lowball

Release 1.0

unknown

Oct 02, 2021

CONTENTS:

What is it? Lowball is designed to add simple endpoint level RBAC to your Flask based API services.

What does it do? Lowball is, at its core, a wrapper around Flask, designed to add authentication and permission management features to Flask's already powerful and modular implementation. Lowball was developed to support three key needs:

1) Easy to use route level RBAC controls.
2) Abstracted authentication providers and databases for easier integration with your operating environment.
3) Ecosystem of 1 - n microservices leveraging a common authentication authority.

Lowball implements this with three components

1) *Route RBAC Enforcement*
2) *Authentication Providers*
3) *Authentication Databases*

Together, these components allow a developer to produce 1 - n microservices that are able to integrate with your existing authentication infrastructure and utilize database technologies of your choice.

Continue reading to find out more!

CHAPTER

ONE

INSTALLING LOWBALL

lowball has been tested to work with only Python 3.6+

1.1 Pip

```
pip install lowball
```

1.2 From Source

```
git clone https://github.com/EmersonElectricCo/lowball
cd ./lowball
pip install -r requirements.txt
python setup.py install
```

CHAPTER

TWO

QUICKSTART

This guide will lead you through the basics of launching a lowball service using the built in authentication provider and authentication database.

The intent of this guide is to give you a quick introduction to the concepts lowball implements.

Note: The authentication database and provider used in this tutorial are the defaults for lowball. They are great for use in a development environment. However, it is highly recommend to only use these for development and NOT in a production environment. Once you are ready to deploy your application(s) we highly recommend changing out the authentication components for other *Available Authentication Providers* and *Available Authentication Databases*.

2.1 Before You Begin

Be sure you have installed the lowball library through your method of choice. For more detail see *Installing lowball*

2.2 Basic lowball App

A minimal lowball service looks like this:

```
from lowball import Lowball, require_admin

app = Lowball()

@app.route("/hello", methods=["GET"])
@require_admin
def hello_world():
    return {"hello":"world"}, 200


if __name__ == '__main__':
    app.run()
```

That's it, if we were to run this application with the below command, we would have a lowball application running using the builtin default authentication provider and authentication database. These components enable to us limit access to our hello world app to only users with the admin role assigned.

```
python3 app.py
```

To to access our */hello* endpoint we will need to have a token with the *admin* role associated with it. This can be achieved by making a request to the lowball builtin */builtins/auth* endpoint to get a token. Because we are using the builtin authentication provider with the default config we do this via the following curl request:

```
curl -i -X POST -H "Content-Type: application/json" http://localhost:5000/builtins/auth -
↪-data "{\"username\": \"admin\", \"password\": \"nimda\"}"
```

which will return a similar response to:

```
{
  "token": "eyJ0e...jjk",
  "token_data": {
    "cid": "admin",
    "r": [
      "admin"
    ],
    "cts": "2021-04-28 14:44:59",
    "ets": "2021-04-28 15:44:59",
    "rcid": "admin",
    "tid": "SbH7opPxReMbyZZr"
  }
}
```

Our successful call to the auth endpoint returned a few things. First and foremost is the token which is what we will use in subsequent calls to other endpoints with RBAC controls associated with them.

In addition to the token itself, we we are given some additional meta data describing the token. This data is what is stored in the authentication database. No actual authentication data is stored in the database including the tokens themselves, passwords, secrets, etc.

cid Client ID associated with the token. This is the "user" according to the authentication provider.

r An array of roles issued to the token

cts Creation / issue time of the token

ets Expiration time of the token

rcid Requesting Client ID or the Client ID that requested the token. This will either be the same as the cid or an administrator that issued the token on the clients behalf.

tid Unique token ID

Note: The default password for the builtin authentication provider is *nimba*. It is highly recommend to overwrite this password. See *Builtin Basic Authentication Provider* for best practices.

Now that we have the token, we can now call our hello endpoint with the following curl command:

```
curl -i -H "Authorization: Bearer eyJ0e...jjk" http://localhost:5000/hello
```

```
{
  "hello" : "world"
}
```

That's it! You now have the basics for getting started with endpoint based RBAC controls and lowball. Read on to learn details on the authentication providers, authentication databases, configs, and more…

CHAPTER

THREE

ROUTE RBAC ENFORCEMENT

Lowball implements role-based access controls (RBAC) at the endpoint level via a handful of route decorators that make it easy for you as a service developer to add RBAC enforcement at the endpoint level without disrupting the structure of your project.

The enforcement decorators use the token's declared roles to validate the authorization for the transaction.

The following sections describe the various RBAC decorators and their function.

3.1 Require Authenticated User

require_authenticated_user allows access to an endpoint for any user who provides a valid token regardless of the token's roles.

Example

```
from lowball import Lowball, require_authenticated_user
...
@app.route("/launch", methods=["GET"])
@require_authenticated_user
def view_upcoming_launches():
    return {...}, 200
```

In the above example, any valid token will be able to access the endpoint.

3.2 Require Any of These Roles

require_any_of_these_roles allows any token with at least one of the roles in the provided list access to the given endpoint.

Example

```
from lowball import Lowball, require_any_of_these_roles
...
@app.route("/launch/<id>", methods=["GET"])
@require_any_of_these_roles(['lead','manager','audit'])
def view_launch_details(id):
    return {...}, 200
```

In the above example, any token that has at least a *lead*, *manager* or *audit* role granted to it will be allowed to access the endpoint.

3.3 Require All of These Roles

require_all_of_these_roles allows only a token possessing all of the specified roles to access the endpoint.

Example

```
from lowball import Lowball, require_all_of_these_roles
...
@app.route("/launch", methods=["POST"])
@require_all_of_these_roles(['manager','certified_specialist'])
def launch_the_rocket():
    return {"hello":"world"}, 200
```

In the above example, only a token that has both a *manager* and *certified_specialist* role assigned to it will be able to access the endpoint.

3.4 Require Admin

require_admin is a convenience decorator for requiring a token to have the *admin* role assigned to it. This is the equivalent of:

```
@require_all_of_these_roles(['admin'])
```

Example

```
from lowball import Lowball, require_admin
...
@app.route("/reboot", methods=["POST"])
@require_admin
def reboot_the_system():
    return {...}, 200
```

CHAPTER

FOUR

INCLUDED ROUTES

GET /builtins/status

get basic status information about authentication provider state of the service

Example request:

```
GET /builtins/status HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – reports status of service including name and whether there is an enabled auth provider

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "name": "string",
    "auth_provider_initialized": true,
    "auth_db_initialized": true
}
```

POST /builtins/auth

obtain a token (login)

Example request:

```
POST /builtins/auth HTTP/1.1
Host: example.com
Content-Type: application/json

{}
```

Status Codes

- 200 OK – successful login

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "token": "string",
    "token_data": {
        "cid": "string",
        "r": [
            "string"
        ],
        "cts": "string",
        "ets": "string",
        "rcid": "string",
        "tid": "string"
    }
}
```

GET /builtins/auth

get information about current authenticated token

Example request:

```
GET /builtins/auth HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – successfully return of token data

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "cid": "string",
    "r": [
        "string"
    ],
    "cts": "string",
    "ets": "string",
    "rcid": "string",
    "tid": "string"
}
```

DELETE /builtins/auth

revoke current authenticated token (logout)

Status Codes

- 204 No Content – successfully revoked current token

GET /builtins/auth/tokens

get listing of all tokens for current authenticated client

Query Parameters

- **roles** (*array*) –
- **exclude_expired** (*boolean*) –

Example request:

```
GET /builtins/auth/tokens HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – tokens returned successfully

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "cid": "string",
        "r": [
            "string"
        ],
        "cts": "string",
        "ets": "string",
        "rcid": "string",
        "tid": "string"
    }
]
```

POST /builtins/auth/tokens

create a token for the current authenticated client or target client

Example request:

```
POST /builtins/auth/tokens HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "client_id": "string",
    "roles": [
        "string"
    ],
    "token_life": 1
}
```

Status Codes

- 201 Created – token created

Example response:

```
HTTP/1.1 201 Created
Content-Type: application/json
```

(continues on next page)

(continued from previous page)

```
{
    "cid": "string",
    "r": [
        "string"
    ],
    "cts": "string",
    "ets": "string",
    "rcid": "string",
    "tid": "string"
}
```

DELETE /builtins/auth/tokens

revoke tokens for the current authenticated clients

Status Codes

- 204 No Content – tokens removed successfully

GET /builtins/auth/tokens/all

get tokens for all clients in auth database

Query Parameters

- **roles** (*array*) –
- **exclude_expired** (*boolean*) –
- **client_ids** (*array*) –

Example request:

```
GET /builtins/auth/tokens/all HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – tokens returned successfully

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "cid": "string",
        "r": [
            "string"
        ],
        "cts": "string",
        "ets": "string",
        "rcid": "string",
        "tid": "string"
    }
]
```

`DELETE /builtins/auth/tokens/all`
delete tokens for all clients in auth database

Status Codes

- 204 No Content – tokens deleted successfully

`POST /builtins/auth/tokens/cleanup`
initiate cleanup operation for expired tokens

Status Codes

- 204 No Content – the cleanup operation was initiated

`GET /builtins/auth/tokens/{token_id}`
get information on a specific token by token_id

Parameters

- **token_id** (*string*) –

Example request:

```
GET /builtins/auth/tokens/{token_id} HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – successfully return of token data

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "cid": "string",
    "r": [
        "string"
    ],
    "cts": "string",
    "ets": "string",
    "rcid": "string",
    "tid": "string"
}
```

`DELETE /builtins/auth/tokens/{token_id}`
revoke a token by token id

Parameters

- **token_id** (*string*) –

Status Codes

- 204 No Content – token deleted successfully

`GET /builtins/auth/clients`
get authenticated client information from auth provider

Example request:

```
GET /builtins/auth/clients HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – the client data is returned successfully

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "client_id": "string",
    "roles": [
        "string"
    ]
}
```

`POST /builtins/auth/clients`
authenticated client update own information in auth provider

Example request:

```
POST /builtins/auth/clients HTTP/1.1
Host: example.com
Content-Type: application/json

{}
```

Status Codes

- 200 OK – the client was updated successfully

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "client_id": "string",
    "roles": [
        "string"
    ]
}
```

`POST /builtins/auth/clients/create`
create a client in the auth provider

Example request:

```
POST /builtins/auth/clients/create HTTP/1.1
Host: example.com
Content-Type: application/json
```

(continues on next page)

(continued from previous page)

```
{}
```

Status Codes

- 201 Created – the client was created successfully

Example response:

```
HTTP/1.1 201 Created
Content-Type: application/json

{
    "client_id": "string",
    "roles": [
        "string"
    ]
}
```

`POST /builtins/auth/clients/register`
allow a client to register itself in the auth provider

Example request:

```
POST /builtins/auth/clients/register HTTP/1.1
Host: example.com
Content-Type: application/json

{}
```

Status Codes

- 201 Created – the client was created successfully

Example response:

```
HTTP/1.1 201 Created
Content-Type: application/json

{
    "client_id": "string",
    "roles": [
        "string"
    ]
}
```

`GET /builtins/auth/clients/all`
return list of all clients in the auth provider

Query Parameters

- **roles** (*array*) –

Example request:

```
GET /builtins/auth/clients/all HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – the clients were returned successfully

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "client_id": "string",
        "roles": [
            "string"
        ]
    }
]
```

`GET /builtins/auth/clients/{client_id}`
get auth provider information for a specific client

Parameters

- **client_id** (*string*) –

Example request:

```
GET /builtins/auth/clients/{client_id} HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – the client data was returned

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "client_id": "string",
    "roles": [
        "string"
    ]
}
```

`POST /builtins/auth/clients/{client_id}`
change auth provider information for the client

Parameters

- **client_id** (*string*) –

Example request:

```
POST /builtins/auth/clients/{client_id} HTTP/1.1
Host: example.com
Content-Type: application/json

{}
```

Status Codes

- 200 OK – the client data was returned

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "client_id": "string",
    "roles": [
        "string"
    ]
}
```

`DELETE /builtins/auth/clients/{client_id}`
delete the client from the auth provider

Parameters

- **client_id** (*string*) –

Status Codes

- 204 No Content – the client was deleted

`DELETE /builtins/auth/clients/{client_id}/roles`
remove all roles from a the client

Parameters

- **client_id** (*string*) –

Status Codes

- 204 No Content – all roles removed

`POST /builtins/auth/clients/{client_id}/roles/{role}`
add a specific role to the client

Parameters

- **client_id** (*string*) –
- **role** (*string*) –

Status Codes

- 204 No Content – the role was added

`DELETE /builtins/auth/clients/{client_id}/roles/{role}`
remove a specific role from the client

Parameters

- **client_id** (*string*) –

- **role** (*string*) –

Status Codes

- 204 No Content – the role was removed

`POST /builtins/auth/clients/{client_id}/enable`
enable the client in the auth provider

Parameters

- **client_id** (*string*) –
- **role** (*string*) –

Status Codes

- 204 No Content – the client was enabled

`POST /builtins/auth/clients/{client_id}/disable`
disable the client in the auth provider

Parameters

- **client_id** (*string*) –
- **role** (*string*) –

Status Codes

- 204 No Content – the client was disabled

CHAPTER

FIVE

AUTHENTICATION PROVIDERS

Lowball Authentication Providers are the interface for your application(s) to the Identity Provider of your choice.

Currently, lowball only supports one Authentication Provider at a time for an application.

5.1 Available Authentication Providers

The following are known existing lowball Authentication Providers. If you have written one and would like it included, please submit a PR or contact us to have it added!

Table 1: Available Authentication Providers

Builtin Basic Authentication Provider	Builtin Provider to the lowball library useful in a development environment.
lowball-ldap-authprovider	Utilize your existing LDAP / Active Directory system as your Authentication Provider.

5.1.1 Builtin Basic Authentication Provider

The builtin Authentication Provider is meant to be minimal, have no external dependencies and help a developer get up and running quickly. It could be used in a small production environment that has limited authentication needs.

It provides a username and password setup with a single *admin* user. The authentication mechanism for this class simply checks if the *username* and *password* of the provided *AuthPackage* matches the configured username and password. If they do, then an *AuthData* object is returned with the provided *username* and given the role of *admin*.

Configuration

No configuration is required for this Authentication Provider but uses two optional items.

username username of the user to be created. Defaults to *admin*.

password password of the user to be created. Defaults to *nimda*.

Example Config

```
auth_provider:
  username: admin
  password: myComplexPassword
```

Example Authentication Request

```
curl -i -X POST -H "Content-Type: application/json" http://localhost:5000/builtins/auth -
↪-data "{\"username\": \"admin\", \"password\": \"nimda\"}"
```

5.2 Implementing Your Own Authentication Provider

Lowball allows you to define your own Authentication Provider and use that Identity Provider in your applications / ecosystem. Let's walk through the process of how to implement one.

Implementing an Authentication Provider starts off by implementing the subclass of *AuthProvider*

```
from lowball.models.provider_models.auth_provider import AuthProvider

class MyCustomAuthProvider(AuthProvider):
    def __init__(self, **kwargs):
        super(MyCustomAuthProvider, self).__init__(**kwargs)
        ...
```

You can pass anything you want into the *__init__* of either class. The top level class is yours to define however you want, and the base class *__init__* doesn't set any attributes or run any methods.

5.2.1 Required Methods

A valid implementation of *AuthProvider must* implement the following methods:

authenticate This is the method that will be used to authenticate by accepting an *AuthPackage*. How this occurs is dependent on the implementation and requirements of the authentication Identity Provider, but this method *must* return an instance of *ClientData*

auth_package_class This is an abstract property that is intended to define the type of authentication package that *authenticate* accepts. This should return the class signature of your implementation, *not* an actual instance of the class. For more information see *AuthPackage*

5.2.2 Optional Methods

The following methods are optional for implementation by the developer. For clarity, any of these implementations would be managing the chosen Identity Provider from lowball.

initialized This is used by lowball to determine if the provider is ready to provide authentication services for the application. It is a *@property* of the class and should be designed as such. By default, this returns *True*.

create_client_package_class This is an abstract property that is intended to define the type of data need to create a user by an admin. This should return the class signature of your implementation, *not* an actual instance of the class. For more information see *CreateClientPackage*

client_registration_package_class This is an abstract property that is intended to define the type of data needed for a user to self-register. This should return the class signature of your implementation, *not* an actual instance of the class. For more information see *ClientRegistrationPackage*

update_client_package_class This is an abstract property that is intended to define the type of data updating a user as an admin. This should return the class signature of your implementation, *not* an actual instance of the class. For more information see *SelfUpdateClientPackage*

self_update_client_package_class This is an abstract property that is intended to define the type of data updating a user as a user accepts. This should return the class signature of your implementation, *not* an actual instance of the class. For more information see *SelfUpdateClientPackage*

create_client Method that can be used to create a client in the Identity Provider. Accepts a *CreateClientPackage*.

client_self_register Method that can be used to self-register a client in the Identity Provider. Accepts a *ClientRegistrationPackage*.

enable_client Used to activate a client in the Identity Provider. It has no requirements on return.

disable_client Used to deactivate a client in the Identity Provider. It has no requirements on return.

delete_client Used to remove a client from the Identity Provider. It has no requirements for its return.

add_roles Used to add a list of roles / group-like attributes to a specific client in the Identity Provider. It takes a *client_id* and a list of the roles to assign. It has no requirements for its return value.

delete_roles Used to remove roles / group-like attributes from a specific client in the Identity Provider. It takes a *client_id* and a list of the roles to remove. It has no requirements for its return value.

update_client Used by an admin to update attributes of an existing client registered in the Identity Provider. It takes a *client_id* and any arguments needed to update the client properly. It should return a *ClientData* Object

client_self_update Identical to *update_client*, but a separate interface was established to allow for a distinction between an admin updating a client and a client updating themself in the system.

list_clients Used to list all the clients who are registered in the system. The return value from this method must be a list of *ClientData* Objects.

get_client Get details on the specified *client_id* provided. It must return an instance of a *ClientData* Object or *ClientData* subclass representing the requested *client_id* and associated roles. This method must be implemented to enable non admin clients to create their own tokens without going through the typical login process, in addition to any features involved with accessing client information.

5.2.3 ClientData Object

For methods that are supposed to return a *ClientData* object, it is best practice, and in many cases required that the result be in the form of a *ClientData* object, either as the base class or a subclass.

The base class takes two arguments to become initialized:

client_id a string representing the identifier for this user. This argument is required.

***roles*:** a list of roles the user has in the system. This must be a list of strings, and is required.

The class also has a method, *to_dict* which will return a dictionary representation of the *ClientData* object. For the base class, this is something like this:

```
{
  "client_id": "client",
  "roles": ["role1", "role2", "role3"]
}
```

Custom Client Objects

If you want to extend the functionality of the *ClientData* class, simply subclass it in your custom object. This custom class will still need *client_id* and *roles* passed in at *__init__* to instantiate, but you may add other arguments if you should choose to do so.

It is recommended that if you do implement new attributes for your custom class that you modify the *to_dict* method in this manner:

```
def to_dict(self):
    base_dict = super(CustomUser, self).to_dict()
    base_dict.update({
        # this is where you add your
        # custom attributes
    })
    return base_dict
```

5.2.4 AuthPackage

AuthPackage is an abstract class that is to be implemented by the Authentication Provider. This class is used to define what data the Authentication Provider expects to be given when a client initially authenticates and requests a token.

The top level objects from the *POST* request to the */builtin/auth* endpoint will be directly mapped to *AuthPackage* class *__init__*.

For example if the *AuthProvider* expects a *username* and *password* then the implementation of the *AuthPackage* would look something like this:

```
from lowball.models.provider_models.auth_provider import AuthPackage

class MyAuthPackage(AuthPackage):

    def __init__(self, username, password, **kwargs):
        super(DefaultAuthPackage, self).__init__(**kwargs)
```

The request body sent to */builtin/auth* would need to come in the following form

```
{
  "username": "the_user",
  "password": "MySuperComplexPassword"
}
```

This would then be mapped to an instance of *MyAuthPackage* and given to the Authentication Provider's *authenticate* method.

5.2.5 CreateClientPackage

CreateClientPackage is an optional abstract class that is to be implemented by the Authentication Provider. This class is used to define what data the Authentication Provider expects to be given when an admin is creating a user in the Authentication Provider.

The top level objects from the *POST* request to the */builtin/auth/create* endpoint will be directly mapped to *CreateClientPackage* class *__init__*.

5.2.6 ClientRegistrationPackage

ClientRegistrationPackage is an optional abstract class that is to be implemented by the Authentication Provider. This class is used to define what data the Authentication Provider expects to be given when a user is self-registering with the Authentication Provider.

The top level objects from the *POST* request to the */builtin/auth/register* endpoint will be directly mapped to *ClientRegistrationPackage* class *__init__*.

5.2.7 UpdateClientPackage

UpdateClientPackage is an optional abstract class that is to be implemented by the Authentication Provider. This class is used to define what data the Authentication Provider expects to be given when an admin is updating attributes of a client in the Authentication Provider.

The top level objects from the *POST* request to the */builtin/auth/clients/<client_id>* endpoint will be directly mapped to *UpdateClientPackage* class *__init__*.

5.2.8 SelfUpdateClientPackage

SelfUpdateClientPackage is an optional abstract class that is to be implemented by the Authentication Provider. This class is used to define what data the Authentication Provider expects to be given when the client is updating attributes of themselves in the Authentication Provider.

The top level objects from the *POST* request to the */builtin/auth/clients* endpoint will be directly mapped to *SelfUpdateClientPackage* class *__init__*.

5.2.9 Using Your Custom Authentication Provider

Once you have created your custom Authentication Provider class, you can use it in your lowball application by passing the class signature to the *auth_provider* argument of the *__init__* of the lowball application:

```
app = Lowball(config=conf, auth_provider=MyCustomAuthProvider)
```

Do not pass an instance of the class at *__init__*. This will fail as lowball is responsible for initializing the class and will map the object the configuration options from the *auth_provider* section of the to the *__init__* of the custom Authentication Provider class.

For example, if your custom auth provider takes three arguments on *__init__* such as *client_id*, *password*, and *hostname*, then the *auth_provider* section of your config should look something like this:

```
auth_provider:
  client_id: user
  password: keepitsecretkeepitsafe
  hostname: host.domain.com
```

These values will be mapped automatically into the *__init___* of your custom class when you instantiate your lowball application.

CHAPTER

SIX

AUTHENTICATION DATABASES

Lowball leverages the concept of an Authentication Database to manage the authentication tokens issued by your application(s). The Authentication Database interface serves as a simple template for interacting with the storage mechanism of your choice e.g. local storage, a traditional database, in memory storage, etc.

The Auth Database does **DOES NOT** store the actual tokens. Rather, it tracks valid tokens and their metadata.

Currently, lowball only supports one Authentication Database at a time for an application.

6.1 Available Authentication Databases

The following are known-existing lowball Authentication Databases. If you have written one and would like it included, please submit a PR or contact us to have it added!

Table 1: Available Authentication Databases

Builtin Basic Authentication Database	Builtin Auth Database to the lowball library for use in your development environment.
lowball-arango-authdb	Utilize an ArangoDB as your Authentication Database backend

6.1.1 Builtin Basic Authentication Database

The builtin Authentication Database makes use of file-system storage for tokens. It is meant to be minimal, have no external dependencies and help a developer get up and running quickly. It could be used in a small production environment that has limited authentication needs.

Configuration

No configuration is required for this Authentication Database but it has one optional item.

token_path This is the path on disk where the token data will be stored. It defaults to */var/lib/lowball/authentication/tokens*

Example Config

```
token_path: "/app/authentication/tokens"
```

6.2 Implementing Your Own Authentication Database

Lowball allows you to define your own Authentication Database to use in your ecosystem. Let's walk through the process of how to implement one.

Implementing an Authentication Database starts off by implementing the subclass of *AuthDatabase*.

```
from lowball.models.provider_models.auth_db import AuthDatabase


class CustomAuthDatabase(AuthDatabase):
    def __init__(self, **kwargs):
        super(CustomAuthDatabase, self).__init__(**kwargs)
        ...
```

You can pass anything you want into the *__init__* of either class. The top level class is yours to define however you want, and the base class *__init__* doesn't set any attributes or run any methods.

6.2.1 Required Methods

A valid implementation of *AuthDatabase *must** implement the following methods:

add_token Used to add a valid token to the Authentication Database. It takes one argument: *token_object*, which **must** be a *Token* object.

lookup_token Used to lookup a single token in the database using a *token_id*. It should return a *Token* object.

revoke_token Used to revoke / delete the token with the supplied *token_id* from the database.

list_tokens Used to return a list of all the tokens in the database. Each item in the returned list should be a *Token* object.

list_tokens_by_client_id Used to look up all tokens associated with a given user. Takes single argument, *client_id*. The output should be a *list* of *Token* objects.

list_tokens_by_role Similar to *list_tokens_by_client_id*; the returned tokens should be all tokens associated with the requested role. Takes single argument *role*. The output should be a *list* of *Token* objects.

cleanup_tokens Is meant to remove all expired tokens from the database.

revoke_all Deletes all tokens from the database

6.2.2 Token Objects

Authentication is carried out in lowball through the use of JSON Web Tokens (JWT). The object representation of these inside of lowball are *Token* objects. These objects are meant to be the return values of several class methods in the codebase that create, update, etc. tokens in the application.

These tokens **do not** actually store the signed JWT, but rather house the data that is encoded in the JWT. These objects take a number of arguments to be initialized (all of which are required):

cid A string representing the client that the token is assigned to.

r A list of roles that this token is authorized for. These roles must all be strings.

cts A datetime string representing when the token was created. The format for the datetime string is *%Y-%m-%d %H:%M:%S*.

ets A datetime string representing when the token will not longer be a valid token for authentication. The format for the datetime string is *%Y-%m-%d %H:%M:%S*.

rcid A string representing the client who requested the particular token.

tid A UUID of the token that is used to identify it in the application. The *[a-zA-Z0-9]{16}*

Token objects have the method, *to_dict* that is used to return a dictionary representation of the *Token* object. When called, it will return something like this:

```
{
  "cid": "user",
  "r": ["role1", "role2", "role3"],
  "cts": "2020-01-01 00:00:00.0000",
  "ets": "2020-02-01 00:00:00.0000",
  "rcid": "issuing_user",
  "tid": "7d760e6d-185a-41f1-b9de-8b87033c5435"
}
```

6.2.3 Using Your Custom Authentication Database

```
app = Lowball(config=conf, auth_database=CustomAuthDatabase)
```

Do not pass an instance of the class at *__init__*. This will fail because the lowball *__init__* will map the object from the *auth_db* section of the to the *__init__* of the custom class.

For example, if your custom Authentication Database takes three arguments on *__init__* such as *username*, *password*, and *database_collection*, then the *auth_db* section of your config should look something like this:

```
auth_db:
  username: user
  password: keepitsecretkeepitsafe
  database_collection: tokens
```

These values will be mapped automatically into the *__init__* of your custom class when you instantiate your lowball application.

CHAPTER

SEVEN

CONFIGS

Lowball configs come in the form of a YAML or JSON file. There are six recognized top-level configuration sections. Each section can contain the configuration object for that section or they can point to a json/yaml file containing the configuration data for that section

meta Metadata about the application such as description, tags, etc.

authentication Configs used for the governance of tokens such as max lifetime of a token.

application Configurations used by the service itself.

auth_provider All data surrounding the chosen Authentication Provider.

auth_db All data surrounding the chosen Authentication Database.

logging All data surrounding the chosen log provider and format.

All configuration values native to lowball have default values and thus have no requirements.

Upon initialization, all configuration values are mapped to the app class variable *lowball_config*, which means that it can be accessed in the following manner:

In the Application Itself

```
self.lowball_config
```

In a View Function

```
from flask import current_app


@app.route("/", methods=["GET"])
def view_func():
    app_config = current_app.lowball_config
```

7.1 Meta Config

The *meta* config section is used to define attributes that describe the application. The primary purpose of this data is exposing it on the */builtin/status* endpoint. However, like all config items it is made available for use anywhere in the application. There is one reserved field used by the operation of the application and four reserved fields used by the status route.

base_route If provided will be prepended to every route. For example if your have *localhost:5000/hello* and then define a base route of */app* then the full url in operation would be *localhost:5000/app/hello*. This functionality can be useful when running the services behind an API Gateway, ingress controller, etc.

name Simple name of the application.

description Simple description of your application.

tags Used to label / tag your application instance.

Meta Example

```
meta:
  name: THE_APP
  base_route: /app
  description: "my super awesome application"
  tags:
    - experimental
    - awesome
```

7.2 Authentication Config

The *authentication* config used for the governance of tokens in the system. There are three available settings:

token_secret The token secret is the string used to encrypt / decrypt the JWTs (tokens). It is highly encouraged to set this to a secure string of your choosing for anything other than local development. If it is not specified in the config, it will default to *CHANGE_ME*.

default_token_life Specifies the lifetime (in seconds) that a generated token is to be valid, if it is not specified by the requester. If a configuration value is not supplied, it will default to 3600 (1 hour).

max_token_life Specifies the maximum lifetime (in seconds) that can be granted to a token. If a configuration value is not specified it will default to 2592000 (~30 days).

```
authentication:
  default_token_life: 3600
  max_token_life: 7200
  token_secret: "supersecrettokensecret"
```

7.3 Application Config

The *application* config is the place where configs that are specific to the application are meant to be stored. This could be anything from URLs for external APIs that the service is meant to make requests to, to usernames for databases that the service pulls data from. Given the nature of this type of data, there is no enforcement on the data that is contained in this section of the config. All that is necessary is that it be in an object format in the config, so that it can be read in as a python *dict* object.

7.4 Authentication Provider Config

Configuration values associated with the chosen Authentication Provider. These values are defined by the implementation of the given provider. See *Available Authentication Providers* for further documentation.

7.5 Auth Database Config

Configuration values associated with the chosen Authentication Database. These values defined by the implementation of the given database. See *Available Authentication Databases* for further documentation.

7.6 Logging Config

Configuration values associated with the chosen Logging Provider. See *Logging* for further documentation.

7.7 Reading in Configs

The configs that lowball uses can be read in using two methods: 1. Directly from a python *dict* using the *config_from_object* builtin method 2. From a JSON or YAML file using the *config_from_file* builtin method

From an object

```
from lowball import config_from_object

config = {
   "meta": {
      "name": "APP",
      "base_route": "/app",
      "description": "example to show config reading methods"
   },
   "authentication": {
      "max_token_life": 7200,
      "default_token_life": 3600,
      "token_secret": "supersecrettokensecret"
   },
   "application": {
      "username": "user_of_import"
   },
   "auth_provider": {
      ...
   },
   "auth_db": {
      ...
   },
   "logging": {
      ...
   }
}

config_object = config_from_object(config)
```

From a JSON File

config.json could look something like this:

```
{
   "meta": {
      "name": "APP",
      "base_route": "/app",
      "description": "example to show config reading methods"
   },
   "authentication": {
      "max_token_life": 7200,
      "default_token_life": 3600,
      "token_secret": "supersecrettokensecret"
   },
   "application": {
      "username": "user_of_import"
   },
   "auth_provider": {

   },
   "auth_db": {

   },
   "logging": {

   }
}
```

We would read it in like this:

```
from lowball import config_from_file

config_object = config_from_file("./config.json")
```

From a YAML File

config.yaml could look something like this:

```
meta:
  name: APP
  base_route: /app
  description: "description of application goes here"
authentication:
  default_token_life: 3600
  max_token_life: 7200
  token_secret: "supersecrettokensecret"
application:
  username: user_of_import
auth_provider:
  ...
auth_db:
  ...
logging:
  ...
```

or this if using sub files:

```
meta: /path/to/meta.yaml
authentication:
  default_token_life: 3600
  max_token_life: 7200
  token_secret: "supersecrettokensecret"
application: /path/to/app.yaml
auth_provider:
  ...
auth_db:
  ...
logging:
  ...
```

We would read it in like this:

```
from lowball import config_from_file

config_object = config_from_file("./config.yaml")
```

CHAPTER

EIGHT

LOGGING

Lowball allows you to use any python supported logging handler and formatter. The only lowball specific consideration is if your chosen logger requires any config.

Like the Authentication Providers and Databases, any config variables needed by your logger will be mapped by the config directly to the corresponding *__init__* arguments. A simple example of this is below.

Given the following handler

```
from logging.handlers import  SocketHandler

class CustomLogHandler(SocketHandler):
    def __init__(self, host, port, **kwargs):
        super(CustomLogHandler, self).__init__(host, port)
        ...
```

The logging section of our config might look like:

```
logging:
  host: 127.0.0.1
  port: 4500
```

8.1 Default Logging Handler

The default logging handler in lowball is a *RotatingFileHandler* that outputs the logs in JSON format.

The all optional available configuration values are:

filename File that the logger will write to. By default, this is *./lowball.log*.

formatter A dictionary that is used to instantiate the formatter for the log handler.

log_level The minimum level of log that is to be recorded by the logger. These follow the python standards.

max_bytes A value that defines the maximum number of bytes that can be written to the log file before it is rolled over or overwritten. This value defaults to 2^20 bytes, or 1mb, and must be a number greater than 0.

backup_count Defines the number of rolled-over log files that are kept by the logging handler. This value defaults to 5 and must be a number greater than 0.

Example Logging Config

```
logging:
  filename: /var/log/app/app.log
  formatter:
     date_format: "%Y-%m-%d %H:%M:%S.%fUTC"
  log_level: DEBUG
  max_bytes: 1048576
  backup_count: 5
```

Example Non-Verbose Log

```
{
  "name": "myApp",
  "msg": {
    "result": "401 UNAUTHORIZED",
    "error_information": null
  },
  "args": [],
  "additional": {
     "user_agent": "curl/7.68.0",
     "src_ip": "127.0.0.1",
     "http_method": "POST",
     "url": "/launch",
     "status_code": 401,
     "user_data": {
        "requesting_user": "jeff",
        "client_token_id": "0347303c-ffc9-46ea-bded-22e3258dd3b2",
     }
  },
  "timestamp": "2021-02-09 15:36:05.062443UTC",
  "level": "ERROR",
  "requesting_user": "jeff",
  "client_token_id": "0347303c-ffc9-46ea-bded-22e3258dd3b2",
  "request_id": "70047f54-d296-4a57-b7cd-b087fa72f269"
}
```

Example Verbose Log

```
{
  "name": "myApp",
  "msg": {
    "result": "401 UNAUTHORIZED",
    "error_information": null
  },
  "args": [],
  "pathname": "/path/to/calling/file/request_finished_log.py",
  "filename": "request_finished_log.py",
  "module": "request_finished_log",
  "exc_info": null,
  "stack_info": null,
  "thread": 140111293720320,
  "process": 54683,
  "additional": {
     "user_agent": "curl/7.68.0",
```

(continues on next page)

(continued from previous page)


```
    "src_ip": "127.0.0.1",
    "http_method": "POST",
    "url": "/launch",
    "status_code": 401,
    "user_data": {
       "requesting_client": "jeff",
       "client_token_id": "0347303c-ffc9-46ea-bded-22e3258dd3b2",
    }
  },
  "timestamp": "2021-02-09 15:36:05.062443UTC",
  "func_name": "request_finished_log",
  "line_number": 45,
  "level": "ERROR",
  "process_name": "MainProcess",
  "thread_name": "Thread-3",
  "requesting_client": "jeff",
  "client_token_id": "0347303c-ffc9-46ea-bded-22e3258dd3b2",
  "request_id": "70047f54-d296-4a57-b7cd-b087fa72f269"
}
```

HTTP ROUTING TABLE

/builtins