Best practices for creating an OAuth app

In this article

Follow these best practices to improve the security and performance of your OAuth app.

Use a GitHub App instead

If possible, consider using a GitHub App instead of an OAuth app. In general, GitHub Apps are preferred over OAuth apps. GitHub Apps use fine-grained permissions, give the user more control over which repositories the app can access, and use short-lived tokens. These properties can harden the security of your app by limiting the damage that could be done if your app's credentials are leaked.

Similar to OAuth apps, GitHub Apps can still use OAuth 2.0 and generate a type of OAuth token (called a user access token) and take actions on behalf of a user. However, GitHub Apps can also act independently of a user.

For more information about GitHub Apps, see "About creating GitHub Apps."

For more information about migrating an existing OAuth app to a GitHub App, see "Migrating OAuth apps to GitHub Apps."

Use minimal scopes

Your OAuth app should only request the scopes that the app needs to perform its intended functionality. If any tokens for your app become compromised, this will limit the amount of damage that can occur. For more information, see "Authorizing OAuth apps."

Secure your app's credentials

With a client secret, your app can authorize a user and generate user access tokens. These tokens can be used to make API requests on behalf of a user.

You must store your app's client secret and any generated tokens securely. The storage mechanism depends on your integrations architecture and the platform that it runs on. In general, you should use a storage mechanism that is intended to store sensitive data on the platform that you are using.

Client secrets

If your app is a website or web app, consider storing your client secret in a key vault, such as Azure Key Vault, or as an encrypted environment variable or secret on your server.

If your app is a native client, client-side app, or runs on a user device (as opposed to running on your servers), you cannot secure your client secret. You should use caution if you plan to gate access to your own services based on tokens generated by your app, because anyone can access the client secret to generate a token.

User access tokens

If your app is a website or web app, you should encrypt the tokens on your back end and ensure there is security around the systems that can access the tokens. Consider storing refresh tokens in a separate place from active access tokens.

If your app is a native client, client-side app, or runs on a user device (as opposed to running on your servers), you may not be able to secure tokens as well as an app that runs on your servers. You should store tokens via the mechanism recommended for your app's platform, and keep in mind that the storage mechanism may not be fully secure.

Use the appropriate token type

OAuth apps can generate user access tokens in order to make authenticated API requests. Your app should never use a personal access token or GitHub password to authenticate.

Make a plan for handling security breaches

You should have a plan in place so that you can handle any security breaches in a timely manner.

In the event that your app's client secret is compromised, you will need to generate a new secret, update your app to use the new secret, and delete your old secret.

In the event that user access tokens are compromised, you should immediately revoke these tokens. For more information, see "OAuth Authorizations."

Conduct regular vulnerability scans

You should conduct regular vulnerability scans for your app. For example, you might set up code scanning and secret scanning for the repository that hosts your app's code. For more information, see "About code scanning" and "About secret scanning."

Choose an appropriate environment

If your app runs on a server, verify that your server environment is secure and that it can handle the volume of traffic that you expect for your app.

Use services in a secure manner

If your app uses third-party services, they should be used in a secure manner:

- Any services used by your app should have a unique login and password.
- Apps should not share service accounts such as email or database services to manage your SaaS service.
- Only employees with administrative duties should have admin access to the infrastructure that hosts your app.

Add logging and monitoring

Consider adding logging and monitoring capabilities for your app. A security log could include:

- Authentication and authorization events
- Service configuration changes
- Object reads and writes
- User and group permission changes
- Elevation of role to admin

Your logs should use consistent timestamping for each event and should record the users, IP addresses, or hostnames for all logged events.

Enable data deletion

If your app is available to other users, you should give users a way to delete their data. Users should not need to email or call a support person in order to delete their data.