EuroPython

Edinburgh 23-29 July 2018

Hardening Plaintext Secrets in Configuration Files

Moisés Guimarães de Medeiros - Software Engineer - Red Hat

Introduction

Many applications and services rely on configuration data to be able to interact with each other and to function according to its purpose. Plaintext configuration files are widely used to separate configuration from code as configuration data varies substantially across deployments while code does not.

OpenStack Common Libraries (Oslo) provides an enhanced alternative to Python's standard module ConfigParser called oslo.config. It supports configuration files, command line arguments, option deprecation, and much more.

The motivation of this work is to give oslo.config the ability to fetch sensitive configuration data (secrets) from places that are better equipped to deal with them.

Problem Description

Best practices say that passwords and other secret values should not be stored as plain text in configuration files and some regulations might even enforce this practice as mandatory.

Although we can rely on file system permissions to restrict access to a configuration file, its content can still be accidentally shared, checked into revision control, or printed to a console or log file, without having all sensitive data stripped out of it.

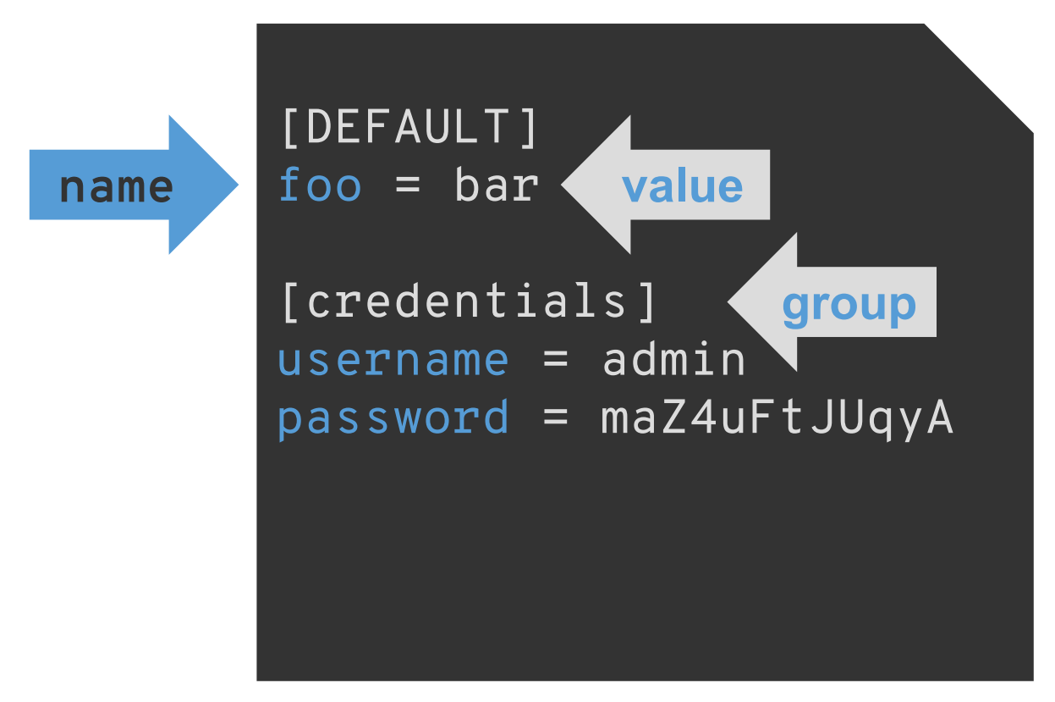
Also, with the popularization of containers and public clouds, there is a higher concern about data getting compromised as we don't know what else might be running on the same machine.

Proposed Solution

There are proper solutions for storing this kind of sensitive data called secret managers. They have features to handle access control, password rotation, data encryption/decryption, X.509 certificates and can also interface with Hardware Security Modules (HSM) if the security requirements are set that high.

OpenStack has its own secret manager service called Barbican, and HashiCorp Vault is one of the best opensource solutions out there. Both are very interesting options to connect oslo.config to, but to provide more flexibility, we defined an API for backend drivers in oslo.config, so it could easily integrate with more options in the future.

In a few words, a config option in oslo.config is identified by its `name` and `group`, when the group is not defined, the option automatically belongs to the `DEFAULT` group. We can see a sample config file with two options below, one belonging to the `DEFAULT` group and another to the `credentials` group:



```
[DEFAULT]
foo = bar

[credentials]
username = admin
password = maZ4uFtJUqyA
```

A new configuration option `DEFAULT.config_source` is added to define extra sources of configuration data. The value for `config_source` is a list of source identifiers used to find configuration settings for other sources. Each source identifier corresponds to a configuration option group, which provides the details for a single source of configuration data.

Each config group representing an extra config source must have an option named `driver` to identify the proper drive to handle it.

When oslo.config looks for an option value, it goes through the defined sources in the order they are provided, starting with the command line, then any configuration files, and finally the sources loaded from `config_source`. The first source that provides a configured value for an option causes the search to end.

Proof of Concept

The Proof of Concept (PoC) of this work aimed to fulfill the following criteria:

1. Implement a simple driver to fetch remote configuration data;
2. Strip out secrets from a configuration file;
3. Store secrets in a new configuration source;
4. Give oslo.config the ability to fetch those secrets.

We created the `remote_file` driver to fetch and parse extra configuration files from an http[s] server with the following options.

`driver = remote_file`

The name of the driver that can load this configuration source.

`uri`

Required option with the URI of the extra configuration file's location.

`ca_path`

The path to a CA_BUNDLE file or directory with certificates of trusted CAs.

`client_cert`

Client side certificate, as a single file path containing either the certificate only or the private key and the certificate.

`client_key`

Client side private key, in case client_cert is specified but does not include the private key.

We can see on the PoC Scenarios session how the configuration files look like before and after the usage of the `remote_file` driver.

Conclusions

By fetching secrets from other sources, one can include extra layers of protection to them and take advantage of features provided by secret managers.

Using the `remote_file` driver in its full capability, it is possible to deny access to secrets by revoking the certificate of a compromised deployment. This requires each deployment to have its own certificate, and `SSL/TLS` client authentication on the `HTTP` server.

Having the secrets to live in a single place also makes the process of password rotation easier as the new password only has to be updated in one place instead of in all configuration files that requires it.

The lifespan of a deployment's certificate can be reduced to enforce secrets leasing for a short time period and minimize the attack window using compromised keys.

Future Work

Castellan is a generic key manager interface developed by the Barbican team. It enables projects to use a configurable key manager that can be deployment specific.

The next step in our roadmap is to create an oslo.config driver for Castellan. This will give oslo.config the capability to retrieve secrets from all of the key manager solutions supported by Castellan, giving access to Barbican or HashiCorp Vault through a single driver implementation.

Direct drivers to Barbican or HashiCorp Vault can also be implemented, but as far as oslo.config doesn't require features other than retrieval of secrets, it makes sense to use Castellan as the lowest-common-denominator instead.

Developers interested in using another solution as an extra source of configuration data can implement their own drivers following this spec:

https://specs.openstack.org/openstack/oslo-specs/specs/queens/oslo-config-drivers

PoC Scenarios

BEFORE

Our application is using configuration files in the INI format. There are a few passwords stored directly in the configuration file `my_app.conf`. This file lives within the deployment of our application and to scale it also needs to be running in a large number of machines, making copies of our passwords all over the cloud.

`file: /etc/my_app.conf`

```
[DEFAULT]
mysql_username = my_app
mysql_password = secretPasswordABCDEF

[dogtag_plugin]
nss_db_path = /etc/my_app/alias
nss_password = secretPassword123456

[kmip_plugin]
username = kmip1923804
password = secretPassword654321
```

AFTER

After identifying the secrets in our configuration file we setup a dedicated server at the arbitrary address `192.168.42.42`. On this server, an `HTTPS` server that allows only client authenticated connections is deployed to serve our `secrets.conf` file containing the secrets stripped out of `my_app.conf`.

The `my_app.conf` file is updated to contain the `remote_file` driver configuration in order to be able to reach the secrets.

Each instance of our application has its own key pair to authenticate to the secrets server making it possible to instantly revoke access to the secrets for a single node. A short lifespan on the client's certificate can also reduce the attack window if by any chance the client keys are compromised.

`file: /etc/my_app.conf`

```
[DEFAULT]
config_source = secrets

[secrets]
driver = remote_file
uri = https://192.168.42.42/secrets.conf
client_cert = /etc/ca-certificates/cert.pem
client_key = /etc/ca-certificates/key.pem

[dogtag_plugin]
nss_db_path = /etc/my_app/alias
```

`file: secrets.conf at 192.168.42.42`

```
[DEFAULT]
mysql_username = my_app
mysql_password = secretPasswordABCDEF

[dogtag_plugin]
nss_password = secretPassword123456

[kmip_plugin]
username = kmip1923804
password = secretPassword654321
```

OpenStack

"OpenStack is a set of software tools for building and managing cloud computing platforms for public and private clouds. Backed by some of the biggest companies in software development and hosting, as well as thousands of individual community members"

https://openstack.org

Oslo

"The Oslo project is a collection of over 30 libraries that are designed to reduce the technical debt of code duplication across projects and provide for a greater quality code path due to the frequency of use in OpenStack projects."

http://ronaldbradford.com

Barbican

"Barbican is the OpenStack Key Manager service. It provides secure storage, provisioning and management of secret data, such as passwords, encryption keys, X.509 Certificates and raw binary data."

https://www.openstack.org

HashiCorp Vault

"HashiCorp Vault secures, stores, and tightly controls access to tokens, passwords, certificates, API keys, and other secrets in modern computing. Vault handles leasing, key revocation, key rolling, and auditing."

https://www.vaultproject.io