Syn Documentation

Release 0.0.21

Victor De Gouveia

Jul 31, 2018

Contents:

Syn is an easy to use password management application developed using the Python programming language and an underlying Soledad engine for data synchronization and security.

Syn started as a project proposal for the LEAP non-profit organization in the Google Summer of Code 2018 program. The project is currently in development, but it is mature enough to be used in its current state. The goal of the project is to create a fully-functional and secure password management application which can be used to synchronize data across devices.

CHAPTER 1

Table of Contents

1.1 Introduction

Syn is an easy to use password management application developed using the Python programming language and an underlying Soledad engine for data synchronization and security. Furthermore, Soledad is built using the Twisted event-driven framework. Both of these components allow Syn to work with data that is encrypted as soon as it enters the system, as well as synchronizing this data across devices and allowing for asynchronous processing.

Syn allows to create locally encrypted databases in the user's filesystem. Within these databases, Syn can store, retrieve, and delete information in a secure way. These databases are meant to be used as secure password repositories, where the user can quickly find an entry and copy the password to the system clipboard. Unlike many other password managers out there today, Syn places great emphasis on being completely functional without needing an active internet connection.

The functionality of Syn can be accessed via a command line interface, which for now, is the main entry point of the application. However, since Syn exposes an API to make use of its functionalities and features, a GUI is planned for some point in the future. a locally encrypted database that holds the stored information via an HTTP server daemon.

1.2 Installation

Syn is distributed as a Python package. As such, `pip` can be used to install it.

To install Syn, execute the following command in a terminal console:

```
$ pip install syn
```

To check if Syn was installed correctly, execute the following command:

```
$ syn --help
```

The following output should appear in your terminal console:

```
Usage: syn [OPTIONS] COMMAND [ARGS]...

Syn is a password manager which offers an easy to use interface and
cryptographically secure storage of passwords and notes.

Options:
--help  Show this message and exit.

Commands:
    add           Store a new entry in the database
    close         Closes a database
    delete_db     Deletes a database
    delete_entry  Delete an entry from the database
    get           Shows the details of an entry
    list          Lists the names of stored entries
    new           Creates a new database
    open          Opens the database
    passphrase    Generates a passphrase
```

1.2.1 Source Code

Source code for Syn can be found here.

1.3 Architecture

Syn is made up of two main components, which act as its building blocks:

The two main components of Syn are Soledad, which is a solution for synchronizing encrypted data, and Twisted, which is an event-driven networking framework for Python.

Syn uses Soledad as its data management engine, meaning all operations that have to do with the creation, deletion, updating or selection of data is done via the Soledad component. Some benefits of using Soledad for this task are:

- Data encryption is done locally and handled completely by this component
- Contains mechanisms to synchronize data over the network, allowing for devices to share state

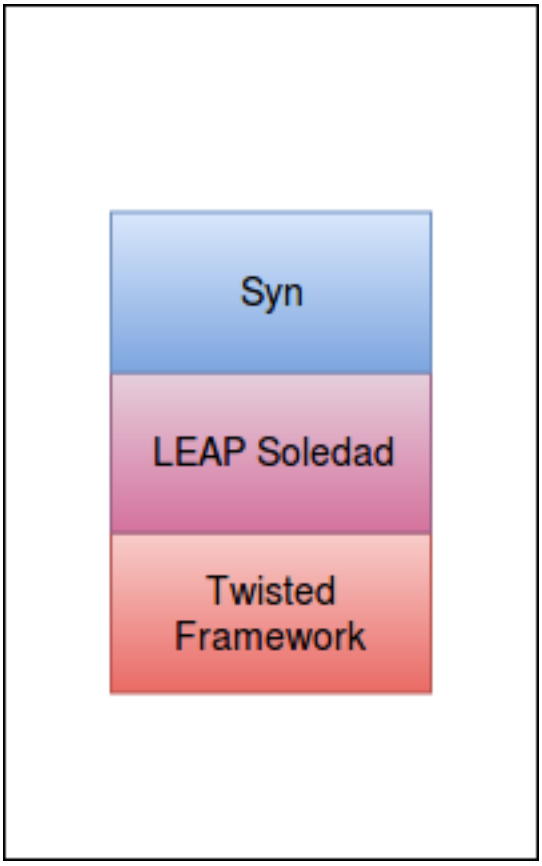
What Syn accomplishes with Soledad is to create an extra layer of functionality that allows the management of multiple different local database instances. A use case for this would be for different users on the same machine to be able to manage their own separately encrypted password stores, or even for the same user to split their passwords into different organizational levels (passwords used at home, for work, entertainment, etc).

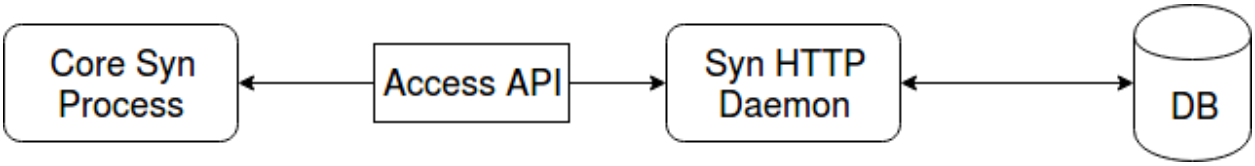
The typical flow behind the execution of Syn looks something like this:

Basically, executing the core Syn process will spawn a daemon process in the background. The daemon process is, essentially, a local HTTP server. The core process communicates with this daemon via an HTTP API, and the daemon executes the desired action on the locally encrypted database and returns a response. The decision to include an HTTP daemon will be explained with greater detail in the *HTTP Daemon* section.

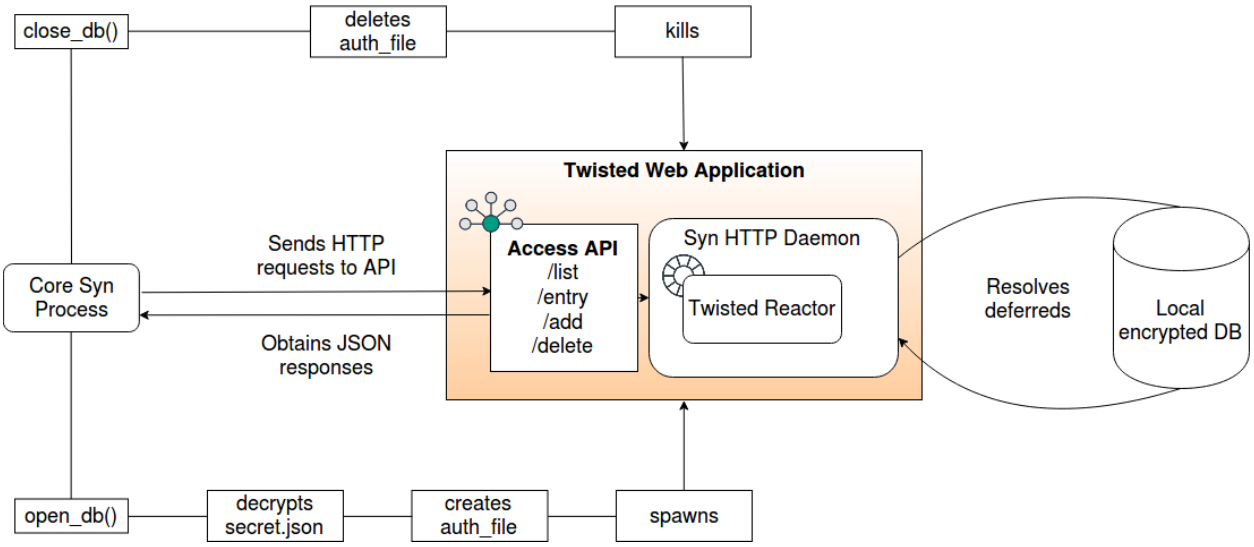
1.3.1 HTTP Daemon

The decision to include this HTTP daemon was to decouple the code as well as to make it more functional and well-aligned with the Twisted framework. By doing so, Syn's architecture can be split up into two components and their concerns separated. Now, the core process can cleanly deal with filesystem related things, while the HTTP daemon


Syn
LEAP Soledad
Twisted
Framework


Core Syn
Process
Access API
Syn HTTP
Daemon
DB

can deal with all things Soledad and TWisted, and they both communicate through a minimal API. The following diagram illustrates this relationship:



In the diagram, we have the following entities:

- **Core Syn Process:** this process takes care of all the filesystem related responsibilities, such as creating and deleting local databases which are, essentially, encrypted files. It is also in charge of "opening" or decrypting the databases and keeping track of the state of the database. This process is presented as a command line interface.
- **Syn HTTP Daemon:** technically, this is a fully fledged Twisted Web Application. As specified, this component takes care of all the Twisted and Soledad related functionality with respect to accessing the locally encrypted database instances.
- **Local encrypted database:** this is the database file stored in the local filesystem which has been encrypted by the Soledad engine.

Taking a look at the diagram, a normal execution would be constructed as follows:

Spawning the core process and HTTP daemon

1. The Core Syn Process is instantiated via the command line
2. Syn is instructed to open a database and authentication details such as the database's UUID and passphrase are provided
3. The database is decrypted and the `auth_file` is created, which keeps the state of the database locally
4. The Syn HTTP Daemon is spawned, exposing a local HTTP API.

Communicating with the HTTP daemon

1. The Core Syn Process, which is basically a CLI application, can send HTTP requests to the daemon via an exposed API. Each request maps to a certain functionality that wants to be executed, such as creation or deletion of passwords
2. The API receives the request, routes the code flow to the corresponding response method and obtains a `Deferred` object
3. Once the `Deferred` object is obtained, the `reactor` resolves the `Deferred` by accessing the local database and encodes the results as JSON strings

4. The result obtained from resolving the `Deferred` is sent back via the API as an HTTP response. The client, or Core Process can now do as it wishes with the result.

To learn more about `Deferred` objects and asynchronous work flows, check out this blog post.

Stopping Syn

1. The Core Syn Process is instructed to close the currently open database
2. The database is closed by deleting the *auth_file*, which means there is no longer a record of the current state of an open database. If there is no state record, Syn assumes there does not exist an open database
3. A kill signal is sent to the Syn HTTP Daemon, prompting it to clean up and terminate. Once this process has terminated, there is no way to interact via the local databases.

1.4 Using Syn

Syn is meant to be used as a straightforward CLI application meant for quick and easy storage of important information that is frequently used, such as passwords. In this section, a brief walkthrough is detailed on how to use the different Syn functions. In order to follow this walkthrough, it is necessary to have installed Syn through *pip*. This setup process is covered in the *Installation* section.

1.4.1 Creating a database

To create a database, we execute the following command:

```
$ syn new
```

This will result in a prompt requesting a UUID and passphrase for the database that is to be created. **Note:** a database cannot have the same UUID as another one.

```
$ syn new

Creating a new database...
UUID: my_db
Passphrase:
Repeat for confirmation:
Successfully created a database!
```

Upon inserting a valid UUID and passphrase, the terminal should tell us that the database was created successfully. Congratulations, you have created your first Syn database!

1.4.2 Opening the database

The database that was just created is encrypted and stored locally. If we want to perform any operations in this database, such as storing, reading, or deleting entries, we first have to "open" the database. Opening a database is equivalent to authenticating yourself towards the Syn system, in order to prove that you know both the UUID and passphrase of said database. Upon completing this step, the database operations will be able to be performed.

To open the database, simply execute the following command:

```
$ syn open
```

A prompt will request the UUID and passphrase of the database you wish to open. Upon successfully entering the requested information, the database will open and the following message will be prompted:

```
$ syn open

UUID: my_db
Passphrase:
Database has been opened
```

1.4.3 Inserting entries into the database

To store an entry into the database, the following command must be executed:

```
$ syn add
```

A prompt will appear requesting for information such as the name of the entry and if you would like to randomly generate a password or provide one yourself.

```
$ syn add

Name the entry: Email
Would you like to randomly generate a passphrase? [y/N]: y
Enter the desired length of the passphrase [8]: 12
Enter a generation seed []:
```

The name of the entry is very important, as it must be unique among all entries within the same database. The entry name is used to identify entries and locate them within the database instance, so make sure you are using names that are not already in use.

1.4.4 Getting a list of entries from the database

Syn allows to get a list of all existing entries within the currently open database with a single command. In order to obtain the list of existing entries, simply execute the following command:

```
$ syn list

Found 4 entries:
Email
Locker
Phone
Server
```

1.4.5 Reading an entry from the database

Reading an entry from the database will allow you to obtain its details such as the content stored within the entry, the date it was created, and the last time it was modified. To do so, the following command must be executed:

```
$ syn get
```

With this operation, you only need to provide the name of the entry you wish to obtain. After that, the details of the entry should be prompted on the command line interface:

```
$ syn get

Name of the entry you wish to obtain: Email
```

(continues on next page)

(continued from previous page)

```
Found the following entry:
──────────────────────────
Name: Email
Time of creation: 2018-07-31 14:00:11 UTC
Time of last modification: 2018-07-31 14:00:11 UTC
Contents of the entry have been copied to the system clipboard!
──────────────────────────
```

After getting the details of an entry from the database, the password that was stored with it will have been copied to your system clipboard. This helps to make password lookup a convenient and quick task when using Syn.

1.4.6 Deleting entries from the database

Deleting an entry will remove the data from the local database. The process to deleting an entry is very similar to the process for reading an entry. To delete an entry, the following command must be executed:

```
$ syn delete
```

Syn will then prompt for the name of the entry, and upon entering the name of an existing entry, Syn will proceed to remove the data from the database:

```
$ syn delete

Name of the entry you wish to delete: Email
Entry was deleted successfully.
```

1.4.7 Closing a database

Closing a database will kill the *HTTP daemon* that grants access to the database, effectively closing off communication between any program and the encrypted local database, including the Syn Core Process. A database should be closed when it is no longer in use.

To close the database, execute the following command:

```
$ syn close
```

Any open database will be closed and the following message will be prompted:

```
$ syn close
Database has been closed
```

1.4.8 Deleting a database

Entire databases can be deleted using Syn. To do so, execute the following command:

```
$ syn delete_db
```

Syn will prompt for authentication before deletion, to make sure you really want to delete the database. Once a database has been deleted, the information cannot be recovered.

```
$ syn delete_db

Deleting a database... Press CTRL+C to abort.
UUID: my_db
Passphrase:
Repeat for confirmation:
Database has been deleted.
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search