
DISCLAIMER

This project is not being maintained and I don't recommend using it in its current form. As an alternative, I recommend using the `jwt` gem directly.

knock



Seamless JWT authentication for Rails API

Description

Knock is an authentication solution for Rails API-only application based on JSON Web Tokens.

Getting Started

Installation

Add this line to your application's Gemfile:

```
1 gem 'knock'
```

Then execute:

```
1 $ bundle install
```

Requirements

Knock makes one assumption about your user model:

It must have an `authenticate` method, similar to the one added by `has_secure_password`.

```
1 class User < ActiveRecord::Base
2   has_secure_password
3 end
```

Using `has_secure_password` is recommended, but you don't have to as long as your user model implements an `authenticate` instance method with the same behavior.

Usage

Include the `Knock::Authenticable` module in your `ApplicationController`

```
1 class ApplicationController < ActionController::API
2   include Knock::Authenticable
3 end
```

You can now protect your resources by calling `authenticate_user` as a `before_action` inside your controllers:

```
1 class SecuredController < ApplicationController
2   before_action :authenticate_user
3
4   def index
5     # etc...
6   end
7
8   # etc...
9 end
```

You can access the current user in your controller with `current_user`.

If no valid token is passed with the request, Knock will respond with:

```
1 head :unauthorized
```

You can modify this behaviour by overriding `unauthorized_entity` in your controller.

You also have access directly to `current_user` which will try to authenticate or return `nil`:

```
1 def index
2   if current_user
3     # do something
4   else
5     # do something else
6   end
7 end
```

Note: the `authenticate_user` method uses the `current_user` method. Overwriting `current_user` may cause unexpected behaviour.

You can do the exact same thing for any entity. E.g. for `Admin`, use `authenticate_admin` and `current_admin` instead.

If you're using a namespaced model, Knock won't be able to infer it automatically from the method name. Instead you can use `authenticate_for` directly like this:

```
1 class ApplicationController < ActionController::Base
2   include Knock::Authenticable
```

```
3
4   private
5
6   def authenticate_v1_user
7     authenticate_for V1::User
8   end
9 end
```

```
1 class SecuredController < ApplicationController
2   before_action :authenticate_v1_user
3 end
```

Then you get the current user by calling `current_v1_user` instead of `current_user`.

Configuration

In the entity model The entity model (e.g. `User`) can implement specific methods to provide customization over different parts of the authentication process.

- **Find the entity when creating the token (when signing in)**

By default, Knock tries to find the entity by email. If you want to modify this behaviour, implement within your entity model a class method `from_token_request` that takes the request in argument.

E.g.

```
1 class User < ActiveRecord::Base
2   def self.from_token_request request
3     # Returns a valid user, `nil` or raise `Knock.
4     #   not_found_exception_class_name`
5     # e.g.
6     #   email = request.params["auth"] && request.params["auth"]["email"]
7     #   self.find_by email: email
8   end
9 end
```

- **Find the authenticated entity from the token payload (when authenticating a request)**

By default, Knock assumes the payload as a subject (`sub`) claim containing the entity's id and calls `find` on the model. If you want to modify this behaviour, implement within your entity model a class method `from_token_payload` that takes the payload in argument.

E.g.

```
1 class User < ActiveRecord::Base
```

```
2  def self.from_token_payload payload
3    # Returns a valid user, `nil` or raise
4    # e.g.
5    #   self.find payload["sub"]
6  end
7  end
```

- **Modify the token payload**

By default the token payload contains the entity's id inside the subject (**sub**) claim. If you want to modify this behaviour, implement within your entity model an instance method `to_token_payload` that returns a hash representing the payload.

E.g.

```
1  class User < ActiveRecord::Base
2    def to_token_payload
3      # Returns the payload as a hash
4    end
5  end
```

- **Token Lifetime**

By default the generated tokens will be valid, after generated, for 1 day. You can change it in the Knock configuration file (`config/knock.rb`), setting the desired lifetime:

E.g.

```
1  Knock.token_lifetime = 3.hours
```

If you are generating tokens for more than one entity, you can pass each lifetime in a hash, using the entities class names as keys, like:

E.g.

```
1  # How long before a token is expired. If nil is provided,
2  # token will last forever.
3  Knock.token_lifetime = {
4    user: 1.day
5    admin: 30.minutes
6  }
```

In the initializer Read `lib/knock.rb` to learn about all the possible configuration options and their default values.

You can create an initializer like in the example below:

Inside `config/initializers/knock.rb`

```
1 Knock.setup do |config|
2   config.token_lifetime = 1.hour
3
4   # For Auth0
5   config.token_audience = -> { Rails.application.secrets.
    auth0_client_id }
6   config.token_secret_signature_key = -> { JWT.base64url_decode Rails.
    application.secrets.auth0_client_secret }
7 end
```

Authenticating from a web or mobile application

Example request to get a token from your API:

```
1 POST /user_token
2 {"auth": {"email": "foo@bar.com", "password": "secret"}}
```

Example response from the API:

```
1 201 Created
2 {"jwt": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9"}
```

To make an authenticated request to your API, you need to pass the token via the request header:

```
1 Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
2 GET /my_resources
```

Knock responds with a 404 **Not Found** when the user cannot be found or the password is invalid. This is a security best practice to avoid giving away information about the existence or not of a particular user.

NB: HTTPS should always be enabled when sending a password or token in your request.

Authenticated tests

To authenticate within your tests:

1. Create a valid token
2. Pass it in your request

e.g.

```
1 class SecuredResourcesControllerTest < ActionDispatch::IntegrationTest
2   def authenticated_header
```

```
3     token = Knock::AuthToken.new(payload: { sub: users(:one).id }).
      token
4
5     {
6       'Authorization': "Bearer #{token}"
7     }
8   end
9
10  it 'responds successfully' do
11    get secured_resources_url, headers: authenticated_header
12
13    assert_response :success
14  end
15 end
```

Without ActiveRecord If no ActiveRecord is used, then you will need to specify what Exception will be used when the user is not found with the given credentials.

```
1 Knock.setup do |config|
2
3   # Exception Class
4   # -----
5   #
6   # Configure the Exception to be used (raised and rescued) for User
7   #   Not Found.
8   # note: change this if ActiveRecord is not being used.
9   #
10  # Default:
11  config.not_found_exception_class_name = 'MyCustomException'
12 end
```

Algorithms

The JWT spec supports different kind of cryptographic signing algorithms. You can set `token_signature_algorithm` to use the one you want in the initializer or do nothing and use the default one (HS256).

You can specify any of the algorithms supported by the jwt gem.

If the algorithm you use requires a public key, you also need to set `token_public_key` in the initializer.

CORS

To enable cross-origin resource sharing, check out the rack-cors gem.

Related links

- 10 things you should know about tokens

Contributing

1. Fork it (<https://github.com/nsarno/knock/fork>)
2. Create your feature branch (`git checkout -b my-new-feature`)
3. Commit your changes (`git commit -am 'Add some feature'`)
4. Push to the branch (`git push origin my-new-feature`)
5. Create a new Pull Request

License

MIT