Jana Documentation

Release 0.1.0

Christopher Petrilli

January 25, 2015

Contents

Contents:

CHAPTER 1

Jana

RESTful manager of secrets

- Free software: BSD license
- Documentation: https://jana.readthedocs.org.

1.1 Overview

Jana enables developers, sysadmins, and users to store secrets in a trusted system. If those secrets represent cryptographic keys, then Jana allows them to be used to perform cryptographic operations within Jana. For secrets, Jana supports the storage of individual secrets up to 8KiB. For cryptographic keys, multiple key types and algorithms are supported.

Jana's individual "vaults" may contain a mix of keys and secrets, and access control for the two types of object is independently controlled.

1.2 Elements

Jana's architecture is composed of several elements:

- Users
- Vaults
- Secrets
- Keys
- Applications

Users are the top-level of the system. One or more users have access to manage individuals vaults. Vaults are the overall container for secrets and keys. Finally, applications are granted access to a vault.

1.3 Features

Features can best be described as the actions one can do with various things within the system. Users, assuming authorization, may:

- Manage keys using import, update, rotate, and delete operations;

- Manage secrets using get, set, and delete operations.

Keys are effectively specialized versions of secrets, where the metadata associated with the secret is focused on that associated with cryptographic algorithms.

In addition, the following are special areas of focus for the design and implementation:

- Full auditing of all activities, both internally, and with optional remote storage.
- Careful validation of all input and output. Any instance of suspicious activity or input is immediately dropped and audit trails generated.

1.4 Limitations

- Currently, all authentication and authorization is self-contained within Jana. Whether this changes in the future is subject to consideration. Opening up is also increasing the threat model to encompass other systems.
- Many, many missing features.

1.5 Future Plans

There are a lot of future plans, but the high-level ones are:

- Implement cryptographic operations that can be performed inside Jana so that keys never need to be disclosed.
- Cryptographicly-strong key generation.

1.6 Name

The name comes from Jana, the Etruscan goddess and consort of Janus. Mythology says that she was the keeper of secrets, mysteries, and hidden things. Her name, along with that of Janus, would pass on into Roman mythology as so many things did.

CHAPTER 2

Installation

At the command line:

```
$ easy_install jana
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv jana
$ pip install jana
```

CHAPTER 3

Dependencies

With something that is as security-sensitive as Jana, it is important to carefully consider all third-party dependencies that are included. So, in addition to the existing Python standard library, the following libraries are included.

Flask Flask is a small web framework. One of the things going for it, and the reason it was chosen, is that it is very small, and therefore has limited functionality.

Flask-Restless This library provides basic RESTfun API functionality.

structlog While the standard library's logging module is sufficient for many applications, in this situation it was critical to be able to have very structured logging. This library was written specifically to allow for flexible, yet structured, logging. It is well tested.

jsonschema Provides validation of JSON Schema descriptions of all REST calls. This allows us to ensure that nothing goes in or out of the system that isn't approved.

CHAPTER 4

Usage

To use Jana in a project:

```
import jana
```

CHAPTER 5

Deployment

Proper deployment configuration is critical to a secure solution. This chapter contains detailed deployment recommendations.

5.1 Database

While it would have been possible to make Jana work with other databases, there is only a single one supported: PostgreSQL. In order to deploy Jana, the following steps need to be followed. Unlike most instructions, these will guide you through using the psql command-line interface (CLI), rather than the administrative commands.

5.1.1 Initial Setup

First, you need to create two distinct users in PostgreSQL. The easiest way to do this is to launch `psql` as a database superuser:

```
$ psql -U postgres
psql (9.4.0)
Type "help" for help.

postgres=#
```

From here, we can create the users we need:

```
CREATE USER jana_admin WITH LOGIN ENCRYPTED PASSWORD '<something>';
CREATE USER jana WITHLOGIN ENCRYPTED PASSWORD '<somethingdifferent>';
```

In each case, you should replace the `<something>` or `<somethingdifferent>` with the actual passwords. These should be as strong as possible and not guessable.

Next, we need to create the database:

```
CREATE DATABASE jana ENCODING 'UTF8';
```

This will create a UTF-8 encoded database. This allows us to safely store strings with non-ASCII characters in them.

5.1.2 Table Migration

The schema for Jana is managed through controlled migrations. These reflect incremental changes to the database that mirror the underlying code development. They are managed using the Alembic library.

5.1.3 Permissions Management

One of the things you should do once the initial tables are created is adjust the permissions. Unfortunately, the standard migrations and table creation scripts assume a single user with full access to the database. This isn't the best way to handle things from a security perspective. Instead, we want to remove some permissions from certain tables from the normal "session" user, `jana`. Once again, at a superuser prompt for `psql`, we want to revoke permissions:

```
REVOKE ALL PRIVILEGES ON TABLE audit FROM jana;
```

Now, we want to restore just the permissions we want the user to have:

```
GRANT SELECT ON TABLE audit TO jana;
GRANT INSERT ON TABLE audit TO jana;
```

CHAPTER 6

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

6.1 Types of Contributions

6.1.1 Report Bugs

Report bugs at https://github.com/petrilli/jana/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

6.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

6.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

6.1.4 Write Documentation

Jana could always use more documentation, whether as part of the official Jana docs, in docstrings, or even on the web in blog posts, articles, and such.

6.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/petrilli/jana/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

6.2 Get Started!

Ready to contribute? Here's how to set up *jana* for local development.

1. Fork the *jana* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/jana.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv jana
   $ cd jana/
   $ python setup.py develop
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

   ```
   $ flake8 jana tests
   $ python setup.py test
   $ tox
   ```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

6.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, and 3.4, and for PyPy. Check https://travis-ci.org/petrilli/jana/pull_requests and make sure that the tests pass for all supported Python versions.

6.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_jana
```

CHAPTER 7

Credits

7.1 Development Lead

- Christopher Petrilli <petrilli@amber.org>

7.2 Contributors

None yet. Why not be the first?

CHAPTER 8

History

CHAPTER 9

0.1.0 (2015-02-15)

- First release on PyPI.

CHAPTER 10

Indices and tables

- *genindex*
- *modindex*
- *search*