OAuth2.0, Digital Certificates

CS 390 – Web Application Development

J. Setpal

November 27, 2023

Outline

1. Why it's Worth Your Time
2. Persistent Session
3. OAuth 2.0
4. Digital Certificates
5. ETC

Outline

1. Why it's Worth Your Time
2. Persistent Session
3. OAuth 2.0
4. Digital Certificates
5. ETC

WIWYT – OAuth2.0, Digital Certificates

- OAuth allows us to access information from a 3rd party, without risking privacy.

WIWYT – OAuth2.0, Digital Certificates

- OAuth allows us to access information from a 3rd party, without risking privacy.
- Certifcates allow us to maintain encrypted client-server communication.

Outline

1. Why it's Worth Your Time
2. Persistent Session
3. OAuth 2.0
4. Digital Certificates
5. ETC

Session Store (last time, I promise)

Sessions are not persistent by default. When the node server is restarted, the server is reset.

We can setup persistence using a **Session Store**. The default implementation of `MemoryStore` includes persistence, but is not meant for production.

It only runs a *single* thread, *leaks* memory, and does *not* scale well.

Instead, we can use **MongoDB** as the session store using `connect-mongo`.

Let's Setup a NoSQL Database!

If you can view this screen, I am making a mistake.

Outline

1. Why it's Worth Your Time
2. Persistent Session
3. OAuth 2.0
4. Digital Certificates
5. ETC

What is OAuth?

Scenario:

- We're building a web service.

What is OAuth?

Scenario:

- We're building a web service.
- We don't have the time/resources to develop our own authorization system.

What is OAuth?

Scenario:

- We're building a web service.
- We don't have the time/resources to develop our own authorization system.
- We need to communicate with a third party service for information about the user.

What is OAuth?

Scenario:

- We're building a web service.
- We don't have the time/resources to develop our own authorization system.
- We need to communicate with a third party service for information about the user.

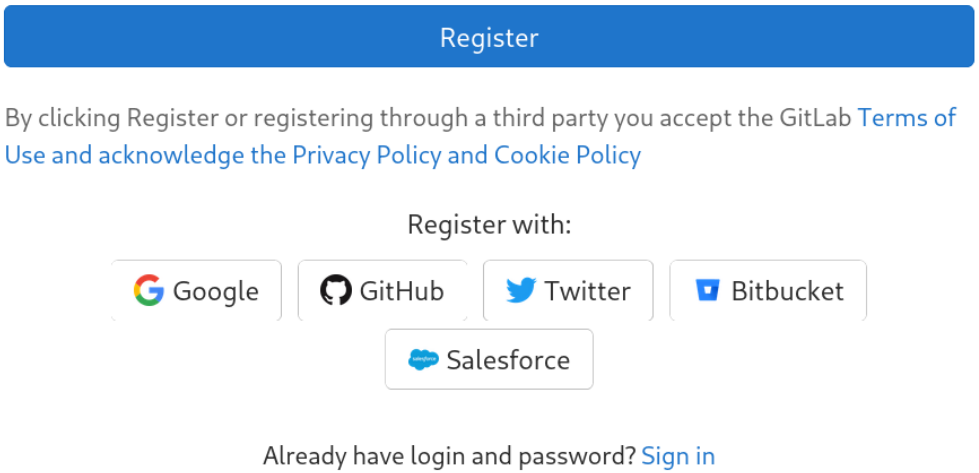
Is there a silver bullet solution for it all?

What is OAuth?

Scenario:

- We're building a web service.
- We don't have the time/resources to develop our own authorization system.
- We need to communicate with a third party service for information about the user.

Is there a silver bullet solution for it all?



<-- right here!

What is OAuth?

OAuth is a secure mechanism for **delegated authorization**.

What is OAuth?

OAuth is a secure mechanism for **delegated authorization**.

It works by *redirecting* the authorization request to the target 3rd party.

What is OAuth?

OAuth is a secure mechanism for **delegated authorization**.

It works by *redirecting* the authorization request to the target 3rd party.

If the authorization is successful, the 3rd party sends us a confirmation with a token, and we can perform our authorized requests.

What is OAuth?

OAuth is a secure mechanism for **delegated authorization**.

It works by *redirecting* the authorization request to the target 3rd party.

If the authorization is successful, the 3rd party sends us a confirmation with a token, and we can perform our authorized requests.

Importantly:

- Our access is **temporary** and **restricted** (eg. we can't update the password and take over the account).

What is OAuth?

OAuth is a secure mechanism for **delegated authorization**.

It works by *redirecting* the authorization request to the target 3rd party.

If the authorization is successful, the 3rd party sends us a confirmation with a token, and we can perform our authorized requests.

Importantly:

- Our access is **temporary** and **restricted** (eg. we can't update the password and take over the account).
- User credentials are secure.

The OAuth Workflow

Here's how we setup an OAuth2.0 Workflow:

a. Register an application with the 3rd Party Server.

The OAuth Workflow

Here's how we setup an OAuth2.0 Workflow:

a. Register an application with the 3rd Party Server.
b. When requested, redirect the authorization request to the 3rd party.

The OAuth Workflow

Here's how we setup an OAuth2.0 Workflow:

a. Register an application with the 3rd Party Server.
b. When requested, redirect the authorization request to the 3rd party.
c. Await the callback from the 3rd party.

The OAuth Workflow

Here's how we setup an OAuth2.0 Workflow:

a. Register an application with the 3rd Party Server.
b. When requested, redirect the authorization request to the 3rd party.
c. Await the callback from the 3rd party.
d. If the authorization is granted, send a request to the authorization server.

The OAuth Workflow

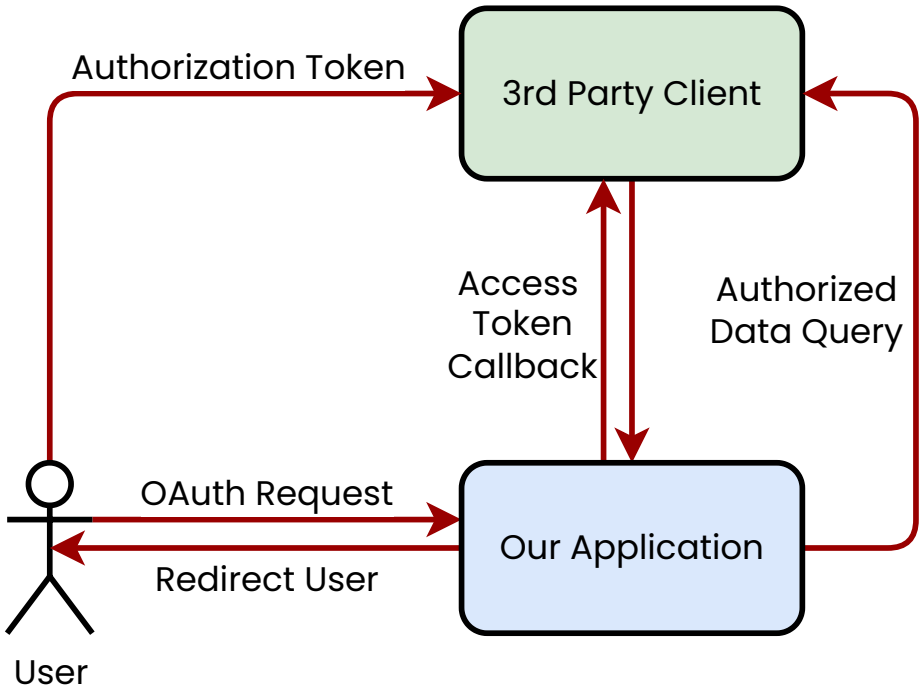
Here's how we setup an OAuth2.0 Workflow:

a. Register an application with the 3rd Party Server.
b. When requested, redirect the authorization request to the 3rd party.
c. Await the callback from the 3rd party.
d. If the authorization is granted, send a request to the authorization server.
e. Use the access token to access authorized API functions!

The OAuth Workflow

Here's how we setup an OAuth2.0 Workflow:

a. Register an application with the 3rd Party Server.
b. When requested, redirect the authorization request to the 3rd party.
c. Await the callback from the 3rd party.
d. If the authorization is granted, send a request to the authorization server.
e. Use the access token to access authorized API functions!



Let's Implement OAuth!

If you can view this screen, I am making a mistake.

Outline

1. Why it's Worth Your Time
2. Persistent Session
3. OAuth 2.0
4. Digital Certificates
5. ETC

Symmetrical and Asymmetrical Encryption

Here's a sample encryption workflow:

Symmetrical and Asymmetrical Encryption

Here's a sample encryption workflow:

Q: Is this secure from intrusion?

Symmetrical and Asymmetrical Encryption

Here's a sample encryption workflow:

Q: Is this secure from intrusion?
A: Not really! Anyone on the same network can sniff keys, and security is thwarted.

Symmetrical and Asymmetrical Encryption

Here's a sample encryption workflow:

Q: Is this secure from intrusion?
A: Not really! Anyone on the same network can sniff keys, and security is thwarted.

We can improve this by making the keys *asymmetrical* – but this requires us to setup an accompanying security policy.

Public Key Encryption

We setup two keys:

a. Public Key

b. Private Key

Public Key Encryption

We setup two keys:

a. Public Key

b. Private Key

These are really the same! We just set one as private, and one as public.

Public Key Encryption

We setup two keys:

a. Public Key

b. Private Key

These are really the same! We just set one as private, and one as public.

The workflow is as follows:

a. We first send a request to the server, to ask for their public key.

Public Key Encryption

We setup two keys:

a. Public Key
b. Private Key

These are really the same! We just set one as private, and one as public.

The workflow is as follows:

a. We first send a request to the server, to ask for their public key.
b. We then encrypt our session symmetric key with the private key, and send it to the server.

Public Key Encryption

We setup two keys:

a. Public Key
b. Private Key

These are really the same! We just set one as private, and one as public.

The workflow is as follows:

a. We first send a request to the server, to ask for their public key.
b. We then encrypt our session symmetric key with the private key, and send it to the server.
c. The server decrypts the key, and can now read information without sending plaintext.

Public Key Encryption

We setup two keys:

a. Public Key

b. Private Key

These are really the same! We just set one as private, and one as public.

The workflow is as follows:

a. We first send a request to the server, to ask for their public key.

b. We then encrypt our session symmetric key with the private key, and send it to the server.

c. The server decrypts the key, and can now read information without sending plaintext.

Q: Is this secure from intrusion?

Public Key Encryption

We setup two keys:

a. Public Key
b. Private Key

These are really the same! We just set one as private, and one as public.

The workflow is as follows:

a. We first send a request to the server, to ask for their public key.
b. We then encrypt our session symmetric key with the private key, and send it to the server.
c. The server decrypts the key, and can now read information without sending plaintext.

Q: Is this secure from intrusion?
A: Still no! If an intruder can inject their *own* public key instead of the server, there is no way for the client to know. Security thwarted.

Certificate Authorities

A certificate authority is a trusted (eg. `https://letsencrypt.org/`) entity that stores and signs digital keys (certificates).

Certificate Authorities

A certificate authority is a trusted (eg. `https://letsencrypt.org/`) entity that stores and signs digital keys (certificates).

The certificate authority signs, or verifies the certificate of a given server.

Certificate Authorities

A certificate authority is a trusted (eg. `https://letsencrypt.org/`) entity that stores and signs digital keys (certificates).

The certificate authority signs, or verifies the certificate of a given server.

Q: But what about verifying the certificate authorities' key? Is that safe?

Certificate Authorities

A certificate authority is a trusted (eg. `https://letsencrypt.org/`) entity that stores and signs digital keys (certificates).

The certificate authority signs, or verifies the certificate of a given server.

Q: But what about verifying the certificate authorities' key? Is that safe?
A: We **recursively verify certificates**, until we leverage a pre-installed root certificate, that is self-signed by the local machine.

Outline

1. Why it's Worth Your Time
2. Persistent Session
3. OAuth 2.0
4. Digital Certificates
5. ETC

Thank you!

Have an awesome rest of your day!

Slides: `https://cs.purdue.edu/homes/jsetpal/slides/oauth,certs.pdf`

If anything's incorrect or unclear, please ping `jsetpal@purdue.edu`
I'll patch it ASAP.