Міністерство освіти і науки України Національний технічний університет «ХПІ»

Навчально-науковий інститут комп'ютерних наук та інформаційних технологій

Кафедра комп'ютерної інженерії та програмування

3BIT

з лабораторної роботи № 12 з дисципліни «Сучасні технології безпечного програмування»

«ДОСЛІДЖЕННЯ ЗАСОБІВ ОБФУСКАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ»

Виконав:

студент гр. КН-Н922б

Кулик Д.І.

Перевірив:

Бульба С. С.

Мета роботи: Дослідити існуючі утиліти обфускації програмного забезпечення.

Індивідуальне завдання

Для обраної мови програмування слід обфускувати будь-який проект.

Слід використовувати декілька утиліт для обфускації. Мінімальна кількість обфускаторів - 4. У звіті навести порівняльну характеристику обраних обфускаторів, що складатиметься з:

- інформації щодо використовуваних методів обфускації
- плюсів та мінусів кожного з обфускаторів.

Навести опис використання кожного з обраних обфускаторів.

Хід роботи

Обфускація в широкому значенні - приведення вихідного тексту або виконуваного коду програми до виду, що зберігає її функціональність, але утруднює аналіз, розуміння алгоритмів роботи та модифікацію при декомпіляції.

Більшість методів обфускації перетворюють такі аспекти коду:

- Дані: роблять елементи коду схожими на те, чим вони не ϵ
- Потік коду: виставляють логіку програми абсурдної або навіть недетермінованої.
- Структура формату: застосовують різне форматування даних, перейменування ідентифікаторів, видалення коментарів коду тощо.

Інструменти обфускації можуть працювати як із source або байт кодом, так і з бінарним, проте обфускація двійкових файлів складніша, і повинна змінюватись в залежності від архітектури системи.

При обфускації коду важливо правильно оцінити, які частини коли можна ефективно заплутати. Слід уникати обфускації коду критичного щодо продуктивності.

```
|def generate_hash(message: bytearray) -> bytearray:
    """Повертає хеш SHA-256 із переданого повідомлення.
    Аргумент має бути об'єктом bytes, bytearray або string."""
    if isinstance(message, str):
       message = bytearray(message, 'ascii')
    elif isinstance(message, bytes):
       message = bytearray(message)
    elif not isinstance(message, bytearray):
       raise TypeError
    # Заповнювання
    length = len(message) * 8 # len(message) - це кількість байтів
    message.append(0x80)
    # перетворене повідомлення заповнити нулями доки дані не стануть кратними 512 без останніх 64 біт
    while (len(message) * 8 + 64) % 512 != 0:
       message.append(0x00)
    message += length.to_bytes(8, 'big') # доповнення до 8 байтів або 64 бітів
    assert (len(message) * 8) % 512 == 0, "Padding did not complete properly!"
```

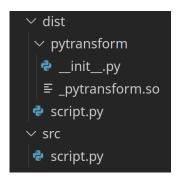
Рисунок 1 – Початковий код (лабораторна робота 01)

Проведемо обфускацію за допомогою утілити руагтог

Заплутувати сценарій Python за допомогою PyArmor так само просто, як виконати таку команду в консолі:

pyarmor obfuscate script.py

Це створить папку dist, яка містить обфускований файл Python з такою самою назвою, як і вихідний, разом із залежностями середовища виконання, як показано в структурі папок нижче:



```
(stbp) C:\Users\Daniil\PvcharmProjects\stbp>pvarmor obfuscate LABS\kulvk12\main.pv
INFO
        PyArmor Trial Version 7.7.3
INFO
        Python 3.8.6
       Target platforms: Native
       Source path is "C:\Users\Daniil\PycharmProjects\stbp\LABS\kulyk12"
INFO
INFO
        Entry scripts are ['LABS\\kulyk12\\main.py']
INFO
        Use cached capsule C:\Users\Daniil\.pyarmor\.pyarmor_capsule.zip
INFO
       Search scripts mode: Normal
INFO Save obfuscated scripts to "dist"
TNFO
       Read product key from capsule
INFO
        Obfuscate module mode is 2
INFO
       Obfuscate code mode is 1
INFO
      Obfuscate string value is False
INFO Wrap mode is 1
INFO
       Restrict mode is 1
INFO
        Advanced value is 0
INFO
       Super mode is False
INFO Super plus mode is not enabled
INFO
       Generating runtime files to dist\pytransform
INFO
        Extract pytransform.key
INFO
        Generate default license file
INFO
       Update capsule to add default license file
INFO
       Copying c:\users\daniil\pycharmprojects\stbp\lib\site-packages\pyarmor\platforms\windows\x86_64\_pytransform.dll
TNFO
       Patch library dist\pytransform\_pytransform.dll
INFO
        Patch library file OK
       Copying c:\users\daniil\pycharmprojects\stbp\lib\site-packages\pyarmor\pytransform.py
INFO
INFO Rename it to pytransform/__init__.py
INFO Generate runtime files OK
INFO
       Start obfuscating the scripts...
INFO
               main.py -> dist\main.py
INFO Patch this entry script with protection code
INFO Insert bootstrap code to entry script dist\main.py
INFO Obfuscate 1 scripts OK.
(stbp) C:\Users\Daniil\PycharmProjects\stbp>
```

Рисунок 2 – Виконання команди для обфускації

Рисунок 3 – Обфускація коду за допомогою руагтог

Спробуємо запустити обфускований файл

```
(stbp) C:\Users\Daniil\PycharmProjects\stbp\LABS>python dist\main_pyarmor.py
P'PIPµPrPµPSPëP№ C,PµPєC CC; Daniil Kulyk
P PsP·C为PsP±P»PµPSPëP№ SHA256 = 88d44f1bf7c5886b9f0eb28963037d7d9fff9e99a7c24a869124fede16bc46b8
Hashlib SHA256 = 88d44f1bf7c5886b9f0eb28963037d7d9fff9e99a7c24a869124fede16bc46b8
```

Рисунок 4 – Успішний запуск обфускованого файлу

Утиліта ору заплутає наш вихідний код Python, але за допомогою іншого алгоритму.

Відомі обмеження:

• Перед коментарем після рядкового літералу має стояти пробіл. «або» всередині рядкового літералу слід екранувати \, а не подвоювати.

• Обфускація рядкових літералів непридатна для конфіденційної інформації, оскільки її можна тривіально зламати.

```
(stbp) C:\Users\Daniil\PycharmProjects\stbp\LABS\kulyk12>python opy.py
Opy (TM) Configurable Multi Module Python Obfuscator Version 1.1.28
Copyright (C) Geatec Engineering. License: Apache 2.0 at <a href="http://www.apache.org/licenses/LICENSE-2.0">http://www.apache.org/licenses/LICENSE-2.0</a>
Obfuscated words: 180

(stbp) C:\Users\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LABS\kulyk12>\Daniil\PycharmProjects\stbp\LAB
```

(stbp) C:\Users\Daniil\PycharmProjects\stbp\LABS\kulyk12>

Рисунок 5 – Виконання команди для обфускації

```
.py × 🚪 opy_config.txt × 🚜 l1l_opy_.py ×
 # coding: UTF-8
 lall11_opy_ = sys.version_info [0] == 2
l1lll1_opy_ = 2048
  l1llll1_opy_ = 7
  def l1l1lll_opy_ (l1lll1l_opy_):
      global lillil_opy_
lillil_opy_ = ord_(lillil_opy_ [-1])
lilli_opy_ = lillil_opy_ [:-1]
      l1lll11_opy_ = l1l111_opy_ % len_(l1l11_opy_)
      l1ll_opy_ = l1l11_opy__[:l1lll11_opy_] + l1l11_opy__[l1lll11_opy_:]
           l11111_opy_ = ll_opy_ () .join_([l11ll_opy_ (ord_(char) - l1lll1_opy_ - (l11_opy_ + l1ll11_opy_) % l1llll1_opy_) for l11_opy_, char in enumerate_(l1ll_opy_)])
      else:
           lilili_opy_ = str_() .join_([chr_(ord_(char) - lillil_opy_ - (lil_opy_ + lillil_opy_) % lilllil_opy_) for lil_opy_, char in enumerate (lill_opy_)])
      return eval (l11111_opy_)
  import hashlib
  111111_opy_ = [
      0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
       0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
      0x4905901, 0xefbe4786, 0x96fc190c6, 0x240ca1cc, 0x2de92c6f, 0x4a7484a, 0x5cb0a9dc, 0x76f988da, 0x985e5152, 0xa831c66d, 0xb0037c8, 0xbf597fc7, 0xc6e00bf3, 0x63e79147, 0x60ca351, 0x14292967, 0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x55380d13, 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85, 0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
      0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3, 0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2
 def l11ll1_opy_(message: bytearray) -> bytearray:
      l1l1lll_opy_ (u"0000
                                                              if isinstance(message, str):
           message = bytearray(message, l1l1lll_opy__(u"00000000"))
      elif isinstance(message, bytes)
```

Рисунок 6 – Обфускація коду за допомогою ору

```
(stbp) C:\Users\Daniil\PycharmProjects\stbp>python LABS\kulyk12_opy\l1l1l_opy_.py
Введений текст: Daniil Kulyk
Розроблений SHA256 = 88d44f1bf7c5886b9f0eb28963037d7d9fff9e99a7c24a869124fede16bc46b8
Hashlib SHA256 = 88d44f1bf7c5886b9f0eb28963037d7d9fff9e99a7c24a869124fede16bc46b8
```

Рисунок 7 – Успішний запуск обфускованого файлу

Висновки: в результаті виконання лабораторної роботи було досліджено існуючі утиліти обфускації програмного забезпечення.