## Application for validating Error Detection Rules

|  |  |  |  |
| --- | --- | --- | --- |
| **Author** | **Changes** | **Version** | **Date** |
| Srikar Achanta, Technical Lead, AppD | Initial Draft | 0.1 | 09/27/2019 |

Table of Contents

[Application for validating Error Detection Rules 1](#_Toc20669301)

[Summary 1](#_Toc20669302)

[Configuring the Host 1](#_Toc20669303)

[Folder Structure 2](#_Toc20669304)

[Python Application 3](#_Toc20669305)

[Validate.py: 3](#_Toc20669306)

[fileCrypto.py: 8](#_Toc20669307)

[CryptoTool.py: 9](#_Toc20669308)

[Sample results.csv (output file): 10](#_Toc20669309)

[Sample temp.csv (input file): 11](#_Toc20669310)

[Steps to encrypt input file: 11](#_Toc20669311)

[Steps to start the program: 11](#_Toc20669312)

## Summary

The intention of this document is to provide details for building an application for detecting problematic REGEX rules in JAVA error detection rules. Scanning the error detection configuration for issues, will prevent excessive resource consumption by java agent or the application. This document contains python source code, that is built and tested to scan AppDynamics applications for REGEX expressions in java error detection rules; and provide a PASS or FAIL result for each application that uses REGEX based rules. In addition, this document also has details of an application to encrypt and decrypt input files. Since, the input files contain appdynamics login credentials, it is recommended to always use encrypted files and store the input files in a safe location.

## Configuring the Host

To run the application on a host, the below pre-requisites need to be met:

Pre-requisites:

Python 2.7

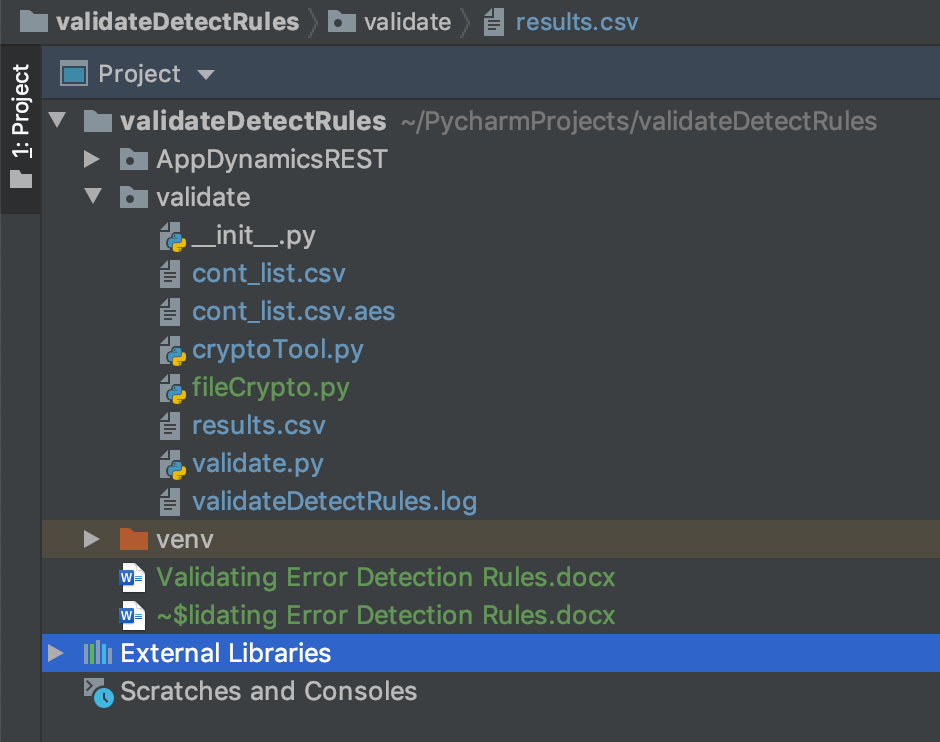
Pip

Funcy – pip install funcy

Pycryptodome – pip install pcryptodome

Other modules used in the app – os, sys, csv, json, logging, requests, base64

## Folder Structure



ValidateDetectRules – Main folder of the program

AppDynamicsREST – Folder containing AppDynamics REST SDK

Validate – Folder with python script files

cont\_list.csv – Plain text file with controller information

cont\_list.csv.aes – Encrypted file with controller information

cryptoTool.py – Script file to encrypt and decrypt a file

fileCrypto.py – Class containing the functionality for encrypting and decrypting files

results.csv – Plain text file with results

validate.py – Script file with the main logic

validateDetectRules.log – Log file with information of script execution

## Python Application

### Validate.py:

import requests  
import logging  
import logging.handlers  
import json  
import csv  
import os  
import sys  
  
from fileCrypto import FileCrypto  
from AppDynamicsREST.appd.request import AppDynamicsClient  
  
  
loginQuery = '/controller/auth?action=login'  
errorDetectionRulesQuery = '/controller/restui/applicationManagerUiBean/applicationConfiguration/'  
  
handler = logging.handlers.WatchedFileHandler(os.environ.get("LOGFILE", "./validateDetectRules.log"))  
formatter = logging.Formatter(logging.BASIC\_FORMAT)  
handler.setFormatter(formatter)  
root = logging.getLogger()  
root.setLevel(os.environ.get("LOGLEVEL", "INFO"))  
root.addHandler(handler)  
  
  
def apply\_exceptions\_match\_rules(pattern):  
 ret = False  
 matches = ['\*.\*', 'com.\*', 'myErrorMessage', '.\*']  
 match = set([])  
 for m in matches:  
 if m in pattern:  
 match.add(m)  
 ret = True  
 return ret, match  
  
  
def apply\_logger\_match\_rules(pattern):  
 ret = False  
 matches = ['\*.\*', 'com.\*', 'myErrorMessage', '.\*']  
 match = set([])  
 for m in matches:  
 if m in pattern:  
 match.add(m)  
 ret = True  
 return ret, match  
  
  
def validate\_error\_detection\_rules(controller\_url, username, password, account\_name, app):  
 *"""* ***:param*** *controller\_url: str - starts with http or https and includes port number, except for default '80' and '443'* ***:param*** *username: str - User name to authenticate to the controller with.* ***:param*** *password: str - Password for authentication to the controller.* ***:param*** *account\_name: str - Account name for multi-tenant controllers. For single-tenant controllers, use  
 the default value of "customer1".* ***:param*** *app: object - app object containing info of an app* ***:return****: none  
 """* result = 'Pass'  
 problem\_regex = ''  
 ignore\_exp\_match\_type = ''  
 ignore\_exp\_match\_pattern = ''  
 ignore\_exp\_regex\_groups = ''  
 ignore\_logger\_match\_type = ''  
 ignore\_logger\_match\_pattern = ''  
 ignore\_logger\_regex\_groups = ''  
  
 rules = get\_error\_detection\_rules(controller\_url, username, password, account\_name, app.id)  
 if rules is None:  
 return  
 if rules['errorConfig']['ignoreExceptionMsgPatterns']:  
 ignore\_exp\_match\_type = rules['errorConfig']['ignoreExceptionMsgPatterns'][0]['extendedMatchType']  
 if rules['errorConfig']['ignoreExceptionMsgPatterns']:  
 ignore\_exp\_match\_pattern = rules['errorConfig']['ignoreExceptionMsgPatterns'][0]['extendedMatchPattern']  
 if rules['errorConfig']['ignoreExceptionMsgPatterns']:  
 ignore\_exp\_regex\_groups = rules['errorConfig']['ignoreExceptionMsgPatterns'][0]['regexGroups']  
 if rules['errorConfig']['ignoreLoggerMsgPatterns']:  
 ignore\_logger\_match\_type = rules['errorConfig']['ignoreLoggerMsgPatterns'][0]['extendedMatchType']  
 if rules['errorConfig']['ignoreLoggerMsgPatterns']:  
 ignore\_logger\_match\_pattern = rules['errorConfig']['ignoreLoggerMsgPatterns'][0]['extendedMatchPattern']  
 if rules['errorConfig']['ignoreLoggerMsgPatterns']:  
 ignore\_logger\_regex\_groups = rules['errorConfig']['ignoreLoggerMsgPatterns'][0]['regexGroups']  
  
 f = open('results.csv', "a")  
 # csv.writer is used to avoid the issue of incorrect formatting, if the string contains a comma.  
 # In csv.writer, strings containing ',' are padded by ""  
 results\_writer = csv.writer(f)  
 if ignore\_exp\_match\_type == 'REGEX':  
 (result1, match1) = apply\_exceptions\_match\_rules(ignore\_exp\_match\_pattern)  
 if result1:  
 result = 'Fail'  
 problem\_regex = str(match1)  
 if ignore\_logger\_match\_type == 'REGEX':  
 (result2, match2) = apply\_logger\_match\_rules(ignore\_logger\_match\_pattern)  
 if result2:  
 result = 'Fail'  
 problem\_regex = str(match2) + " - " + problem\_regex  
 print ("%r ::: %r" % (result2, match2))  
 if ignore\_exp\_match\_type == 'REGEX' or ignore\_logger\_match\_type == 'REGEX':  
 results\_writer.writerow((result, problem\_regex, controller\_url, app.name, ignore\_exp\_match\_type,  
 ignore\_exp\_match\_pattern, ignore\_exp\_regex\_groups, ignore\_logger\_match\_type,  
 ignore\_logger\_match\_pattern, ignore\_logger\_regex\_groups))  
 f.close()  
 else:  
 f.close()  
  
  
def read\_controller\_info(row):  
 *"""* ***:param*** *row: Takes the row from the input file and populates controller fields* ***:return****: returns controller fields with values  
 """* values = []  
 count = 0  
  
 for col in row:  
 if count == 0:  
 controller\_url = col  
 # print("%10s"%col)  
 if count == 1:  
 account\_name = col  
 # print("%10s"%col)  
 if count == 2:  
 username = col  
 # print("%10s"%col)  
 if count == 3:  
 password = col  
 # print("%10s"%col)  
 if count == 4:  
 application = col  
 # print("%10s"%col)  
 count += 1  
 return controller\_url, username, password, account\_name  
  
  
def get\_appdynamics\_client(controller\_url, username, password, account\_name):  
 cli = AppDynamicsClient(controller\_url, username, password, account\_name)  
 return cli  
  
  
def generate\_controller\_api\_session(controller\_url, username, password, account\_name):  
 user = username + '@' + account\_name  
 response = requests.get(controller\_url + loginQuery, auth=(user, password))  
 # print(response.cookies)  
 for c in response.cookies:  
 if c.name == 'JSESSIONID':  
 JSESSIONID = c.value  
 # print(JSESSIONID)  
 if c.name == 'X-CSRF-TOKEN':  
 csrf\_token = c.value  
 # print(csrf\_token)  
 cookies = {  
 'JSESSIONID': JSESSIONID,  
 }  
  
 headers = {  
 'X-CSRF-TOKEN': csrf\_token,  
 }  
 return headers, cookies  
  
  
def get\_error\_detection\_rules(controller\_url, username, password, account\_name, app\_id):  
 url = '{0}{1}{2}'.format(controller\_url, errorDetectionRulesQuery, app\_id)  
 headers, cookies = generate\_controller\_api\_session(controller\_url, username, password, account\_name)  
 try:  
 res = requests.get(url, headers=headers, cookies=cookies)  
 except requests.exceptions.RequestException as e:  
 logging.error(e)  
 return None  
 rules = json.loads(res.text)  
 return rules  
  
  
def start():  
 *"""  
 This method will read the input file and begins the validation process* ***:return****:  
 """* logging.info("Starting the program...")  
 output\_filename = "results.csv"  
 controller\_list = 'controllers.csv'  
 encrypted\_controller\_list = input("Enter the encrypted file name: ")  
  
 if os.path.exists(encrypted\_controller\_list):  
 print("Reading the file ...\n Output file name is %s" % output\_filename)  
 else:  
 print("File does not exist. Trying to read the default file: 'controllers.csv.aes'")  
 if os.path.exists("controllers.csv.aes"):  
 print("Reading the default file - 'controllers.csv.aes' ... \n Output file name is %s" % output\_filename)  
 encrypted\_controller\_list = 'controllers.csv.aes'  
 else:  
 print("Default file 'controllers.csv.aes' does not exist. Aborting the program!")  
 sys.exit(0)  
 secret = input("Enter the password to decrypt the file: ")  
  
 # Writing the headers row to the output file  
 if os.path.exists(output\_filename):  
 os.remove(output\_filename)  
 results\_file = open(output\_filename, "w")  
 results\_file.write(  
 "Result, Problem Regex, Controller, Application, ignore\_exp\_match\_type, ignore\_exp\_match\_pattern, ignore\_exp\_regex\_groups, "  
 "ignore\_logger\_match\_type, ignore\_logger\_match\_pattern, ignore\_logger\_regex\_groups \n")  
 results\_file.close()  
  
 fc1 = FileCrypto(encrypted\_controller\_list, controller\_list, secret)  
 fc1.decrypt\_file()  
 print os.path.abspath(controller\_list)  
 with open(controller\_list, 'r') as f:  
 if sum(1 for line in f) < 2:  
 print("Input file - %s - has no controllers listed or file is incorrectly formatted!\n"  
 "Refer to template.csv for correct format.\n Aborting the program!")  
 sys.exit(-1)  
 else:  
 print("Reading the file...")  
 with open(controller\_list, 'r') as csvfile1:  
  
 csvreader = csv.reader(csvfile1)  
 os.remove(controller\_list)  
 next(csvreader)  
 for row in csvreader:  
 if len(row) < 4:  
 print ("incomplete controller information: %r" % row)  
 break  
 controller\_url, username, password, account\_name = read\_controller\_info(row)  
 # Using the AppDynamicsREST SDK, creating a controller object  
 cli = get\_appdynamics\_client(controller\_url, username, password, account\_name)  
 # Checking if we can login to the controller.  
 # If not, then throw an exception and go for next row  
 try:  
 # Get the list of applications in this controller  
 logging.info("Validating the controller - %s" % controller\_url)  
 cli.get\_applications()  
 for app in cli.get\_applications():  
 logging.info("App Name -- %s" % app.name)  
 validate\_error\_detection\_rules(controller\_url, username, password, account\_name, app)  
 except requests.exceptions.RequestException as e:  
 logging.error(e)  
 continue  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 start()

### fileCrypto.py:

# taken from https://tutorialsoverflow.com/how-to-make-a-file-encryption-and-decryption-tool/  
  
import sys  
import funcy  
import base64  
import Crypto.Protocol  
from Crypto import Random  
from Crypto.Cipher import AES  
  
  
def start():  
 print("Start file crypto ...")  
  
  
class FileCrypto(object):  
 *"""  
 Crypto class providing file encrypt and decrypt functions  
 """* def \_\_init\_\_(self, input\_file, output\_file, secret\_key):  
 *"""  
 Creates a new instance of the client.* ***:param*** *input\_file: str input file path.* ***:param*** *output\_file: str output file path.  
 :secret\_key: str secret key to encrypt or decrypt file  
  
 """* (self.input\_file, self.output\_file, self.secret\_key) = (input\_file, output\_file, secret\_key)  
  
 def decrypt\_file(self):  
  
 with open(self.input\_file, "rb") as encryptedFile:  
 chunk\_size = 24 \* 1024  
 encrypted = base64.b64decode(encryptedFile.read(64))  
 setup = encrypted[:48]  
 # key\_confirm = input("Please enter the key used to encrypt the file:- ")  
 salt = b'\x9aX\x10\xa6^\x1fUVu\xc0\xa2\xc8\xff\xceOV'  
 key\_check = Crypto.Protocol.KDF.PBKDF2(password=self.secret\_key, salt=salt, dkLen=32, count=10000)  
  
 def unpad(s):  
 return s[:-ord(s[len(s) - 1:])]  
  
 if key\_check == setup[:32]:  
 print("Password Correct!")  
 else:  
 print("Wrong Password!")  
 sys.exit(0)  
  
 iv = setup[32:]  
 cipher = AES.new(key\_check, AES.MODE\_CBC, iv)  
 with open(self.output\_file, "wb") as decryptedFile:  
 encrypted = base64.b64decode(encryptedFile.read())  
 chunks = list(funcy.chunks(chunk\_size, encrypted))  
 for chunk in chunks:  
 decrypted\_chunk = unpad(cipher.decrypt(chunk))  
 decryptedFile.write(decrypted\_chunk)  
  
 def encrypt\_file(self):  
  
 salt = b'\x9aX\x10\xa6^\x1fUVu\xc0\xa2\xc8\xff\xceOV'  
 key = Crypto.Protocol.KDF.PBKDF2(password=self.secret\_key, salt=salt, dkLen=32, count=10000)  
 iv = Random.new().read(AES.block\_size)  
 bs = AES.block\_size  
 chunk\_size = 64 \* 1024  
  
 def pad(s):  
 return s + (bs - len(s) % bs) \* chr(bs - len(s) % bs).encode('utf-8')  
  
 cipher = AES.new(key, AES.MODE\_CBC, iv)  
 with open(self.input\_file, "rb") as plain:  
 with open(self.output\_file, "wb") as outFile:  
 outFile.write(base64.b64encode(key + iv))  
 while True:  
 chunk = plain.read(chunk\_size)  
 if len(chunk) == 0:  
 break  
 chunk = pad(chunk)  
 outFile.write(base64.b64encode(cipher.encrypt(chunk)))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 start()

### CryptoTool.py:

from fileCrypto import FileCrypto  
  
banner = r'''  
  
Enter 'E' to Encrypt a File  
Enter 'D' to Decrypt a File  
Enter 'Q' to Quit  
'''  
print(banner)  
  
  
def quit():  
 alpha = input("Are You Sure?[yes/no] - ").lower()  
 if alpha == "yes":  
 exit()  
 if alpha == "no":  
 print(banner)  
 choice()  
  
  
def choice():  
 try:  
 selection = input("tool:- ").upper()  
 if selection == "E":  
  
 input\_file = input("Enter name of the file to encrypt:- ")  
 usr\_key = input("Please enter a key to use as your encryption key:- ")  
 output\_file = input("Enter name of the encrypted file:- ")  
  
 fc1 = FileCrypto(input\_file, output\_file, usr\_key)  
 fc1.encrypt\_file()  
  
 if selection == "D":  
  
 input\_file = input("Enter name of the file to decrypt:- ")  
 usr\_key = input("Please enter the key used to encrypt the file:- ")  
 output\_file = input("Enter name of the decrypted file:- ")  
  
 fc1 = FileCrypto(input\_file, output\_file, usr\_key)  
 fc1.decrypt\_file()  
  
 if selection == 'Q':  
 quit()  
  
 except KeyboardInterrupt:  
 print("Programme Interrupted")  
 exit()  
  
  
choice()

### Sample results.csv (output file):

Result, Problem Regex, Controller, Application, ignore\_exp\_match\_type, ignore\_exp\_match\_pattern, ignore\_exp\_regex\_groups, ignore\_logger\_match\_type, ignore\_logger\_match\_pattern, ignore\_logger\_regex\_groups   
Fail, set(['myErrorMessage']) - , http://controller4221bare-ddsrikar-ndca8gxk.srv.ravcloud.com:8090, TestAPM2, REGEX, myExceptionMessage, None, REGEX, myErrorMessage, None   
Pass, , http://controller4221bare-ddsrikar-ndca8gxk.srv.ravcloud.com:8090, TestAPM3, REGEX, myExceptionMessage, None, REGEX, \d{5}(-\d{4})?, None   
Pass, , http://controller4221bare-ddsrikar-ndca8gxk.srv.ravcloud.com:8090, TestAPM4, REGEX, /^([a-z0-9\_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$/, None, REGEX, \d{5}(-\d{4})?, None

### Sample temp.csv (input file):

controller,account,username,password,application  
http://controller4221bare-ddsrikar-ndca8gxk.srv.ravcloud.com:8090,customer1,admin,admin

### Steps to encrypt input file:

Run ‘cryptoTool.py’

Enter the following arguments:

‘E’ to encrpt a file

‘temp.csv’ as the input file name (substiture with the actual input file name)

‘pass’ as the secret to encrypt the file (substiture with the actual secret)

‘temp.csv.aes’ as the output file name (substiture with the actual output file name)

### Steps to start the program:

Run ‘validate.py’

Enter the following arguments:

‘temp.csv.aes’ as the encrypted input file (contains of list of controllers and login information)

‘pass’ for secret to decrypt the file

Results are saved in ‘results.csv’ file. Logs are writted to ‘validateDetectRules.log’.