ダウンロード・データあります

Python使用時でも知財の流出を防ぐ

簡単な暗号化で実現する Pythonスクリプトの難読化

宮澤 賢児

```
#!/usr/bin/env python
def main():
    print('Hello, World!')
if __name__ == '__main__':
    main()
```

Pythonスクリプト

```
…
LOAD_GLOVAL    0 (print)
LOAD_CONST     1 ('Hello, World!')
CALL_FUNCTION 1
POP_TOP
LOAD_CONST     0 (None)
RETURN_VALUE
…
```

コンパイル後

```
…
??????    ???
??????    ???
??????    ???
??????    ???
??????    ???
??????    ???
…
```

難読化されたため，人間には読めない

難読化後

図1　今回行う難読化

機械学習をはじめとして，Pythonは多くの業務や商用製品に活用されています．

Pythonスクリプトをユーザの実行環境に配布するとき，スクリプトの内容を知られたくないと思うことはないでしょうか．特に，独自に作成したアルゴリズム，生産性向上のための共通関数，プログラミング・テクニックなどはスクリプト実装者の財産であるため，できれば知られたくないものです．

そこで，簡単な暗号化を用いてスクリプトの処理内容を容易には知られないようにしつつ，スクリプト自体はこれまで通り実行できる方法について解説します(**図1**)．

難読化の必要性

● 難読化とは

ここでは，「動作には影響がなく意図的に処理内容を分かりにくくしたスクリプト」を難読化と定義します．そのため，難読化したスクリプトは，リバース・エンジニアリングやデバッガなどを用いることで，労力や時間をかければ処理内容を理解できるものです．

難読化にもレベルがありますが，特に商用のスクリプトはリバース・エンジニアリングをあきらめるレベルの難読化が求められることがあります．

また，鍵を知っている人だけが復号できる内容を暗号と定義します．暗号化によって難読化を実現します．難読化したものを復号して実行するしくみも必要になるので，これも実装します．

● Pythonスクリプトの問題点…バイト・コードの配布だけでは解決できない

Pythonスクリプトは人間が読めるテキスト形式なので，理解も容易です．一方で，動作させたい環境(実行環境)へスクリプトを配布する必要があり，処理内容を知られることで知的財産を守るのが難しいという大きな課題があります．

なお，Pythonスクリプトをコンパイルするとバイト・コードというPythonインタープリタで実行するバイナリの命令コードに変換されます．バイト・コードは人間が直接は読めない形式で，.pycまたは.pyoの拡張子が付いたファイルです．バイト・コードを実行環境へ配布すれば元のスクリプトと同じ処理内容で実行できるので，一見これで良さそうに感じます．しかし，このバイト・コードは逆アセンブルなどリバース・エンジニアリングによりスクリプトの処理内容を容易に知ることができてしまいます．

今回のアプローチ…バイト・コードを難読化する

そこで，このバイト・コードを難読化し，リバース・エンジニアリングを難しくします．

● バイト・コードを暗号化する

バイト・コードを暗号化すれば，解読を試みても直接は理解できず，リバース・エンジニアリングも困難なため，処理内容を解読したいと思う人が諦めるレベルの難読化が実現できます．