Python as an automation tool in IS. Protecting Database Access in Python

V Y Dronov and G A Dronova
Novosibirsk State Technical University, Novosibirsk, Russia

E-mail: dronov@corp.nstu.ru

Abstract. The activity of modern organizations, regardless of the form of ownership, is highly risky. There are up to 600 risk factors in total. It is possible to reduce risk levels only after assessing them. The basis of risk assessment is analytical processing of information relating to the sphere of organization's activity. The information to be analyzed is usually accumulated in databases. This information is usually accessed with SQL queries and analyzed with application software. The most rational variant is when the business unit employees are able to select the data from the database and perform their analytics independently. The most convenient, both for sampling information from databases, and for its analysis to use python. However, python, which is an interpreted language, has a disadvantage in terms of information security - access parameters, including user login and password, database connection parameters, are located in the program body in clear form. The option proposed in this article allows you to reduce the risks of privacy violation when accessing the database.

1. Introduction
At present, the activities of all market-driven organizations, without exception, are essentially venture capital-increased risk. Managers of organizations spend up to 96% of their working time every day working against activated risks. The list of current risks can include about six hundred risk factors. One of the most important requirements on the part of organizational leaders is to identify risks and project the organization's business prospects as accurately as possible.

The main provider of data for risk assessment is analytical processing of information related to the organization's area of activity. The more extensive and in-depth information is used in analytics, the more accurate risk and outlook assessments are obtained by stakeholders. An additional critical circumstance for conducting accurate and effective risk assessments in a rapidly changing risk environment is the ability of employees to perform non-routine analytical inquiries. Employees with programming and analytical skills in their area of expertise are particularly valuable to fulfill such inquiries in organizations.

Modern analytics based on large data sets involve the need to work with databases (DB). The most important security requirement, in this case, is the requirement to provide secure access to the database. Within the framework of the client-server architecture of database management systems (DBMS) or the direct use of the SQL query language for data unloading the relative security of access to the database is provided. However, the need to conduct non-trivial, situational analytical queries leads to inexpediency of using preemptively programmed DBMS procedures. Using such a complex tool as SQL to perform non-standard queries leads to unreasonable expenditure of time, human,
computing resources spent on unloading data from the database, processing, normalizing these data and transferring them for analysis, for example, to MS Excel.

Python as a tool for automating non-standard procedures has an absolute advantage as a free development environment, characterized by ease of learning, ease of use, rich libraries, a fairly powerful analytical functionality.

Meanwhile, python has a significant disadvantage in the field of security, due to its merits. Python is a development environment, the interpreter, and, accordingly, the program code is stored in python in plain text format. Database connection parameters, such as employee login and password, database server data and connection port, database name, driver used, are in the program text in plain text.

Below are examples of python programs that provide connection to different databases.

1. Microsoft recommendation, to connect to a SQL Azure or Synapse SQL database:

   ```python
   01 import pyodbc
   02 server = '<server>.database.windows.net'
   03 database = '<database>'
   04 username = '<username>'
   05 password = '<password>'
   06 driver= '{ODBC Driver 17 for SQL Server}'
   07 with
       08 pyodbc.connect('DRIVER=' + driver + ';SERVER=' + server + ';PORT=1433;
                        DATABASE=' + database + ';UID=' + username + ';PWD=' + password) as conn:
           09   with conn.cursor() as cursor:
               ...
   ```

   Security-critical information for the database is located in lines 02, 03, 04, 05, 06, 08.

2. Access the python program to the Oracle database:

   ```python
   import cx_Oracle
   ip = 'north address'
   port = 'port of connection'
   SID = 'xxxxxxxx'
   dsn_tns = cx_Oracle.makedsn(ip, port, SID)
   conn = cx_Oracle.connect('username', 'password', dsn_tns)
   ```

3. Access the python program to one of the most popular DBMS MySQL:

   ```python
   import MySQLdb
   db = MySQLdb.connect("localhost","testuser","test123","TESTDB")
   ```

   Leakage of both the whole string (#08), used for connection, and individual combinations of components (login and password (#04, #05), server parameters (#02), database name (#03), data about the driver used (#06)), increases chances of an attacker to successfully conduct an attack against the database.

   Based on the above, the solution to the problem of protecting access to the database, carried out with the help of python, will allow to use the advantages that python has for conducting both standard and, which is extremely important, non-standard, situational analytical control procedures (controls).

2. Task statement

The work algorithm of one of the typical analytical procedures includes the following stages - access to the database, downloading of necessary data from individual fields of database tables, information processing with results output, including such services as visualization, formation of data marts, etc.
Performing these stages in manual mode is quite a labor-intensive process and using software (especially developed in python), which automates these operations, is the only right solution.

In order to connect to the database it is necessary to place sensitive data in the program code similar to the ones mentioned above, i.e. server name and connection port, employee login and password, database name, interaction driver name. At the same time there are risks of leakage of this information either through unauthorized access to the program's file which has text or other easily opened format or through accidental disclosure by an employee himself.

The need to conceal sensitive information providing access to the employee’s program to the database is an urgent IS task.

3. Theory
For the reasons given above, it is impossible to completely eliminate the risk of leakage of sensitive information when working with interpreted languages. This risk can be significantly minimized. A possible way to reduce the risk is to reduce the window of opportunity for an attack, for example, by reducing the "visibility" time of sensitive information.

Hiding an employee's username and password separately, as often recommended by IS, while leaving other potentially sensitive information open, is not an acceptable option. The name of the driver, the server address and/or the connection port may be enough for an attacker to carry out a successful attack. The preferred security solution is to protect the entire database connection string. One possible simple variant of the solution to this problem is provided below.

4. Results
Functionally, there are two actions - forming a secure string and using a secure string. To ensure the security of these actions, two separate software modules are created - one to form an encrypted access string (the preparation stage of implementation) and the second to use the encrypted string to directly access the database (the operation stage).

Preparation for Implementation
The following was required for the job:

```
# Loading libraries and variables
import getpass
import base64
import binascii
from Crypto.Cipher import AES
from Crypto.Util import Counter
from Crypto import Random

# AES supports multiple key sizes: 16 (AES128), 24 (AES192), or 32 (AES256)
# we use the simplest version AES128
key_bytes = 16

# Required functions (found on the Internet)
# Takes as input a 16-byte key and an arbitrary-length plaintext and returns a
# pair (iv, ciphertext). "iv" stands for initialization vector.
def encrypt(key, plaintext):
    assert len(key) == key_bytes
    # Choose a random, 16-byte IV.
    iv = Random.new().read(AES.block_size)
    # Convert the IV to a Python integer.
    iv_int = int(binascii.hexlify(iv), 16)
    # Create a new Counter object with IV = iv_int.
```

ctr = Counter.new(AES.block_size * 8, initial_value=iv_int)
# Create AES-CTR cipher.
aes = AES.new(key, AES.MODE_CTR, counter=ctr)
# Encrypt and return IV and ciphertext.
ciphertext = aes.encrypt(plaintext)
return (iv, ciphertext)

# Takes as input a 16-byte key, a 16-byte IV, and a ciphertext, and outputs the
# corresponding plaintext.
def decrypt(key, iv, ciphertext):
    assert len(key) == key_bytes
    # Initialize counter for decryption. iv should be the same as the output of
    # encrypt().
    iv_int = int(binascii.hexlify(iv), 16)
    ctr = Counter.new(AES.block_size * 8, initial_value=iv_int)
    # Create AES-CTR cipher.
aes = AES.new(key, AES.MODE_CTR, counter=ctr)
    # Decrypt and return the plaintext.
    plaintext = aes.decrypt(ciphertext)
    return plaintext

# topping up to 16 bytes with a character ~
def pad_text(s):
    diff = 16 - len(s) % 16
    return s + diff*'~'

# remove unnecessary ~
def unpad_text(s):
    return s.decode().rstrip('~')

String generation stage
When forming an access string, you can use either a ready-made string or the option of entering
data into separate variables, followed by assembly:

SQLline = 'mssql+pymssql://sibDomain\'
Server = input('SQL Server, please = ')
Table = input('SQL Table, please = ')
Acc = input('Account, please = ')

When entering a password, it is preferable to use the getpass library, which ensures confidentiality.
The characters entered are not displayed on the screen and are not recorded in any system logs.
Passwd = getpass.getpass('Pass to database = ')

Forming module:
# main() access to ‘OAOIC-1C’, ‘NFSVA’;
pas = getpass.getpass('Pass: ')
key = pad_text(pas)
K1 = key.encode('UTF-8')
# form an access string
S1 = pad_text(SQLline + str(Acc) + ':' + str(Passwd) + '@' + str(Server) + '/' + str(Table)).encode('UTF-8')
(iv, ciphertext) = encrypt(K1, S1)
file_save(flName.iv + ciphertext)     # write to a binary file – flName

The result is shown in Figure 1.

![Encrypted access string](image)

**Figure 1.** Encrypted access string

**Operation stage**

The second module, designed for converting the encrypted access string back into its normal form for work, is used when necessary. Its task is to get the password from the employee and decrypt the access string from the binary file. The decrypted string is stored in a string variable which is overwritten with any random data immediately after connecting to the database. As a result, there is a reduction of the attacker’s window of opportunity to a small interval of time, during which the access string is located in the string variable in the open form.

```python
# Load encrypted string from binary file
binLine = file_load(flName)
iv = binLine[0:key_bytes]
ciphertext = binLine[key_bytes:]
# entering the password to decrypt
pas = getpass.getpass('Pass: ')
key = pad_text(pas)
K2 = key.encode('UTF-8')
# line transcription
Line = decrypt(K2, iv, ciphertext) #x3, x4)
# using a string to access the database, for example, like this
pyodbc.connect(Line) as conn:
...
# overwrites the string with random data
...
```

**5. Conclusions**

The proposed solution for protecting database access required when performing analytical queries during an IS audit allows:

1) Use the advantages of using the python "small automation" tool to conduct both standard and non-trivial, ad hoc control procedures with the processing of large data sets.

2) Significantly reduce the risk of leakage of sensitive information when organizing auditor access to data sets.

3) To make changes in the code of the program developed in python to improve security measures when organizing access to large data arrays.

**References**

[1] Python3 language documentation [Electronic resource]: Python 3 language reference documentation URL: https://docs-python.ru/ (accessed 25.09.2021).

[2] Python language reference [Electronic resource]: Python.org. URL: https://www.python.org/about/help/ (accessed 25.09.2021)
[3] Connecting to SQL with pymssql [Electronic resource] : Microsoft Corp. URL : https://docs.microsoft.com/ru-ru/sql/connect/python/pymssql/step-3-proof-of-concept-connecting-to-sql-using-pymssql?view=sql-server-ver15. (accessed 25.09.2021).

[4] Connecting to Oracle Database in Python [Electronic resource]: Oracle tutorial. URL : https://www.oracletutorial.com/python-oracle/connecting-to-oracle-database-in-python/. (accessed 25.09.2021).

[5] Accessing databases in Python with pyodbc [Electronic resource] : Programming Languages. URL: https://dvsemenov.ru/dostup-k-bazam-dannyx-v-python-s-pomoshhyu-pyodbc/.(accessed 25.09.2021).