COLA

Generated by Doxygen 1.8.15

Contents

Chapter 1

Component-security-policy-manager

v1.0:

Overview:

This module provides APIs to manage sensitive information, including application sensitive information and infrastructure sensitive information.

Infrastructure sensitive information or infrastructure secret:

- Initialize a vault to store secrets for the infrastructure
- Create or update a secret
- Read a secret
- Delete a secret

Application sensitive information or application secret:

- Add application sensitive information as docker secret
- Provision the secret to application services in worker nodes

How to use the API:

Infrastructure sensitive information or infrastructure secret

- Initialize a vault to store secrets

curl -d "shares=3&threshold=2" -X POST spm:5003/v1.0/vault

(Shares must be equal or larger than threshold. If shares $>$ 1, threshold must be larger than 1)

- Add a secret 'secret1' into the initialized vault. If the secret exists, it will be overwritten.

curl -d "name=secret1&value=123" -X POST spm:5003/v1.0/secrets

- Update the secret named 'secret1' with a new value

curl -d "name=secret1&value=456" -X PUT spm:5003/v1.0/secrets

- Read a secret named 'secret1' from the vault

curl -d "name=secret1" -X GET spm:5003/v1.0/secrets

- Delete a secret named 'secret1' from the vault

curl -d "name=secret1" -X DELETE spm:5003/v1.0/secrets

Application sensitive information or application secret

- Assuming that a service named app1 is already created
- Add an applicaton sensitive information as docker secret and distribute it to containers of the application app1 (If the application service has existing secrets, this function add one more while keeping the other secrets intact):

curl -d "secret_name=db_pass1&secret_value=123&service=app1" -X POST spm:5003/v1.0/appsecrets

- Verify if the secret is added to the service or not by calling the below command line in the master node

docker service inspect app1

How to use command line in the master node

Infrastructure sensitive information or infrastructure secret

- Initialize a vault to store secrets with shares = 2, threshold = 2

micadoctl.sh initvault 2 2

- Add a secret 'secret1' with value 123 into the initialized vault. If the secret exists, it will be overwritten.

micadoctl.sh addsecret secret1 123

- Read a secret named 'secret1' from the vault

micadoctl.sh readsecret secret1

- Delete a secret named 'secret1' from the vault

micadoctl.sh removesecret secret1

Application sensitive information or application secret

- Add a secret 'secret1' with value 123 in docker secrets and provision it to the service 'app1' (assuming that the service 'app1' is deployed already)

micadoctl.sh addappsecret secret1 123 app1

- Check if 'secret1' is provisioned to 'app1' or not

docker service inspect app1

You shall see something like this

Spec": {

```
    "TaskTemplate": {
            "ContainerSpec": {
                "Secrets": [
                    {
                        "File": {
                            "Name": "secret1",
                        },
                        "SecretID":,
                        "SecretName": "secret1"
                    },
                    ...
```

How to use the automatic test script for managing secrets infrastructure sensitive information:

Assuming that you installed Robot framework successfully (Please follow this link if you has not installed the Robot framework yet: https://github.com/robotframework/QuickStartGuide/blob/master/↩ QuickStart.rst#demo-application)

- Launch the vault server in localhost ++ Download the vault server from https://www.vaultproject.↩ io/downloads.html ++ Create a config file named vault.hcl with the below content:

storage "file" {

path = "datafile"

}

listener "tcp" {

address = "127.0.0.1:8200"

tls_disable = 1

}

(all secrets will be written in the file 'datafile' which resides in the same directory with the executable file 'vault')

++ Launch the vault server by command line

./vault server -config=vault.hcl

- Edit the file app/vaultclient.py to change VAULT_URL into

VAULT_URL = "http://127.0.0.1:8200"

- Run the source code by command line

python my_script.py

- Run the test script by command line

robot test_script.rst

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 app Namespace Reference

Namespaces

- dksecrets
- routes
- vaultclient

Variables

- app = Flask(__name__)
- logHandler = RotatingFileHandler('error.log', maxBytes=1000, backupCount=1)
- formatter = logging.Formatter('%(asctime)s - %(name)s - %(module)s - %(funcName)s - %(lineno)d- %(levelname)s - %(message)s')

6.1.1 Variable Documentation

6.1.1.1 app

```
app.app = Flask(__name__)
```

6.1.1.2 formatter

```
app.formatter = logging.Formatter('%(asctime)s - %(name)s - %(module)s - %(funcName)s - %(lineno)d-
%(levelname)s - %(message)s')
```

6.1.1.3 logHandler

```
app.logHandler = RotatingFileHandler('error.log', maxBytes=1000, backupCount=1)
```

6.2 app.dksecrets Namespace Reference

Functions

- def create_secret (secretname, secretvalue)
- def add_secret_api ()

Variables

- reader = csv.DictReader(open('resource_dksecret.csv', 'r'))
- dictionary msg_dict = {}
- int HTTP_CODE_OK = 200
- int HTTP_CODE_CREATED = 201
- int HTTP_CODE_BAD_REQUEST = 400

6.2.1 Function Documentation

6.2.1.1 add_secret_api()

```
def app.dksecrets.add_secret_api ( )
```

```
[summary]
Creates a docker secret and distribute it to docker service
[description]
Input:
    name -- secret's name
    value -- secret's value
    service -- service' name
Assuming that the service exists, this API creates a docker secret from the inputted name and value, then dist
Returns:
    [type] Json object -- [description] Successful message or error message
```

```
46 def add_secret_api():
47     '''[summary]
48     Creates a docker secret and distribute it to docker service
49     [description]
50     Input:
51         name -- secret's name
52         value -- secret's value
53         service -- service' name
54     Assuming that the service exists, this API creates a docker secret from the inputted name and value,
       then distribute the docker secret to the corresponding service
55     Returns:
56         [type] Json object -- [description] Successful message or error message
57     '''
58
59     # Parse query arguments
60     parser = reqparse.RequestParser()
61     parser.add_argument('name', help='Secret name')
62     parser.add_argument('value', help='Secret value')
63     parser.add_argument('service', help='Application/ service name')
```

```
64
65      args = parser.parse_args()
66      secret_name = args['name']
67      secret_value = args['value']
68      service_name = args['service']
69
70      # Verify query arguments
71      if(secret_name is None or secret_value  is None or service_name is None or secret_name=='' or
        secret_value=='' or service_name==''):
72          data = {
73              'code' : HTTP_CODE_BAD_REQUEST,
74              'user message'   : msg_dict['bad_request_write_dksecret'],#'Bad request',
75              'developer message' : msg_dict['bad_request_write_dksecret']
76          }
77          js = json.dumps(data)
78          resp = Response(js, status=HTTP_CODE_BAD_REQUEST, mimetype='application/json')
79          return resp
80
81      # Create docker secret and add it to corresponding service
82      try:
83          secret_id = create_secret(secret_name, secret_value) # Create docker secret
84      except Exception as e:
85          data = {
86              'code' : HTTP_CODE_BAD_REQUEST,
87              'user message'   : msg_dict['existed_secret'],#'Bad request',
88              'developer message' : msg_dict['existed_secret']
89          }
90          js = json.dumps(data)
91          resp = Response(js, status=HTTP_CODE_BAD_REQUEST, mimetype='application/json')
92          return resp
93
94      try:
95          client = docker.APIClient(base_url='unix://var/run/docker.sock')
96          service = client.services(filters={"name" : service_name}).pop(0) # Get the service by name
97      except Exception as e:
98          app.logger.error(e)
99          data = {
100             'code' : HTTP_CODE_BAD_REQUEST,
101             'user message'   : msg_dict['non_exist_service'],#'Bad request',
102             'developer message' : msg_dict['non_exist_service']
103         }
104         js = json.dumps(data)
105         resp = Response(js, status=HTTP_CODE_BAD_REQUEST, mimetype='application/json')
106         return resp
107
108     service_id = service['ID'] # get service_id
109     service_version = service['Version']['Index'] # get service_version
110
111     container_spec = service['Spec']['TaskTemplate']['ContainerSpec'] #get service container specification
112     current_secrets = []
113     try:
114         current_secrets = container_spec['Secrets'] # get the service's current secret list (if existed)
115     except:
116         pass
117
118     secret_list = [docker.types.SecretReference(secret_id,secret_name)] # Create list of SecretReference
119     container_spec['Secrets']=current_secrets+secret_list # update list of secrets for the service
120     task_tmpl = docker.types.TaskTemplate(container_spec) # Create TaskTemplate from container
        specification
121     #print "task template"
122     #print task_tmpl.container_spec
123
124     client.update_service(service=service_id,name=service_name,version=service_version,task_template=
        task_tmpl) # Update service with new secret list
125     #print "config of service after update"
126     #print client.inspect_service(service=service_name)
127
128     data = {
129         'code' : HTTP_CODE_CREATED,
130         'user message'   : msg_dict['write_dksecret_success'],
131         'developer message'   : msg_dict['write_dksecret_success']
132     }
133     js = json.dumps(data)
134     resp = Response(js, status=HTTP_CODE_CREATED, mimetype='application/json')
135
136     return resp
```

6.2.1.2 create_secret()

```
def app.dksecrets.create_secret (
```

```
                secretname,
                secretvalue )


[summary]
Create a docker secret
[description]
    secretname -- name of secret
    secretvalue -- value of secret
Returns:
    [type] Integer -- [description] Id of the created secret

Raises:
    e -- [description]
```

```
23 def create_secret(secretname, secretvalue):
24     '''[summary]
25     Create a docker secret
26     [description]
27         secretname -- name of secret
28         secretvalue -- value of secret
29     Returns:
30         [type] Integer -- [description] Id of the created secret
31
32     Raises:
33         e -- [description]
34     '''
35     client = docker.DockerClient(base_url='unix://var/run/docker.sock')
36     try:
37         secret=client.api.create_secret(name=secretname,data=secretvalue.encode('utf-8'))
38         secretid=secret.get('ID')
39
40     except Exception as e:
41         app.logger.error(e)
42         raise
43
44     return secretid
45
```

6.2.2 Variable Documentation

6.2.2.1 HTTP_CODE_BAD_REQUEST

```
int app.dksecrets.HTTP_CODE_BAD_REQUEST = 400
```

6.2.2.2 HTTP_CODE_CREATED

```
int app.dksecrets.HTTP_CODE_CREATED = 201
```

6.2.2.3 HTTP_CODE_OK

```
int app.dksecrets.HTTP_CODE_OK = 200
```

6.2.2.4 msg_dict

```
dictionary app.dksecrets.msg_dict = {}
```

6.2.2.5 reader

```
app.dksecrets.reader = csv.DictReader(open('resource_dksecret.csv', 'r'))
```

6.3 app.routes Namespace Reference

Variables

- add_secret_api
- methods
- init_vault_api
- write_secret_api
- read_secret_api
- delete_secret_api

6.3.1 Variable Documentation

6.3.1.1 add_secret_api

```
app.routes.add_secret_api
```

6.3.1.2 delete_secret_api

```
app.routes.delete_secret_api
```

6.3.1.3 init_vault_api

```
app.routes.init_vault_api
```

6.3.1.4 methods

app.routes.methods

6.3.1.5 read_secret_api

app.routes.read_secret_api

6.3.1.6 write_secret_api

app.routes.write_secret_api

6.4 app.vaultclient Namespace Reference

Functions

- def init_vault_api ()
- def read_token ()
- def read_unseal_keys ()
- def init_client ()
- def unseal_vault (client)
- def seal_vault (client)
- def write_secret_api ()
- def read_secret_api ()
- def delete_secret_api ()

Variables

- reader = csv.DictReader(open('resource_vault.csv', 'r'))
- dictionary msg_dict = {}
- string VAULT_URL = "http://credstore:8200"
- int HTTP_CODE_OK = 200
- int HTTP_CODE_CREATED = 201
- int HTTP_CODE_BAD_REQUEST = 400
- int HTTP_CODE_NOT_FOUND = 404
- int HTTP_CODE_SERVER_ERR = 500
- int DEFAULT_SHARES = 1
- int DEFAULT_THRESHOLD = 1
- string VAULT_TOKEN_FILE = 'vaulttoken'
- string UNSEAL_KEYS_FILE = 'unsealkeys'

6.4.1 Function Documentation

6.4.1.1 delete_secret_api()

```
def app.vaultclient.delete_secret_api ( )
```

[summary]
Remove a secret from the vault
[description]

```
302 def delete_secret_api():
303     """[summary]
304     Remove a secret from the vault
305     [description]
306     """
307 
308     parser = reqparse.RequestParser()
309     parser.add_argument('name',help='Name of sensitive information')
310 
311     args = parser.parse_args()
312     secret_name = args['name']
313 
314     if(secret_name is None or  secret_name==''): # verify parameters
315         data = {
316             'code' : HTTP_CODE_BAD_REQUEST,
317             'user message'  : msg_dict['bad_request_delete_secret'],#'Lack of secret name',
318             'developer message' : msg_dict['bad_request_delete_secret']
319         }
320         js = json.dumps(data)
321         resp = Response(js, status=HTTP_CODE_OK, mimetype='application/json')
322         return resp
323 
324     # unseal the vault
325     client = init_client()
326     try:
327         unseal_vault(client)
328     except Exception as e:
329         app.logger.error(e)
330         data = {
331             'code' : HTTP_CODE_SERVER_ERR,
332             'user message'  : msg_dict['vault_not_initialized'],#'Add secret successfully',
333             'developer message' : msg_dict['vault_not_initialized']
334         }
335         js = json.dumps(data)
336         resp = Response(js, status=HTTP_CODE_SERVER_ERR, mimetype='application/json')
337         return resp
338 
339     client.delete('secret/'+secret_name)
340     seal_vault(client)
341 
342     data = {
343         'code' : HTTP_CODE_OK,
344         'user message'  : msg_dict['delete_secret_success'],#'Delete secret successfully',
345         'developer message' : msg_dict['delete_secret_success']
346     }
347     js = json.dumps(data)
348     resp = Response(js, status=HTTP_CODE_OK, mimetype='application/json')
349     return resp
350 
```

6.4.1.2 init_client()

```
def app.vaultclient.init_client ( )
```

[summary]
Initialize the vault client
[description]

Returns:
  [type] -- [description]

```
152 def init_client():
153     """[summary]
154     Initialize the vault client
155     [description]
156 
157     Returns:
158       [type] -- [description]
159     """
160     client = hvac.Client(url=VAULT_URL)
161     return client
162 
```

6.4.1.3 init_vault_api()

```
def app.vaultclient.init_vault_api ( )
```

```
[summary]
Initialize a vault in the Vault Server (Credential Store)
[description]
A number of keys will be generated from the master key, then the master key is thrown away (The Server will nc
shares = The number of generated keys
threshold = The minimum number of generated keys needed to unseal the vault
```

```
46 def init_vault_api():
47     """[summary]
48     Initialize a vault in the Vault Server (Credential Store)
49     [description]
50     A number of keys will be generated from the master key, then the master key is thrown away (The Server
       will not store the key). The generated keys are kept by the Vault Client (Security Policy Manager)
51     shares = The number of generated keys
52     threshold = The minimum number of generated keys needed to unseal the vault
53     """
54     parser = reqparse.RequestParser()
55     parser.add_argument('shares', type=int, help='Default: threshold=1, shares=1', default=DEFAULT_SHARES)
56     parser.add_argument('threshold', type=int, help='Default: threshold=1, shares=1', default =
      DEFAULT_THRESHOLD)
57 
58     args = parser.parse_args()
59     shares = args['shares']
60     threshold = args['threshold']
61 
62     if(threshold>shares or (shares>=2 and threshold==1) or shares<=0 or threshold<=0):
63         data = {
64             'code' : HTTP_CODE_BAD_REQUEST,
65             'user message'   : msg_dict['init_vault_fail_due_to_parameter'],
66             'developer message' : msg_dict['init_vault_fail_due_to_parameter']
67         }
68         js = json.dumps(data)
69         resp = Response(js, status=HTTP_CODE_BAD_REQUEST, mimetype='application/json')
70         return resp
71 
72     vault_exist = False
73     try:
74         client = init_client()
75         vault_exist = client.is_initialized()
76     except Exception as e:
77         app.logger.error(e)
78         data = {
79             'code' : HTTP_CODE_SERVER_ERR,
80             'user message'   : msg_dict['init_vault_fail'],#'Fail to Initialize vault',
81             'developer message' : msg_dict['init_vault_fail']
82         }
83         js = json.dumps(data)
84         resp = Response(js, status=HTTP_CODE_SERVER_ERR, mimetype='application/json')
85         return resp
86 
87     if(vault_exist): # if vault existed
88         data = {
89             'code' : HTTP_CODE_CREATED,
90             'user message'   : msg_dict['vault_existed'],#Vault is existed
91             'developer message' : msg_dict['vault_existed']
92         }
93     else:
94         vault = client.initialize(shares,threshold)
```

```
95          root_token = vault['root_token']
96          unseal_keys = vault['keys']
97          # write root token into file
98          f = open(VAULT_TOKEN_FILE, 'w')
99          f.write(root_token)
100          f.close()
101
102          # write unseal_keys into file
103          f = open(UNSEAL_KEYS_FILE,'w')
104          for key in unseal_keys:
105              f.write("%s\n" % key)
106          f.close()
107          data = {
108              'code' : HTTP_CODE_CREATED,
109              'user message'  : msg_dict['init_vault_success'],#'Initialize vault successfully',
110              'developer message' : msg_dict['init_vault_success']
111          }
112
113      js = json.dumps(data)
114      resp = Response(js, status=HTTP_CODE_CREATED, mimetype='application/json')
115      return resp
116
117
```

6.4.1.4 read_secret_api()

```
def app.vaultclient.read_secret_api ( )
```

```
[summary]
Read a secret from the vault
[description]

Returns:
  [type] json -- [description] a dictionary of all relevant information of the secret
```

```
240 def read_secret_api():
241     """[summary]
242     Read a secret from the vault
243     [description]
244
245     Returns:
246       [type] json -- [description] a dictionary of all relevant information of the secret
247     """
248
249     parser = reqparse.RequestParser()
250     parser.add_argument('name',help='Name of sensitive information')
251
252     args = parser.parse_args()
253     secret_name = args['name']
254
255     #print secret_name
256
257     if(secret_name is None or  secret_name==''): # verify parameters
258         data = {
259             'code' : HTTP_CODE_BAD_REQUEST,
260             'user message'  : msg_dict['bad_request_read_secret'],#'Add user successfully',
261             'developer message' : msg_dict['bad_request_read_secret']
262         }
263         js = json.dumps(data)
264         resp = Response(js, status=HTTP_CODE_OK, mimetype='application/json')
265         return resp
266     # unseal the vault
267     client = init_client()
268     try:
269         unseal_vault(client)
270     except Exception as e:
271         app.logger.error(e)
272         data = {
273             'code' : HTTP_CODE_SERVER_ERR,
274             'user message'  : msg_dict['vault_not_initialized'],#'Add secret successfully',
275             'developer message' : msg_dict['vault_not_initialized']
276         }
277         js = json.dumps(data)
278         resp = Response(js, status=HTTP_CODE_SERVER_ERR, mimetype='application/json')
```

```
279         return resp
280     secret_values = client.read('secret/'+secret_name)
281     seal_vault(client)
282 
283     if(secret_values is None): # If the required secret does not exist
284         data = {
285             'code' : HTTP_CODE_NOT_FOUND,
286             'user message'  : msg_dict['secret_not_exist'],#'Secret does not exist',
287             'developer message' : msg_dict['secret_not_exist']
288         }
289         js = json.dumps(data)
290     else:
291         data = {
292             'code' : HTTP_CODE_OK,
293             'user message'  : msg_dict['read_secret_success'],#'Read secret successfully',
294             'developer message' : msg_dict['read_secret_success']
295         }
296         data.update(secret_values)
297         js = json.dumps(data)
298 
299     resp = Response(js, status=HTTP_CODE_OK, mimetype='application/json')
300     return resp
301 
```

6.4.1.5 read_token()

```
def app.vaultclient.read_token ( )
```

```
[summary]
Read the token from file 'vaulttoken'
[description]

Returns:
  [type] string -- [description] the token
```

```
118 def read_token():
119     """[summary]
120     Read the token from file 'vaulttoken'
121     [description]
122 
123     Returns:
124       [type] string -- [description] the token
125     """
126     try:
127         f = open(VAULT_TOKEN_FILE, 'r')
128         root_token = f.read()
129         f.close()
130         return root_token
131     except Exception as e:
132         app.logger.error(e)
133         raise
134 
```

6.4.1.6 read_unseal_keys()

```
def app.vaultclient.read_unseal_keys ( )
```

```
[summary]
Read keys used to unseal the vault from file 'unsealkeys'
[description]

Returns:
  [type] List -- [description] List of keys
```

```
135 def read_unseal_keys():
136     """[summary]
137     Read keys used to unseal the vault from file 'unsealkeys'
138     [description]
139
140     Returns:
141       [type] List -- [description] List of keys
142     """
143     try:
144         f = open(UNSEAL_KEYS_FILE, 'r')
145         unseal_keys = f.read().splitlines()
146         f.close()
147         return unseal_keys
148     except Exception as e:
149         app.logger.error(e)
150         raise
151
```

6.4.1.7 seal_vault()

```
def app.vaultclient.seal_vault (
              client )

[summary]
Seal the vault
[description]
This should be done to protect the vault while not using it
Arguments:
  vault client {[type]} -- [description] vault client
```

```
176 def seal_vault(client):
177     """[summary]
178     Seal the vault
179     [description]
180     This should be done to protect the vault while not using it
181     Arguments:
182       vault client {[type]} -- [description] vault client
183     """
184     client.seal()
185
```

6.4.1.8 unseal_vault()

```
def app.vaultclient.unseal_vault (
              client )

[summary]
Unseal (open) the vault
[description]
This must be done prior to read contents from the vault.
Arguments:
  vault client {[type]} -- [description] vault client
```

```
163 def unseal_vault(client):
164     """[summary]
165     Unseal (open) the vault
166     [description]
167     This must be done prior to read contents from the vault.
168     Arguments:
169       vault client {[type]} -- [description] vault client
170     """
171     client.token = read_token()
172     # unseal the vault
173     unseal_keys = read_unseal_keys()
174     client.unseal_multi(unseal_keys)
175
```

6.4.1.9 write_secret_api()

```
def app.vaultclient.write_secret_api ( )

[summary]
Write/ update a secret to the vault
[description]
Arguments:
    name -- name of secret
    value -- value of secret
```

```
186 def write_secret_api():
187     """[summary]
188     Write/ update a secret to the vault
189     [description]
190     Arguments:
191         name -- name of secret
192         value -- value of secret
193     """
194 
195     parser = reqparse.RequestParser()
196     parser.add_argument('name',  help='Name of sensitive information')
197     parser.add_argument('value', help='Value of sensitive information')
198 
199     args = parser.parse_args()
200     secret_name = args['name']
201     secret_value = args['value']
202 
203     if(secret_name is None or secret_value  is None or secret_value=='' or  secret_name==''): # verify
    parameters
204         data = {
205             'code' : HTTP_CODE_BAD_REQUEST,
206             'user message'  : msg_dict['bad_request_write_secret'],#'Bad request',
207             'developer message' : msg_dict['bad_request_write_secret']
208         }
209         js = json.dumps(data)
210         resp = Response(js, status=HTTP_CODE_BAD_REQUEST, mimetype='application/json')
211         return resp
212 
213     client = init_client()
214     try:
215         unseal_vault(client)
216     except Exception as e:
217         app.logger.error(e)
218         data = {
219             'code' : HTTP_CODE_SERVER_ERR,
220             'user message'  : msg_dict['vault_not_initialized'],#'Add secret successfully',
221             'developer message' : msg_dict['vault_not_initialized']
222         }
223         js = json.dumps(data)
224         resp = Response(js, status=HTTP_CODE_SERVER_ERR, mimetype='application/json')
225         return resp
226 
227     #print client
228     client.write('secret/'+secret_name, secret_value=secret_value)#, lease='1h'
229     seal_vault(client)
230 
231     data = {
232         'code' : HTTP_CODE_CREATED,
233         'user message'  : msg_dict['write_secret_success'],#'Add secret successfully',
234         'developer message' : msg_dict['write_secret_success']
235     }
236     js = json.dumps(data)
237     resp = Response(js, status=HTTP_CODE_OK, mimetype='application/json')
238     return resp
239 
```

6.4.2 Variable Documentation

6.4.2.1 DEFAULT_SHARES

```
int app.vaultclient.DEFAULT_SHARES = 1
```

6.4.2.2 DEFAULT_THRESHOLD

```
int app.vaultclient.DEFAULT_THRESHOLD = 1
```

6.4.2.3 HTTP_CODE_BAD_REQUEST

```
int app.vaultclient.HTTP_CODE_BAD_REQUEST = 400
```

6.4.2.4 HTTP_CODE_CREATED

```
int app.vaultclient.HTTP_CODE_CREATED = 201
```

6.4.2.5 HTTP_CODE_NOT_FOUND

```
int app.vaultclient.HTTP_CODE_NOT_FOUND = 404
```

6.4.2.6 HTTP_CODE_OK

```
int app.vaultclient.HTTP_CODE_OK = 200
```

6.4.2.7 HTTP_CODE_SERVER_ERR

```
int app.vaultclient.HTTP_CODE_SERVER_ERR = 500
```

6.4.2.8 msg_dict

```
dictionary app.vaultclient.msg_dict = {}
```

6.4.2.9 reader

```
app.vaultclient.reader = csv.DictReader(open('resource_vault.csv', 'r'))
```

6.4.2.10 UNSEAL_KEYS_FILE

```
string app.vaultclient.UNSEAL_KEYS_FILE = 'unsealkeys'
```

6.4.2.11 VAULT_TOKEN_FILE

```
string app.vaultclient.VAULT_TOKEN_FILE = 'vaulttoken'
```

6.4.2.12 VAULT_URL

```
string app.vaultclient.VAULT_URL = "http://credstore:8200"
```

6.5 app_linebr Namespace Reference

6.5.1 Detailed Description

```
[summary]
Init module
[description]
The init module creates Flask object, databases, and logging handler
```

6.6 CredStoreLibrary Namespace Reference

Classes

- class CredStoreLibrary

Variables

- int http_code_ok = 200

6.6.1 Variable Documentation

6.6.1.1 http_code_ok

```
int CredStoreLibrary.http_code_ok = 200
```

6.7 my_script Namespace Reference

Variables

- host
- port

6.7.1 Variable Documentation

6.7.1.1 host

```
my_script.host
```

6.7.1.2 port

```
my_script.port
```

6.8 my_script_linebr Namespace Reference

6.8.1 Detailed Description

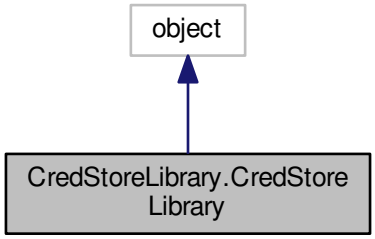
```
[summary]
Main module.
[description]
The main module starts the web service
```

Chapter 7

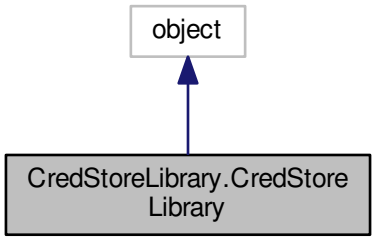
Class Documentation

7.1 CredStoreLibrary.CredStoreLibrary Class Reference

Inheritance diagram for CredStoreLibrary.CredStoreLibrary:



Collaboration diagram for CredStoreLibrary.CredStoreLibrary:



Public Member Functions

- def __init__ (self)
- def add_a_secret (self, name, value)
- def status_should_be (self, expected_status)
- def data_should_be (self, expected_data)
- def delete_a_secret (self, secretname=None)
- def read_a_secret (self, secretname=None)
- def init_a_vault (self, shares, threshold)

7.1.1 Constructor & Destructor Documentation

7.1.1.1 __init__()

```
def CredStoreLibrary.CredStoreLibrary.__init__ (
              self )
```

```
14     def __init__(self):
15         #self._sut_path = os.path.join(os.path.dirname(__file__),
16          #                             '..', 'sut', 'login.py')
17         self._status = ''
18         self._data = ''
19
```

7.1.2 Member Function Documentation

7.1.2.1 add_a_secret()

```
def CredStoreLibrary.CredStoreLibrary.add_a_secret (
              self,
              name,
              value )
```

```
25     def add_a_secret(self, name, value):
26         url     = 'http://127.0.0.1:5003/v1.0/secrets'
27         payload = {'name': name, 'value': value}
28         res = requests.post(url, data=payload)
29         json_data = json.loads(res.text)
30         self._status = json_data['code']
31
```

7.1.2.2 data_should_be()

```
def CredStoreLibrary.CredStoreLibrary.data_should_be (
              self,
              expected_data )
```

```
37     def data_should_be(self, expected_data):
38         if expected_data != str(self._data):
39             raise AssertionError("Expected data to be '%s' but was '%s'."
40                                  % (expected_data, self._data))
41
```

7.1.2.3 delete_a_secret()

```
def CredStoreLibrary.CredStoreLibrary.delete_a_secret (
              self,
              secretname = None )
```

```
42     def delete_a_secret(self, secretname=None,):
43         url     = 'http://127.0.0.1:5003/v1.0/secrets'
44         payload = {'name': secretname}
45         res = requests.delete(url, data=payload)
46         json_data = json.loads(res.text)
47         self._status = json_data['code']
48
```

7.1.2.4 init_a_vault()

```
def CredStoreLibrary.CredStoreLibrary.init_a_vault (
              self,
              shares,
              threshold )
```

```
58     def init_a_vault(self, shares, threshold):
59         url     = 'http://127.0.0.1:5003/v1.0/vault'
60         payload = {'shares': shares, 'threshold': threshold}
61         res = requests.post(url, data=payload)
62         json_data = json.loads(res.text)
63         self._status = json_data['code']
```

7.1.2.5 read_a_secret()

```
def CredStoreLibrary.CredStoreLibrary.read_a_secret (
              self,
              secretname = None )
```

```
49     def read_a_secret(self, secretname=None):
50         url     = 'http://127.0.0.1:5003/v1.0/secrets'
51         payload = {'name': secretname}
52         res = requests.get(url, data=payload)
53         json_data = json.loads(res.text)
54         self._status = json_data['code']
55         if(self._status == http_code_ok):
56             self._data = json_data['data']['secret_value']
57
```

7.1.2.6 status_should_be()

```
def CredStoreLibrary.CredStoreLibrary.status_should_be (
              self,
              expected_status )
```

```
32     def status_should_be(self, expected_status):
33         if expected_status != str(self._status):
34             raise AssertionError("Expected status to be '%s' but was '%s'."
35                                  % (expected_status, self._status))
36
```

The documentation for this class was generated from the following file:

- lib/CredStoreLibrary.py

Chapter 8

File Documentation

8.1 app/__init__.py File Reference

Namespaces

- app
- app_linebr

Variables

- app.app = Flask(__name__)
- app.logHandler = RotatingFileHandler('error.log', maxBytes=1000, backupCount=1)
- app.formatter = logging.Formatter('%(asctime)s - %(name)s - %(module)s - %(funcName)s - %(lineno)d-%(levelname)s - %(message)s')

8.2 app/dksecrets.py File Reference

Namespaces

- app.dksecrets

Functions

- def app.dksecrets.create_secret (secretname, secretvalue)
- def app.dksecrets.add_secret_api ()

Variables

- app.dksecrets.reader = csv.DictReader(open('resource_dksecret.csv', 'r'))
- dictionary app.dksecrets.msg_dict = {}
- int app.dksecrets.HTTP_CODE_OK = 200
- int app.dksecrets.HTTP_CODE_CREATED = 201
- int app.dksecrets.HTTP_CODE_BAD_REQUEST = 400

8.3 app/routes.py File Reference

Namespaces

- app.routes

Variables

- app.routes.add_secret_api
- app.routes.methods
- app.routes.init_vault_api
- app.routes.write_secret_api
- app.routes.read_secret_api
- app.routes.delete_secret_api

8.4 app/vaultclient.py File Reference

Namespaces

- app.vaultclient

Functions

- def app.vaultclient.init_vault_api ()
- def app.vaultclient.read_token ()
- def app.vaultclient.read_unseal_keys ()
- def app.vaultclient.init_client ()
- def app.vaultclient.unseal_vault (client)
- def app.vaultclient.seal_vault (client)
- def app.vaultclient.write_secret_api ()
- def app.vaultclient.read_secret_api ()
- def app.vaultclient.delete_secret_api ()

Variables

- app.vaultclient.reader = csv.DictReader(open('resource_vault.csv', 'r'))
- dictionary app.vaultclient.msg_dict = {}
- string app.vaultclient.VAULT_URL = "http://credstore:8200"
- int app.vaultclient.HTTP_CODE_OK = 200
- int app.vaultclient.HTTP_CODE_CREATED = 201
- int app.vaultclient.HTTP_CODE_BAD_REQUEST = 400
- int app.vaultclient.HTTP_CODE_NOT_FOUND = 404
- int app.vaultclient.HTTP_CODE_SERVER_ERR = 500
- int app.vaultclient.DEFAULT_SHARES = 1
- int app.vaultclient.DEFAULT_THRESHOLD = 1
- string app.vaultclient.VAULT_TOKEN_FILE = 'vaulttoken'
- string app.vaultclient.UNSEAL_KEYS_FILE = 'unsealkeys'

8.5 lib/CredStoreLibrary.py File Reference

Classes

- class CredStoreLibrary.CredStoreLibrary

Namespaces

- CredStoreLibrary

Variables

- int CredStoreLibrary.http_code_ok = 200

8.6 my_script.py File Reference

Namespaces

- my_script
- my_script_linebr

Variables

- my_script.host
- my_script.port

8.7 README.md File Reference