Vagabond: A Web Browser Project Proposal

Desmond Roberts

October 12, 2023

1 Introduction

The early 1970s saw the emergence of the first personal computers and the mid 1980s saw the explosion of growth that was the release of the internet. Since then, web browsers have become some of the most ubiquitous pieces of software in the entire world. It is impossible for the average computer user to pick up a personal computing device and interact with the internet without a web browser. Therefore, every major platform that exists in 2023 (Google, Apple, Meta, Amazon, Microsoft) chooses to bundle their own proprietary web browser with their operating system. Furthermore, these companies' operating systems make attempts of varying degrees to encourage or force their users into browsing the internet using their proprietary web browser. As stated in a research report by Mozilla, the average computer user claims to know how to install a browser, the vast majority of people never actually install an alternative browser in practice. Modern commercial operating systems are incentivized to prefer their own browsers at the expense of the consumer's choice. Modern commercial operating systems developers also make no effort to facilitate the porting of data from one browser to another. In the most extreme cases, Apple actually bans alternative web browsers that don't use Safari's WebKit browser engine. Historically, this issue came to a head in 2001 in the landmark case United States of America v. Microsoft Corporation, where a judge ruled that the measures that Microsoft took to prevent the uninstallation of Internet Explorer from their operating systems constituted a monopoly and were illegal.

This all goes to show that as commercial operating systems have become larger and more bloated with software over time, web browsers have become an almost indispensable part of any operating system that the average consumer downloads. In fact, it is hard to even imagine a scenario where a user would purposefully delete the web browser from their computer. In this current scenario, I believe that it is safe to say that web browsers are a "part" of the operating system. While technically a web browser is simply another application that runs in user space, its ubiquity and the deliberate attempts of operating systems developers to include them as important parts of the OS separate them from other applications such as, say, Spotify or Discord, which do not come pre-downloaded and do not have special interactions with the proprietary operating system. Similarly, many modern desktop applications are delivered as web apps, or web pages that are loaded by a browser but used like installed applications.

Why is this important? Well, while kernel developers spend much of their time thinking about problems such as scheduling, TLB flushes, memory management, etc. web browsers ARE an important part of the OS they are building. A buggy web browser can also pose a risk to the security of the entire operating system. Take Apple's Safari WebKit browser engine as an example. When security issues have arisen in the past, every single macOS and iOS system will be vulnerable until Apple identifies, fixes, and patches the vulnerability.

Implementing a simple web browser in Python will give considerable insight into not only the implementation details of a web browser but will also increase understanding of how a web browser interfaces with the operating system in order to handle networking. Beyond understanding web browsers from a systems perspective, I believe that their study is a worthwhile endeavor all on their own and will lead to increased understanding in the areas of communication protocols, parsing, the Document Object model, and security concerns.

2 Background

A modern browser consists of millions of lines of code and contains extremely complicated and powerful features. These include a powerful rendering engine, a full networking stack, a virtual machine, a just-in-time compiler, a security sandbox, and many more features. However, the core features of a browser can be simplified to a program that allows a user to see the contents of a web page and discover other web pages. A web browser serves as a mediator between the user and the internet but the operating system serves as a mediator between the web browser and the internet. Ultimately, a web browser makes system calls to establish a socket to send packets using an internet protocol.

3 Proposed Approach and Evaluation Plan

The name I have chosen for my web browser is "Vagabond" because it allows the user to wander around the internet however they please and is also chosen after my obsession with a manga of the same name. With my current level of experience (not even mentioning the mind-bending millions of lines of code that encompasses a modern web browser) implementing many of the impressive features mentioned above is not feasible. However, implementing the basic features of a web browser in a high level language such as Python should not be excessively difficult.

The features I currently plan to implement are:

- get web pages over both HTTP and HTTPS connections
- correctly parse both HTML and CSS and then draw the corresponding objects to the screen
- allow the user to follow hyperlinks to other websites
- have multiple tabs open at once
- maintain a cache of recently visited websites
- allow submission of HTML forms using POST requests
- maintain cookies and address basic web security concerns

4 Expected Resources

I plan to write my web browser in Python. The choice to use this language is based on my familiarity with the language as well as the fact that any machine running python will be able to download the dependencies and run my web browser with little difficulty.

However, the question of which libraries to use in the development of the web browser remains. For example, the Python "requests" library is commonly used to send, receive, parse, and modify HTTP requests. However, Vagabond will avoid this library in order to implement its functionality myself for the sake of more fully understanding the intricacies of handling HTTP requests for a web browser. However, because implementing a GUI is not of interest to this project, I will use the very reliable "tkinter" library in order to implement the front end. Vagabond will also use the "ssl" library in order to support https connections because implementing TLS would be very difficult and does not fit in the scope of this project. Vagabond will also use the "sys" library in order to create and manage sockets. Though it is not certain yet, Vagabond may also need support from the "html" library in order to parse and write HTML.

Some relevant resources for the project include the documentation of the various python libraries (1), resources explaining the DOM (2), detailed explanations of the HTTP protocol such as a complete list of all headers (3), guides for CSS and HTML standards (4), and high level road maps from previous developers who worked on web browsers (5).

1. Python Libraries Documentation
2. Mozilla DOM
3. HTTP 1.0 Documentation and HTTP 1.1 Documentation
4. Mozilla HTML Standards and Mozilla CSS Standards
5. Browser Engineering and Browser From Scratch

5 Milestones

October 13, 2023 - Vagabond should be able to accept a URL as input, parse the necessary information from the URL (scheme, port, hostname, path, etc.), send a HTTP request, receive the response, parse it into its various components (status line, body, etc.), and draw basic shapes and write text to a GUI.

October 20, 2023 - Vagabond should be able to parse the body of the HTTP response. This includes the ability to walk though some HTML code, character by character, and draw the text to the screen. Vagabond will also respond to various keyboard inputs such as the up/down arrow and ctrl+ in order to perform actions on screen.

October 27, 2023 - Vagabond should be able to construct a Document Tree in order to represent the Document Object Model. Vagabond should be able to identify hyperlinks and implement the functionality of following a link once it is clicked on.

November 3, 2023 - Vagabond will implement interpreting some basic CSS.

November 9, 2023 - Vagabond should also support opening multiple tabs and caching recently downloaded web pages.

November 16, 2023 - Vagabond should transition from being a read-only web service to having the ability to handle HTML forms and send information to servers using HTTP POST requests.

November 23, 2023 - Vagabond should begin to address basic web security concerns in an attempt to keep user data private. Vagabond should implement cookie caching. Vagabond should also protect itself against cross-site requests using the same-origin policy.

If time allows there are a myriad of other functionality that could be implemented. Some examples include:

- opening files from the operating system using the “file:///” scheme
- inlining HTML content into the URL using the “data:” scheme
- showing the HTML source using the “view-source” scheme
- support for HTTP compression
- support for error code 300 (redirects)
- use the Cache-Control HTTP header to affect the browser cache
- support for cookie expiration
- support for emojis
- add a back and forward button
- support for URL “fragments”
- benchmarking vagabond’s performance

A modern browser’s capabilities are so extensive that there is no end to extra functionality to be implemented. However, the above options are achievable and could be implemented if the project specifications change over the course of its development.