the users will interact with it. The idea is that the app will be an invaluable tool for pentesters, college students and web developers looking to strengthen their web apps.

PROJECT OVERVIEW

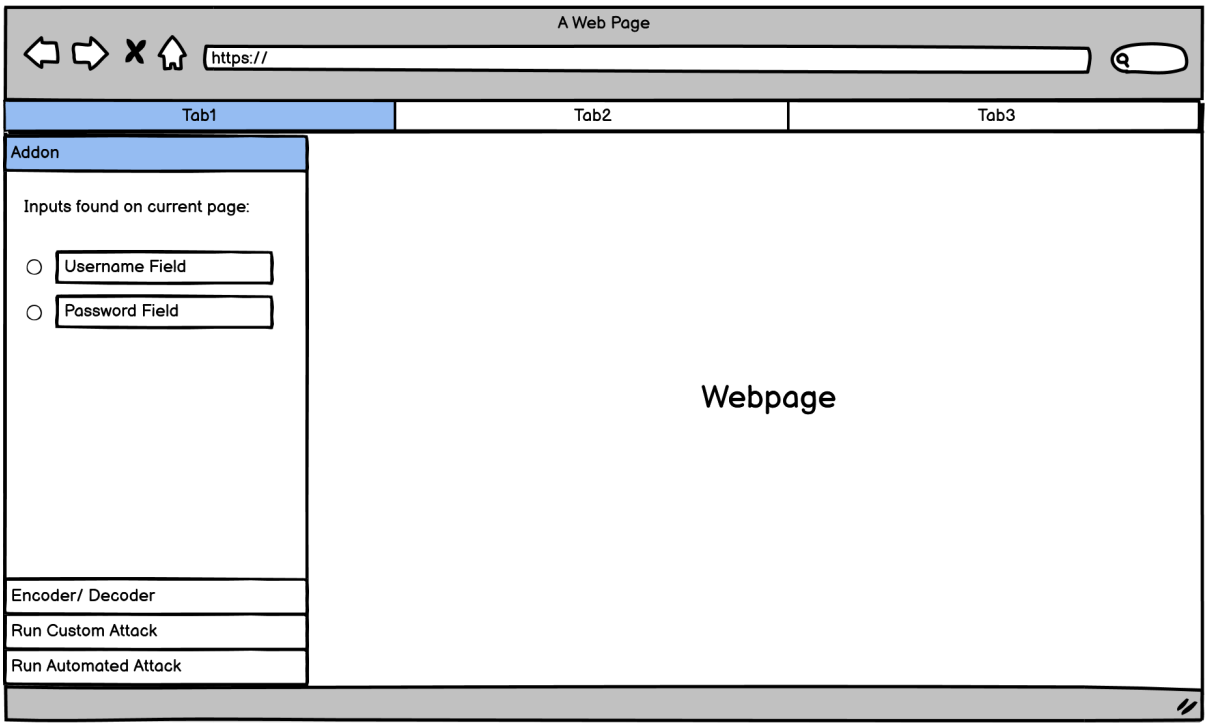
MAIN GOALS

- Extension will be open a sidebar over the web page in the browser.
- The extension will pull in all user input fields to which the user can select for their attack.
- The extension will run the specified tests against the chosen input fields and will return the result.
- The user will be given the option to run tests against all input fields or whatever fields they choose.



I plan for the extension to have added functionality beyond just running automated tests. Other useful tools for my target market would include a Payload/ URL Encoder/ Decoder for encoding payloads that are filtered by some web apps. An education tab would also be of huge benefit to students so they can have quick access to invaluable info while running tests against web apps in labs.

USER INTERFACE



The above photo illustrates my proposed UI. This may change over the course of the development as I realise how I can streamline certain elements. The basic idea is to pop up an overlay over the active web page when the user presses on the extension in the toolbar. The current input fields on the page are pulled in to the extension and can be selected via a radio button for use in testing.

The user will have the option to run a set of common attacks against these input fields, or use they're own if they're trying to test for something within their scope. The extension

will create a list of every input field available on the current web page and will render them inside the application. Each input field rendered within the extension will have a radio button beside it which the user will use to select which input field to run the attacks against. The user will then simply press whether they want to run their own payload or to run the automated set of tests.

TARGET MARKET

Having a set target market in mind is a huge help when creating any sort of application. If you know what the target market likes or dislikes, you can manipulate your application to suit.

My main target market is going to be Web App Developers. Most of their work would be done by switching between their IDE and their browser. They'll make a change, check in the browser, go back to the IDE to make more changes, then back to the browser to test again. Having a third application running for testing the security portion would interrupt their workflow. My idea is to have the application seamlessly integrated into their browser of choice, be it Firefox or Chrome. This way they can simply run tests while they already have the web app open.

Penetration testers would also classify as my target market here. My extension can be another tool to add to their arsenal, quickly running tests when they need them, giving them a basic idea of the kind of security they are dealing with.

Finally, I want my extension to appeal to college students such as myself. During my studies, we had to d SQL Injection and Cross Site Scripting testing ourselves, which can prove difficult for some. My application can help automate this testing, and with an education system built into the app, an unsure student can get answers to whatever questions they have on the topic with the click of a button.

SYSTEM REQUIREMENTS

PROGRAMMING LANGUAGES

HTML

As the extension is completely based in the web browser, and is simply a miniature website running in a popup, use of HTML is absolutely necessary for getting the user interface created.

CSS

You won't often find HTML without some CSS to follow. CSS is paramount in creating an enjoyable user experience and will make or break the application. Good CSS will make the user interface look more professional.

JAVASCRIPT

Some might wince at the thought of JavaScript, but for functionality such as interacting with the DOM, rendering input fields in my extension on the fly, and creating a report for the user, JavaScript is inevitable. I will need to book up on my JavaScript knowledge but it will be a fun challenge.

EXTERNAL LIBRARIES

External libraries are always a huge help for large projects, they can save so much valuable time during the development of the application. I plan on using Bootstrap or Semantic-UI for my CSS to help quickly create a professional looking extension without spending weeks designing from scratch.

I will no doubt need to make used of Chrome and Firefox's API's and in-built JavaScript functions to allow easy access to the DOM and to complete other tasks.

FURPS

I will use the FURPS model to help further outline my application and its demands.

FUNCTIONALITY

- The extension must run in Chrome and Firefox.
- The extension must run on any web page, regardless of text box naming conventions.
- The extension must run a set of payloads and present a report back to the user.
- The extension must include extra functionality (encoding, education) for the user.
- The encoding must work both ways, to encode or decode anything the user chooses.

USABILITY

- The extension must not be too intrusive of the active web page.
- The extension must clearly outline which text box is which from the page when rendered in the extension itself.
- The extension must have a clear layout, the user should intuitively know which button does what.
- The report must be clear as to which payload was run and its result.

RELIABILITY

- The extension must run on any web page, regardless of text box naming conventions.
- The extension must run through each payload in the automation, regardless of errors.

PERFORMANCE

- The report must be given back to the user within 25 seconds.
- A progress bar should be given to the user to show current progress.
- Performance will be greatly impacted by users own network connection and/ or web server speed.

SUPPORTABILITY

- The code will be open source, anyone with the knowledge can edit and upgrade.
- As it is just a mini web page, adding extra functionality is relatively easy.
- Chrome and Firefox extension code are very similar, updates can be made to both easily.

TESTING

Testing will be a huge part of the development of my application as it needs to be able to interact with web pages, read and edit the DOM, and JavaScript errors will need to be fixed as I go along. As testing an application like this on a website that isn't your own, or you haven't been asked to pentest, is illegal, another solution was needed. I have used the "Damn Vulnerable Web App" before during my studies. This runs locally on my machine and can be started up whenever I need it. This can be used for whatever kind of testing I like as it has different pages in the app that are vulnerable to different attacks.

I can use the DVWA until my testing and report creation are working, I can then move on to other websites with their permission, to get some real world testing in.