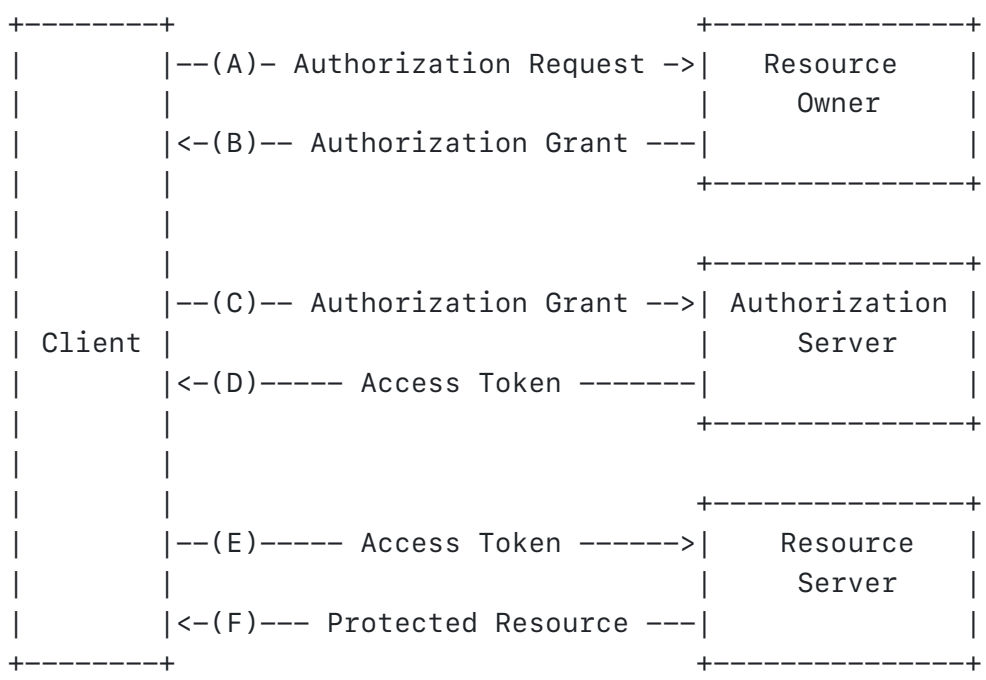
OAuth (An Example PKE Article)

Please see this article on creating a Personal Knowledge Encyclopedia (PKE) for more information. This article is an example.

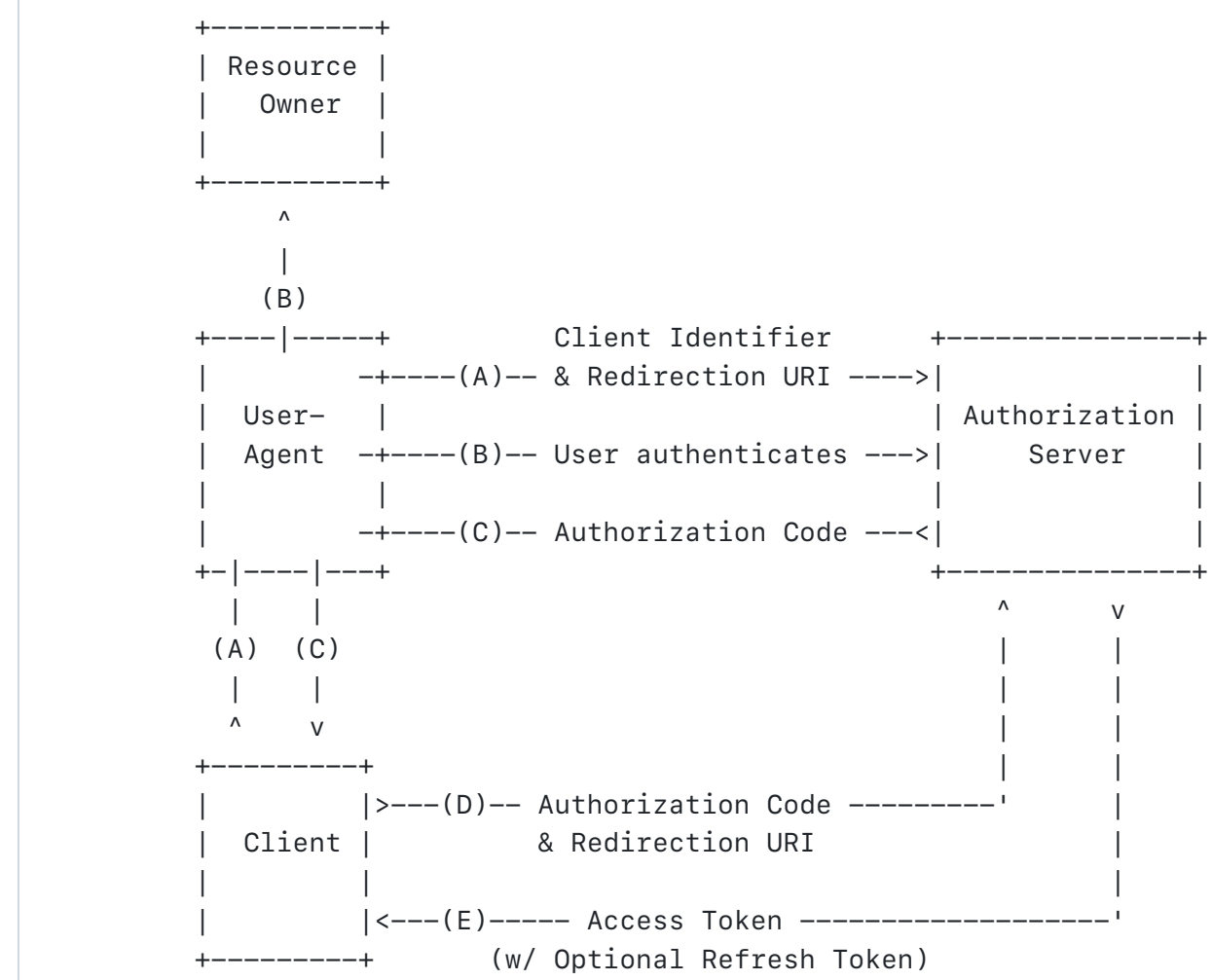
OAuth is a standard that enables a user (resource owner) to allow a third-party application (client) access to protected data (hosted on a resource server). OAuth is designed so that the third party application doesn't need to know the the user's credentials to access the data.

Typical OAuth2 flow



High level flow from RFC 6749*

Before the high-level flow above can be executed, the authorization server must somehow know about the client. In the RFC, this is called "client registration"* . The details of this process are largely left up to the service hosting the resource and ability to authenticate and authorize access to the resource. For example, the service may provide a web page for users to authenticate clients with the service. On another extreme, a client could literally contact the service owner in person and the service owner configures the server to understand the appropriate details about the client.



When the client exchanges the authorization code for the access token, it may need to provide a client secret* to further provide confidence the client is legitimate. Note that some services may allow "Single-Page Apps"* which do not require using a secret provided by the authorization server during registration time, but they do generate a secret dynamically for each request.

Example

A simple example of using an OAuth2 workflow is to enable a properly authenticated and authorized user / resource owner (Geet Duggal) to allow a third-party application (geetduggal.com client) to issue an API call to Github on his behalf. This API call will eventually trigger a Github action to append a line of text to file. Note that in this process, there are no tokens stored anywhere in the client code.

- **Client.** geetduggal.com
- **Resource owner.** Geet Duggal
- **Authorization and resource server(s)** Github

First, the user (Geet Duggal) needs to be able to login to Github via Github's authorization endpoint:

```
<form id="geetForm">
    <label for="geetInput">Enter your Geet:</label>
    <input class="input-box" type="text" id="geetInput" name="geetInp
    <button class="button" type="submit">Submit</button>
    <button class="button" id="loginButton">Login with GitHub</buttor
</form>
<script>
    const clientId = 'Ov23lijztEAKMo8ij7cx';
    const redirectUri = 'https://geetduggal.com/callback.html';

    document.getElementById('loginButton').addEventListener('click',
        const githubAuthUrl = `https://github.com/login/oauth/authori
        window.location.href = githubAuthUrl;
    });

// ...
```

After Geet successfully authenticates, 'https://geetduggal.com/callback.html' (the client) is provided with an authorization code that can be exchanged for a token. (You can see how this is desirable from a security perspective because the tokens can be relatively short-lived and potentially revoked in a more granular fashion).

```
<script>
    const params = new URLSearchParams(window.location.search);
    const code = params.get('code');

    console.log('Received code:', code);

    if (code) {
        fetch('https://geetduggal.netlify.app/.netlify/functions/exchange
            method: 'POST',
            headers: {
                'Accept': 'application/json',
                'Content-Type': 'application/json',
            },
            body: JSON.stringify({ code }),
        })
        .then(response => {
            console.log('Response received:', response);
            if (!response.ok) {
                throw new Error('Network response was not ok');
            }
            return response.json();
        })
        .then(data => {
            console.log('Data received:', data);
            if (data.access_token) {
                localStorage.setItem('githubToken', data.access_token);
                window.location.href = '/';
            } else {
                alert('Failed to authenticate');
            }
        })
        .catch(error => {
            console.error('Error during fetch:', error);
            alert('Error during authentication: ' + error.message);
        });
    } else {
        alert('No code provided');
    }
</script>
```

The token is exchanged using a serverless function on Netlify without exposing the client secret:

```
exports.handler = async (event) => {
    if (event.httpMethod === 'OPTIONS') {
        // Handle CORS preflight request
        return {
            statusCode: 204, // No Content
            headers: headers,
            body: '',
        };
    }

    try {
        const { code } = JSON.parse(event.body);
        const clientId = process.env.GITHUB_CLIENT_ID;
        const clientSecret = process.env.GITHUB_CLIENT_SECRET;

        const response = await fetch('https://github.com/login/oauth/acce
            method: 'POST',
            headers: {
                'Accept': 'application/json',
                'Content-Type': 'application/json',
            },
            body: JSON.stringify({
                client_id: clientId,
                client_secret: clientSecret,
                code: code,
            }),
        });

        const data = await response.json();
        console.log('GitHub response data:', data);

        return {
            statusCode: 200,
            headers: headers,
            body: JSON.stringify({ access_token: data.access_token }),
        };
    } catch (error) {
        console.error('Error fetching access token:', error);
        return {
            statusCode: 500,
            headers: headers,
            body: JSON.stringify({ error: 'Internal Server Error' }),
        };
    }
};
```

The token is then stored in the browser local storage for use to call the Github API to initiate a Github action.

References

- ^ "RFC 6749" Figure 1, RFC 6749 Section 1.2. Protocol Flow
- ^ "client registration", RFC 6749 Section 2. Client Registration
- ^ "client secret" RFC 6749 Section 2.3.1 Client Password
- ^ "Single-Page Apps" OAuth2 Simplified, Single-Page Apps
- ^ "authorization code flow", RFC 6749 Section 4.1 Authorization Code Grant