Introduction

An old friend of mine and I like to refer to our random hobby projects as "squirrels". It's a reference to Dug the dog from the Pixar movie Up. The term sort of implies that there are other things we could be focusing on, but in the moment, we can't help but chase a 🐿️.

The 🐿️ we'll be covering in this post is an online debate platform located at HitchCourse.com. Admittedly no one I know likes the name HitchCourse, but it combines the name of the late Christopher Hitchens - a world-class debater - with the word Discourse. Hitchens + Discourse = HitchCourse.

HitchCourse

The goal for HitchCourse was to create a platform where people could have public debates online without revealing their personal identities. The site doesn't require any personal information (like your email address or phone number) to join. People are free to use whatever user handle they'd like and keep their real-world identities anonymous.

From a software engineering perspective, the question becomes how do you create a public online platform that anyone can anonymously join, and not let it devolve into lowest-common-denominator bots, spam, and hate speech?

Though not thoroughly vetted - HitchCourse.com has only been live for about a week as of this writing - a proof-of-concept using the proof-of-work hashing algorithm has been put in place. The question is will these proof-of-work hashing techniques be enough to deter an uncontrollable onslaught of spam and bot posts? Time will tell.

For now, at the risk of giving away the secret sauce that helps keep the platform secure, let's dive deeper into the proof-of-work hashing techniques that are being used on HitchCourse.com.

Proof-of-Work

The proof-of-work algorithm (made popular by cryptocurrencies like Bitcoin) has no known work-arounds. The only way to generate a proof-of-work hash is to run the algorithm over and over again until you find a hash that satisfies the requirements. For Bitcoin, it's a hash that begins with X number of zeros. For HitchCourse, it's a hash that begins with X number of ones, and ends with two valid lowercase letters.

What makes proof-of-work so powerful is you get to control the initial input, so you can include things like random salts and timestamps - making it so no one can pre-solve the proof-of-work. Every proof-of-work request must be processed in real time. The CPU power must be used. There are no work-arounds.

Architecture

Here's a basic diagram of how proof-of-work is currently being used on the HitchCourse site:

<image redacted>

Non-static pages of the site will request a proof-of-work hash be generated when the page initially loads. The UI will display a loading spinner until the proof-of-work is solved.

A web worker thread is used to solve the proof-of-work because the heavy computation can cause the main browser thread to lag. By offloading the computation to a web worker thread the main browser thread stays fast and responsive during new user inputs (scrolling, typing, expanding/collapsing elements, etc.).

Once solved, the proof-of-work values (input, salt, timestamp, nonce, etc.) will be appended as HTTP headers for all follow-up API calls to the server. The server has middleware to read and verify the proof-of-work headers. Requests that do not contain valid proof-of-work headers can be discarded without further processing.

Difficulty

One of the challenges is finding the right level of difficulty. Finding a hash that begins with one or two one's is pretty easy. Finding a hash that begins with four one's can take a minute, and finding a hash that starts with five one's can take an unreasonably long time to solve.

For now, pages that only send GET requests to the server are configured to use a proof-of-work hash difficulty of two, and pages that POST data to the server require a hash difficulty of three. It's not an exact science, but these seem to be sane settings for the time being. If the server ever comes under unusually high load, the difficulty settings can be increased to help throttle the incoming requests.

Blockchain

One way of increasing the amount of work a client machine needs to perform is to ask it to solve a consecutive series of proof-of-work hashes, often referred to as a blockchain.

Using a blockchain we can keep the hash difficulty low, but increase the amount of work by requiring the client machine to solve the low difficulty hashes multiple times.

On HitchCourse, the proof-of-work hash that gets created when a page loads becomes the first input/block in the page's proof-of-work CAPTCHA chain.

In the video below, by the time the user enters the final "ef" combination, their machine has already generated four proof-of-work hashes (each two character placeholder is generated by proof-of-work).

<video redacted>

Conclusion

It remains to be seen if the proof-of-work CAPTCHA techniques implemented on HitchCourse.com will prove to be effective.