Security Assessment

Abuse and Misuse cases

1. Package management hijacking:

   The repositories that host the dependencies are vulnerable to a malicious actor assuming control over one of the dependencies and uploading malicious code that eventually makes it into our build. NPM has had this happen, whereas Cargo has not, but both rely on the same fundamental approach to security, so both remain vulnerable.

2. Docker image hijacking:

   Just like package management hijacking, a malicious actor could assume control over an image from DockerHub that we rely on to create our development and production environment. This could lead to having our infrastructure compromised.

3. Leaking API credentials:

   Our twitter credentials are all handled on the backend, but the code is hosted in a public github account, so anyone could access them. Conceptually, an attacker can't do much with them, as there is a whitelist of urls that the twitter login could redirect to. This means that an attacker couldn't use our credentials for their own site. Our weather API token is in the code, and it is present in the frontend. This means that an attacker could take our token, and then exhaust the number of free requests we get per day, causing our weather components to fail to work.

4. No utilization limit on API usage:

   There is no limiting function that prevents a user from sending millions of POST requests, creating millions of rows in our database, leading to significant degredation of service.

5. No DDOS protection:

   Our application is just a single server that could not withstand the load generated by a botnet sending millions of requests per second. The backend application being asynchronous, does mitigate the threat of DDOS, because very little resources are used per connection. The fact that most endpoints require authentication, and the cryptography ensuring their security is fast, means that most of these requests would get rejected immediately, meaning the impact should be low. The backend, even running on the single core VM that it has, could still easily service at least 10,000 requests per second, which makes it slightly resilient to small botnet DDOS attacks.

Main areas of vulnerability for your project

- JSON Web Tokens (JWTs), while cryptographically secure, do have some pain points when dealing with key revocation. There is no easy way to terminate a session that relies on JWTs for authentication. Because JWTs are all generated and verified using the same cryptographic key, if you want to invalidate a session, you have to rotate the key. But as consequence, all user sessions for everyone else using the app also become invalidated, making it massively inconvenient to perform these rotations.

Ruled out security concerns

An incomplete list of stuff we covered and don't consider a threat

- SSL. We set up SSL using LetsEncrypt. This prevents our requests and responses from being readable by anyone.
- SlowLoris. The underlying HTTP implementation in our backend (Hyper) has built-in timeouts that prevent an attacker from sending big requests very slowly, tying up resources. Also, the asynchronous nature of the framework means that resources are only consumed when a request delivers bytes to the server, not when a connection is open and held indefinitely.
- SQL injection. Diesel, the query builder used by the backend has built-in protections against SQL injection.

Tools

We talked to Krutz about this, and he told us not to do any write-up on security tools utilized.