Защита приложений написанных на языке Python

Оглавление

1. Общее

Существует два возможных варианта защиты Python приложений:

- Защита через шифрование данных (Sentinel Envelope с использованием Sentinel Data File Protection (DFP))
- Экспорт модулей через Cython(https://cython.org/) с последующей защитой через Sentinel Envelope

Первый вариант защищает Python приложения, комбинируя Sentinel Envelope с Sentinel Data File Protection (DFP). Таким образом, защищается непосредственно интерпретатор Python, которому на уровне защиты указывается с какими файлами он должен работать как с защищенными (зашифрованными), следовательно, такие файлы интерпретатор будет предварительно автоматически расшифровывать и только затем выполнять. Незашифрованные файлы будут работать также, как и ранее. Сами *.py модули, требующие защиты шифруются с помощью инструмента Sentinel Data File Protection (DFP).

Второй вариант подразумевает защиту Python приложений, комбинируя Cython (https://cython.org/) с Sentinel Envelope. Сначала необходимо требующие защиты модули Python перевести в C-Code с помощью Cython и затем скомпилировать их в модули *.pyd / *.so, которые в последствие защищаются с помощью Sentinel Envelope.

Второй метод обеспечивает более высокий уровень безопасности, поскольку дополнительный этап компиляции снижает уровень абстракции кода и позволяет Sentinel Envelope защищать приложение как код, а не только как данные, что позволяет применять более сложные механизмы защиты. С другой стороны, второй метод немного сложнее в настройке, так как требует дополнительно “прослойки” в виде Cython и работающий C-компилятор.

Примечание. Оба метода не могут защитить стартовый скрипт приложения, а только его модули Python. Поэтому рекомендуется поместить фактическую точку входа вашего приложения в модуль Python и использовать только стартовый скрипт для вызова модуля.

2. Метод #1: защита через шифрование данных (Sentinel Data File Protection)

Защита приложения Python с помощью Sentinel Data File Protection состоит из трех этапов:

- Соберите ваше приложение в модули байт-кода *.pyc.
- Зашифруйте получившиеся файлы *.pyc с помощью утилиты Sentinel dfcrypt.

- Защитите интерпретатор Python, включив в настройках защиты опцию “Enable data file protection (Data Protection Utility) = Version 2” для работы защищённого интерпретатора с зашифрованными файлами данных.

Защищенное приложение может распространяться путем упаковки защищенного с помощью Envelope интерпретатора вместе с зашифрованными *.pyc файлами (например, с использованием pyinstaller).

Примечание: Важно всегда защищать скомпилированный байт-код Python (*.pyc), а не простой исходный код Python (.py). Причина в том, что интерпретатор Python сначала переводит файлы *.py в файлы *.pyc, которые затем записываются на диск, чтобы ускорить последующее выполнение. При предоставлении защищенного файла *.py вместо файла *.pyc интерпретатор Python сгенерирует открытый текстовый файл *.pyc и запишет его на диск, где он будет доступен для анализа в открытом виде**.**

2.1. Примеры защиты

2.1.1. Для Windows

После установки Sentinel LDK пример, демонстрирующий защиту Python приложения для Windows с помощью Sentinel Data File Protection, можно найти в директории:

```
C:\Users\<username>\Documents\Gemalto\Sentinel LDK
   version\Samples\Envelope\Python \data_file_protection
```

Прилагаемые скрипты демонстрируют защиту простого Python приложения, которое можно найти в директории:

```
C:\Users\<username>\Documents\Gemalto\Sentinel LDK version\Samples\Envelope\Python
   \sample_app
```

Пример представляет собой простое Python приложение командной строки, которое состоит из сценария запуска (main.py) и 3’х модулей (moduleA.py, moduleB.py, moduleC.py). Этот пример был написан так, чтобы работать одинаково в обеих версиях Python2 и Python3.

Директория `data_file_protection` содержит:

- `build_dfp_protected_python2_app.bat`
  Этот сценарий защищает и упаковывает пример приложения для Python2 дляWindows, с использованием: кода разработчика DEMOMA, Sentinel Envelope, утилиты шифрования файлов данных для Sentinel LDK (dfcrypt) и pyinstaller.
- `envelope_python2.prjx`
  Файл проекта Sentinel Envelope, в котором указаны параметры Envelope для защиты интерпретатора Python2 (python27.dll) для приведенного выше сценария.
- `build_dfp_protected_python3_app.bat`
  Этот сценарий защищает и упаковывает пример приложения для Python3 для Windows, с использованием: кода разработчика DEMOMA, Sentinel Envelope, утилиты шифрования файлов данных для Sentinel LDK (dfcrypt) и pyinstaller.

- `envelope_python3.prjx`
  Файл проекта Sentinel Envelope, в котором указаны параметры Envelope для защиты интерпретатора Python3 (python37.dll) для приведенного выше сценария.

2.1.2. Для Linux

Пример, демонстрирующий защиту Python приложения для Linux с использованием Sentinel Data File Protection, можно найти в директории:

```
<DVD_Root>/Linux/Samples/Envelope/Python/data_file_protection
```

Прилагаемые скрипты демонстрируют защиту простого Python приложения, которое можно найти в директории:

```
<DVD_Root>/Linux/Samples/Envelope/Python/sample_app
```

Пример представляет собой простое Python приложение командной строки, которое состоит из сценария запуска (main.py) и 3'х модулей (moduleA.py, moduleB.py, moduleC.py). Этот пример был написан так, чтобы работать одинаково в обеих версиях Python2 и Python3.

Директория `data_file_protection` содержит:

- `build_dfp_protected_python2_app.sh`
  Этот сценарий защищает и упаковывает пример приложения для Python2 для Linux с использованием: кода разработчика DEMOMA, Sentinel Linux Envelope, утилиты шифрования файлов данных для Sentinel LDK (dfcrypt) и pyinstaller.
- `build_dfp_protected_python3_app.sh`
  Этот сценарий защищает и упаковывает пример приложения для Python3 для Linux с использованием: кода разработчика DEMOMA, Sentinel Linux Envelope, утилиты шифрования файлов данных для Sentinel LDK (dfcrypt) и pyinstaller.

2.2. Процесс защиты

Шаги для создания защищенного приложения с использованием Sentinel Data File Protection и Sentinel Envelope:

2.2.1. Сборка приложения и зависимостей в *.py модуль

Запустите pyinstaller с параметрами: "-d noarchive" для требующего защиты*.py модуля, например:

```
pyinstaller -d noarchive main.py
```

Этот шаг собирает зависимости вашего приложения и компилирует их в байт-код Python. «-d noarchive» инструктирует pyinstaller хранить скомпилированные модули Python как отдельные файлы, что является обязательным требованием для следующего шага шифрования.

Результатом этого шага является автономная папка (dist/<application_name>), содержащая ваше приложение и все его зависимости.

Примечание: Если установленный pyinstaller не поддерживает ключ "-d noarchive", он слишком стар и его необходимо обновить с помощью команды pip:

```
pip install pyinstaller --upgrade
```

2.2.2. Шифрование *.py модулей

Используйте dfcrypt для шифрования чувствительных модулей байт-кода вашего приложения, например:

```
dfcrypt --encrypt --encver:2 "--key:A secret" --vcf:DEMOMA.hvc --fid:0
dist/<application_name>/moduleA.pyc encrypted/moduleA.pyc

dfcrypt --encrypt --encver:2 "--key:a secret" --vcf:DEMOMA.hvc --fid:0
dist/<application_name>/moduleB.pyc encrypted/moduleB.pyc
```

Хотя это и не является строго обязательным, рекомендуется использовать ключ «--key:» для указания общего ключа шифрования для всех модулей приложения. Это позволяет использовать логику кэширования, которая может существенно улучшить время запуска приложения, использующего большое количество защищенных модулей. Не указывая "--key:", команда dfcrypt выбирает случайный ключ для каждого файла.

Когда закончите с шифрованием, замените исходные файлы их зашифрованным аналогом, например:

```
copy encrypted/moduleA.pyc dist/<application_name>/

copy encrypted/moduleB.pyc dist/<application_name>/
```

2.2.3. Защита интерпретатора Python

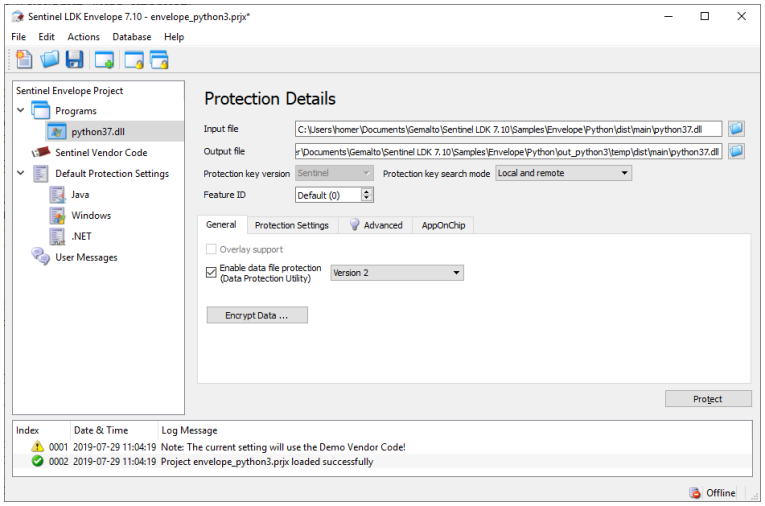
Защитите с помощью Envelope библиотеку интерпретатора Python (*.dll / *.so) с поддержкой возможности чтения зашифрованных файлов данных:

Например в Linux и Python2:

```
linuxenv --vcf:DEMOMA.hvc --fid:0 --dfp dist/<app_name>/libpython2.7.so.1.0
dist/<app_name>/libpython2.7.so.1.0
```

Ключ «--dfp» активирует Data File Protection, что позволяет защищенному интерпретатору Python выполнять зашифрованные модули Python.

В Windows используйте Sentinel Envelope и активируйте Data File Protection Version 2, см. скриншот ниже:



Включение Data File Protection Version 2

Скопируйте выходные файлы Envelope в дтректорию: `dist/<application_name>`.

Чтобы развернуть защищенное приложение, просто скопируйте папку `dist/<application_name>` на целевой компьютер.

Ваше защищенное приложение можно запустить, выполнив команду вида: `dist/<application_name>/<application_name>`.

3. Метод #2: экспорт модулей через Cython и защита с помощью Sentinel Envelope

Защита приложения Python с помощью Cython и Sentinel Envelope состоит из трех этапов:

1. Переведите ваши модули Python (*.py) в C-Code, используя Cython.
2. Скомпилируйте полученные C-файлы в модули расширения Python (*.pyd / *.so), используя C-компилятор для вашей платформы[1].
3. Защитите полученные модули расширения Python (*.pyd / *.so) с помощью Sentinel Envelope.

Защищенное приложение можно распространять, предоставляя защищенные модули расширения Python вместе со стартовым скриптом вашего приложения (например: pyinstaller).

3.1. Примеры защиты

3.1.1. Для Windows

После установки Sentinel LDKпример, демонстрирующий защиту приложения Python для Windows с использованием Cython и Sentinel Envelope, можно найти в директории:

```
C:\Users\<username>\Documents\Gemalto\Sentinel LDK
   version\Samples\Envelope\Python \cythonize_and_envelope
```

Скрипты демонстрирующие защиту простого приложения Python, которое можно найти в директории:

```
C:\Users\<username>\Documents\Gemalto\Sentinel LDK version\Samples\Envelope\Python
   \sample_app
```

Пример приложения представляет собой простое Python приложение командной строки, которое состоит из сценария запуска (main.py) и 3’х модулей (moduleA.py, moduleB.py, moduleC.py). Этот пример был написан так, чтобы работать одинаково в обеих версиях Python2 и Python3.

Директория `cythonize_and_envelope` содержит:

- `build_python2.bat` Этот сценарий защищает и упаковывает пример приложения для Python2 под Windows, с использованием: кода разработчика DEMOMA, Cython, компилятора Microsoft Visual C ++ для Python 2.7, Sentinel Envelope и pyinstaller.

- `envelope_cythonized_py2modules.prjx`
  Файл проекта Sentinel Envelope, который задает параметры Envelope для защиты расширений Python (moduleA.pyd, moduleB.pyd), которые были созданы с помощью приведенного выше сценария.
- `build_python3.bat` Этот сценарий защищает и упаковывает пример приложения для Python3 под Windows, с использованием: кода разработчика DEMOMA, Cython, Microsoft Build Tools для Visual Studio 2019, Sentinel Envelope и pyinstaller.
- `envelope_cythonized_py3modules.prjx`
  Файл проекта Sentinel Envelope, который задает параметры Envelope для защиты расширений Python (moduleA.pyd, moduleB.pyd), которые были созданы с помощью приведенного выше сценария.

3.1.2. Для Linux

Пример защиты приложения Python для Linux с использованием Cython и Sentinel Envelope можно найти в директории:

```
<DVD_Root>/Linux/Samples/Envelope/Python/cythonize_and_envelope
```

Прилагаемые скрипты демонстрируют защиту простого приложения Python, которое можно найти в директории:

```
<DVD_Root>/Linux/Samples/Envelope/Python/sample_app
```

Пример представляет собой простое Python приложение командной строки, которое состоит из сценария запуска (main.py) и 3’х модулей (moduleA.py, moduleB.py, moduleC.py). Этот пример был написан так, чтобы работать одинаково в обеих версиях Python2 и Python3.

Директория `cythonize_and_envelope` содержит:

- `build_python2.sh`
  Этот сценарий защищает и упаковывает приложение для Python2 под Linux с использованием: кода разработчика DEMOMA, Cython, GCC, Sentinel Envelope и pyinstaller.
- `build_python3.sh`
  Этот сценарий защищает и упаковывает приложение для Python3 под Linux с использованием: кода разработчика DEMOMA, Cython, GCC, Sentinel Envelope и pyinstaller.

3.2. Процесс защиты

Шаги для создания защищенного приложения с использованием Cython и Sentinel Envelope:

3.2.1. Трансляция *.py модулей в C-Code с использованием Cython

Для защиты необходимо сначала изменить расширение вашего модуля Python с *.py на *.pyx, потому что это позволяет Cython генерировать код, который может быть лучше защищен с помощью Sentinel Envelope.

Cython может быть установлен с использованием команды pip:

```
pip install cython --upgrade
```

Запустите cython и укажите, должен ли он обрабатывать код как Python 2 или Python 3:

```
cython -2 --no-docstrings .\moduleA.pyx
```
или

```
cython -3 --no-docstrings .\moduleA.pyx
```

Результатом является представление модуля Python в виде C-кода (`moduleA.c`).

3.2.2. Сборка C-Файлов в нативное Python приложение (исполняемый файл)

Для Windows:

Установите требуемый компилятор:

- Python2: Microsoft Visual C++ Compiler для Python 2.7 https://www.microsoft.com/download/details.aspx?id=44266
- Python3: Microsoft Build Tools для Visual Studio 2019 https://www.visualstudio.com/downloads/#build-tools-for-visual-studio-2019

Запустите консоль (cmd-shell) и выполните команду:

- Python2: `C:\Users\<username>\AppData\Local\Programs\Common\Microsoft\Visual C++ for Python\9.0\vcvarsall.bat" amd64`
- Python3: `C:\<vs_build_tools_install_dir>\VC\Auxiliary\Build\vcvarsall.bat" amd64`

Выполните следующие команды в консоли, чтобы скомпилировать C-код:

- Python2: `cl /nologo /c /MD /Ox /W3 /I C:\Python27\include moduleA.c link /nologo /dll -out:moduleA.pyd C:\Python27\libs\python27.lib moduleA.obj`
- Python3: `cl /nologo /c /MD /Ox /W3 /IC:\Python37\include moduleA.c link /nologo /dll -out:moduleA.pyd C:\Python37\libs\python37.lib moduleA.obj`

В результате получите модуль расширения Python для Windows (moduleA.pyd).

Для Linux:

Установите GCC-Compiler с помощью менеджера пакетов вашего дистрибутива Linux, например следующей командой в консоли:

```
apt install gcc
```

Установите пакет разработчика Python, используя менеджер пакетов вашего дистрибутива Linux, например:

```
apt install python2-dev
```

или

```
apt install python3-dev
```

Выполните следующие команды, чтобы скомпилировать C-код:

- Python2: `gcc -fPIC -pthread -fwrapv -O2 -Wall -fno-strict-aliasing -I/usr/include/python2.7 -c moduleA.c gcc -pthread -shared moduleA.o -o moduleA.so`
- Python3: `gcc -fPIC -pthread -fwrapv -O2 -Wall -fno-strict-aliasing -I "/usr/include/python3.7" -c moduleA.c gcc -pthread -shared moduleA.o -o moduleA.so`

В результате получите модуль расширения Python для Linux (moduleA.so).

3.2.3. Защита нативного Python приложения (исполняемого файла) с помощью утилиты Envelope

Пример команды для защиты для Linux:

```
linuxenv --vcf:DEMOMA.hvc --fid:0 plain/moduleA.so prot/moduleA.so
```

Для Windows используйте Sentinel Envelope точно так же, как при защитите обычного dll/exe файла.