
falcon-epdb Documentation

Josh Wilson

Feb 07, 2019

Contents:

1	Installation	3
2	Usage	5
2.1	Configuring the middleware	5
2.2	Constructing the X-EPDB header	5
2.3	Connecting the client	6
2.4	Backends	6
2.4.1	Base64	6
2.4.2	Fernet	6
2.4.3	JWT	7
3	Troubleshooting	9
4	API	11
4.1	Middleware	11
4.1.1	EPDBServe	11
4.2	Backends	12
4.2.1	Base64Backend	12
4.2.2	FernetBackend	12
4.2.3	JWTBackend	13
4.2.4	EPDBBackend	13
4.3	Exceptions	14
4.3.1	EPDBException	14
5	Development	15
5.1	Style checks and testing	15
5.2	Adding dependencies	16
5.3	Publishing a new release	16
6	Changelog	17
6.1	v1.0.0	17
6.1.1	Initial release	17

A **Falcon middleware** that wraps the excellent **epdb** tool and allows one to connect to a running Falcon app and use interactive debugging to step through the code.

Better documentation can be found at [readthedocs](#).

CHAPTER 1

Installation

If you are only planning on debugging in a development environment where access to your service is restricted to you or trusted partners, you may find the *Base64* sufficient to your purposes. You can just install the library as you would any Python library.

Listing 1: **requirements.txt**

```
falcon-epdb
```

Listing 2: **pip**

```
pip install falcon-epdb
```

Listing 3: **poetry**

```
poetry add falcon-epdb
```

However, if you need a little more security, you can use one of the other authenticated backends (*Fernet*, *JWT*). Choose the one that best fits your use case and install it as a Python *extra*.

Listing 4: **requirements.txt**

```
falcon-epdb[fernet]
```

Listing 5: **pip**

```
pip install falcon-epdb[fernet, jwt]
```

Listing 6: **poetry**

```
poetry add falcon-epdb[jwt]
```


This library adds a middleware to your Falcon API stack, and as such will run for all requests, save those excluded by `exempt_methods` provided to the `EPDBServer` constructor. If it detects a well-formed (and possibly authenticated) `X-EPDB` header on the request it will start the `epdb` server on the configured port and block until it establishes a connection from an `epdb` client, at which point processing continues but under the control of the remote debugging session.

Subsequent requests with an acceptable header will reuse the client connection and automatically drop into the remote debugging session again.

2.1 Configuring the middleware

The `EPDBServe` middleware accepts a handful of parameters. The most important are the `backend` and `serve_options` parameters. The `backend` determines how a request is examined for the “secret knock” to start the remote debugging server. The included implementations assume a well-formed `X-EPDB` header, but nothing precludes you from sub-classing `EPDBBackend` and implementing your own.

The `serve_options` are options that are passed through to the `epdb.serve()` call. See [Backends](#) for details on how to add this middleware to your API.

2.2 Constructing the X-EPDB header

The content of the header is as follows:

```
{
  "epdb": {}
}
```

Depending on the backend in use, one should encode this content into the appropriate header-safe value. Then append this value to the name of the backend.

```
X-EPDB: Base64 eyJlcGRiIjog319
```

2.3 Connecting the client

Example code for connecting to the waiting port:

```
import epdb

epdb.connect(host=<host>, port=9000)
```

2.4 Backends

2.4.1 Base64

Listing 1: Server side configuration

```
epdb_middleware = EPDBServe(
    backend=Base64Backend(),
    serve_options={'port': 9000})
api = falcon.API(middleware=[epdb_middleware])
```

Listing 2: Crafting an appropriate header

```
import base64
import json

header_content = base64.b64encode(json.dumps({'epdb': {}}).encode()).decode()
header_value = 'Base64 {}'.format(header_content)
```

2.4.2 Fernet

Listing 3: Server side configuration

```
fernet_key = Fernet.generate_key() # The shared key
epdb_middleware = EPDBServe(
    backend=FernetBackend(key=fernet_key),
    serve_options={'port': 9000})
api = falcon.API(middleware=[epdb_middleware])
```

Listing 4: Crafting an appropriate header

```
import json
from cryptography.fernet import Fernet

f = Fernet(<fernet_key>) # Key configured on the server
header_content = f.encrypt(json.dumps({'epdb': {}}).encode()).decode()
header_value = 'Fernet {}'.format(header_content)
```

2.4.3 JWT

Listing 5: Server side configuration

```
jwt_key = uuid.uuid4().hex # The shared key
epdb_middleware = EPDBServe(
    backend=JWTBackend(key=jwt_key),
    serve_options={'port': 9000})
api = falcon.API(middleware=[epdb_middleware])
```

Listing 6: Crafting an appropriate header

```
import jwt

header_content = jwt.encode({'epdb': {}}, <jwt_key>, algorithm='HS256').decode()
header_value = 'JWT {}'.format(header_content)
```


CHAPTER 3

Troubleshooting

You must be sure to allow access to the configured port on your host. Be sure to check your security groups and firewall rules.

Configure your web app to only run one worker process. If you have multiple workers, only the first one will be able to serve on the configured port. If this is not possible you will have to take steps to ensure that all requests that wish to use the remote debugging port are routed to the same worker. This will depend heavily on your HTTP stack and is beyond the scope of this documentation.

Be sure to up your request timeout limit to something on the order of minutes so that the HTTP server doesn't close your request connection or kill your worker process while you're debugging.

You may need to provide the `HTTP-` prefix on your `X-EPDB` header for it to be handled correctly. So instead of sending `X-EPDB`, you would send `HTTP-X-EPDB`.

4.1 Middleware

4.1.1 EPDBServe

class `falcon_epdb.EPDBServe` (*backend*, *exempt_methods*=('OPTIONS',), *serve_options*=None)

A middleware to enable remote debugging via an `epdb` server.

Parameters

- **backend** (`EPDBBackend`) – An instance of the class that will validate and decode the X-EPDB header
- **exempt_methods** (*iterable of strings*) – HTTP methods which will be ignored by this middleware
- **serve_options** (*dictionary*) – Parameters passed-through to `epdb.serve()`

A client may include a special X-EPDB header containing an appropriately formed payload. If they do, the header will be passed to the configured backend for processing. If the payload passes authentication and meets the content requirements, the app will begin listening for `epdb` client connections.

A well-formed header has content simply of the form:

```
{
  "epdb": {}
}
```

The encoding and encryption of this payload is determined by the `EPDBBackend` provided to the middleware.

process_request (*req*, *resp*)

Check for a well-formed X-EPDB header and if present activate the `epdb` server.

Parameters

- **req** – The Falcon request object

- **resp** – The Falcon response object (unused)

This will block, waiting for an `epdb` client connection, the first time a valid header is received. Once the client is connected, subsequent passes will simply activate the connected client and drop it into the `epdb` shell.

The header processing is delegated to the configured `EPDBBackend`.

4.2 Backends

4.2.1 Base64Backend

class `falcon_epdb.Base64Backend`

A simple unauthenticated backend for local development.

decode_header_value (*epdb_header*)

Pull the encrypted data out of the header, if present.

Parameters `epdb_header` (*string*) – The content of the X-EPDB header.

Returns The decoded header payload

Return type dictionary

Raises `EPDBException`

It expects `epdb_header` to have the Base64 prefix.

4.2.2 FernetBackend

class `falcon_epdb.FernetBackend` (*key*)

A Python cryptography-based backend that supports a pre-shared key (ie. password) protocol.

Parameters `key` (*bytes*) – The fernet key used to encrypt the header content

Note: To use this backend, one must install the `cryptography` package. The easiest way to do this is to specify the `[fernet]` extra when adding the `falcon-epdb` dependency to your project.

Listing 1: `requirements.txt`

```
falcon-epdb[fernet]
```

decode_header_value (*epdb_header*)

Pull the encrypted data out of the header, if present.

Parameters `epdb_header` (*string*) – The content of the X-EPDB header.

Returns The decoded and decrypted header payload

Return type dictionary

Raises `EPDBException`

It expects `epdb_header` to have the Fernet prefix.

4.2.3 JWTBackend

class `falcon_epdb.JWTBackend` (*key*)

A JWT-based backend that supports a pre-shared key (ie. password) protocol.

Parameters `key` (*bytes*) – The JWT key used to encrypt the header content

Note: To use this backend, one must install the `PyJWT` package. The easiest way to do this is to specify the `[jwt]` extra when adding the `falcon-epdb` dependency to your project.

Listing 2: `requirements.txt`

```
falcon-epdb[jwt]
```

decode_header_value (*epdb_header*)

Pull the encrypted data out of the header, if present.

Parameters `epdb_header` (*string*) – The content of the X-EPDB header.

Returns The decoded and decrypted header payload

Return type dictionary

Raises `EPDBException`

It expects `epdb_header` to have the JWT prefix.

4.2.4 EPDBBackend

class `falcon_epdb.EPDBBackend`

The abstract base class defining the header-processing backend interface.

An inheriting subclass must define `decode_header_value()`, but may define other methods if necessary. This class is structured to provide a balance of convenience and flexibility.

decode_header_value (*epdb_header*)

Process the X-EPDB header content.

Parameters `epdb_header` (*string*) – The content of the X-EPDB header

Returns The decoded and decrypted header payload

Return type dictionary

This does not need to do any content validation, as that is handled in `validate_header_content()`.

get_header_data (*req*)

Process a request and return the contents of a conforming payload.

Parameters `req` (*Request*) – The Falcon request object

Returns The payload content or `None`

Return type dictionary or `None`

This implementation assumes that the payload is present on the X-EPDB header, but subclasses may override this method if their use-case demands it.

If the request does not appear to be attempting begin a debugging session, this will return `None`.

static validate_header_content (*header_content*)

Ensure that the decoded X-EPDB header content is well-formed.

Parameters **header_content** (*dictionary*) – The decoded X-EPDB header content

Returns The value of the `epdb` key

Return type dictionary

Raises EPDBException

`header_content` must be of the form:

```
{
    "epdb": {}
}
```

4.3 Exceptions

4.3.1 EPDBException

exception `falcon_epdb.EPDBException`

Raised when an error occurs during the processing of an X-EPDB header.

Issues and pull requests are welcome at [github](#). Please be sure to add or update the documentation appropriately along with your code changes.

5.1 Style checks and testing

Any pull requests will be validated with [Travis CI](#), but you may run the tests locally with [tox](#) and/or [poetry](#). We use [tox](#) to wrap [poetry](#) commands in our Travis CI configuration.

Run the entire suite of tests:

Listing 1: Using [tox](#)

```
tox
```

Or just run one off tests:

Listing 2: Using [poetry](#)

```
# Install the project dependencies, including dev-dependencies into a
# poetry-managed virtual environment.
poetry install -E jwt -E fernet

# Run the individual test commands as needed in the virtual environment
poetry run pytest -v tests
poetry run flake8 falcon_epdb tests
poetry run pylint falcon_epdb
poetry run pydocstyle falcon_epdb tests

# Build the docs and find them in docs/_build
poetry run sphinx-build -b html docs docs/_build
```

5.2 Adding dependencies

Use the `poetry add` command to add dependencies to the `pyproject.toml` file.

Listing 3: Using `poetry add`

```
poetry add cryptography
poetry add --dev coveralls
```

5.3 Publishing a new release

The project is configured to publish a release anytime a tag is pushed to the github repository and the build succeeds. The tagging convention is `v<Major>.<minor>.<patch>`, and it should follow [semver](#) conventions. One can bump the version using the `poetry version` command.

When creating a release, ensure the following:

- The documentation is up to date with the new changes.
- The changes have been noted in the `CHANGELOG.rst`.
- The build “badges” are all passing. The readthedocs one seems somewhat finicky these days.
- The version has been incremented accordingly.

CHAPTER 6

Changelog

6.1 v1.0.0

6.1.1 Initial release

- Add support for Fernet backend
- Add support for JWT backend

B

Base64Backend (class in falcon_epdb), [12](#)

D

decode_header_value() (falcon_epdb.Base64Backend method), [12](#)

decode_header_value() (falcon_epdb.EPDBBackend method), [13](#)

decode_header_value() (falcon_epdb.FernetBackend method), [12](#)

decode_header_value() (falcon_epdb.JWTBackend method), [13](#)

E

EPDBBackend (class in falcon_epdb), [13](#)

EPDBException, [14](#)

EPDBServe (class in falcon_epdb), [11](#)

F

FernetBackend (class in falcon_epdb), [12](#)

G

get_header_data() (falcon_epdb.EPDBBackend method), [13](#)

J

JWTBackend (class in falcon_epdb), [13](#)

P

process_request() (falcon_epdb.EPDBServe method), [11](#)

V

validate_header_content() (falcon_epdb.EPDBBackend static method), [13](#)