2023-04-26-be-mountain

Research Paper

Cody Schliebe

JSON Web Tokens

JSON web tokens (JWT) are an open standard that define a way for securely transmitting data between parties as a self-contained JSON object. Signed JSON tokens have three components separated by dots: the header, the payload, and the signature. A typical JWT could look like the following:

AAAAAAA.BBBBBBBB.CCCCCCCC

The header contains information about the token itself, divided into two parts: the type of the token (which is JWT), and the signing algorithm being used, which we will see in the signature. The payload contains the claims, which are statements about an entity (usually the user), and some additional data. There are three types of claims that can be issued: registered claims (predefined claims that are recommended but optional, containing information such as the issuer, expiration time, audience, etc.), public claims (these can be defined at will but should be defined in the IANA JSON Web Token Registry to avoid collisions), and private claims (custom claims created to share information between the consenting parties.

The signature is a way to take the encoded header, payload, a secret, the algorithm specified in the header, and sign it. This is done so the user can verify that the information wasn't changed during transmission, and with tokens containing a private key, it can verify that the sender is who they say they are.[1]

Hashing and Salting in Encryption

Information security is very important when working with the internet, so much information is encrypted to prevent unauthorized 3rd parties from seeing it. Encryption is the process of using an algorithm to transform information into a mess of seemingly random characters that can only be decoded using a key that only the intended users have. Hashing is a method of encryption that takes a string of data of any size and outputs a string of a predetermined length. It doesn't matter if the original string is someone's first name or the text of the bible, the hash output will be the same length for both. Hashing is one-way, so you can't just figure out the original data by looking it (or running another hash, etc.). One cool use for this is that, when you create login credentials with a company, they never have to store your password on a hackable server somewhere, they just put your password through this hash encryption and store the output value.

While the same input through a hash will always result in the same output, sometimes people pick really weak passwords (like 1234567, or "password" for example). That's where salting can be useful. The process of salting means inserting random data before the data to be encrypted before running it through the hash. If even one character is put before the password that's input into the hash, the output will be completely different. If a hacker has access to a database of hashes, they could potentially crack a hash value and extract a password using hash tables. Hash tables are collections of common passwords

that hackers can run through a hash to see if the output matches the hash value they possess. If a password is salted, however, it basically creates a much more complex password. What started out as 1234567 could become:

029283094F09W2G09188BA1234567

which would produce a very different hash value and be much more difficult for a hacker to parse. While 1234567 is probably in their hash table, 029283094F09W2G09188BA1234567 almost certainly is not.[2]

1 https://jwt.io/introduction/

2 https://www.comparitech.com/blog/information-security/encryption-hashing-salting/