Запускаем **Мой компьютер**, переходим в папку **Домашняя/apps**.

В этой папке находится *тестовый скрипт на питоне* (он *имитирует* работу, например, какого-либо системного скрипта), файлы с его hash-суммой и электронной подписью.

Открываем терминал **Сервис->Запустить терминал**

Запускаем тестовый скрипт:

**python2 test.py**

он выводит *Hello world*.

**Закрываем терминал**

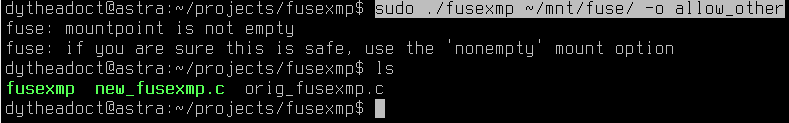
Используя менеджер файлов, переходим в папку

**projects/fusexmp**

в которой находится исполняемый файл виртуальной файловой системы.

Открываем терминал **Сервис’->’Запустить терминал** и монтируем виртуальную файловую систему в папку ~/mnt/fuse/

**sudo ./fusexmp ~/mnt/fuse/ -o allow\_other**



Переходим в папку ~/mnt/

**cd ~/mnt**

Выполняем команду

**sudo su**

для получения root прав и переходим в папку, где находится ***измененный*** скриптовый файл test.py

**cd fuse/home/dytheadoct/apps**

Просматриваем содержание этой папки:

**ls**

Как видно, в этой же папке находятся файлы с хеш-суммой и цифровой подписью от оригинального скрипта – они будут использоваться для проверки подлинности файла test.py.

Запускаем "вредоносный" скрипт

**python2 test.py**

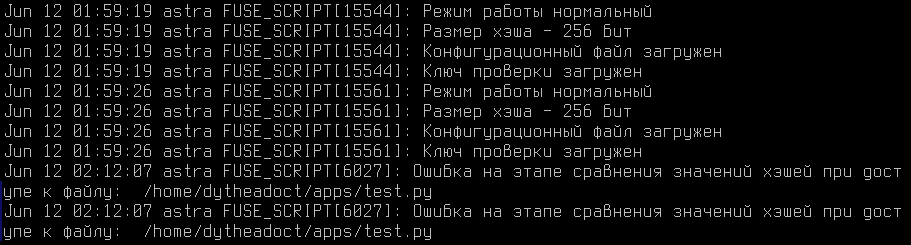
и получаем ошибку.

Чтобы увидеть причину, переходим в соответствующую папку

**cd /var/log**

и выводим содержимое файла syslog:

**cat syslog | grep FUSE\_SCRIPT**



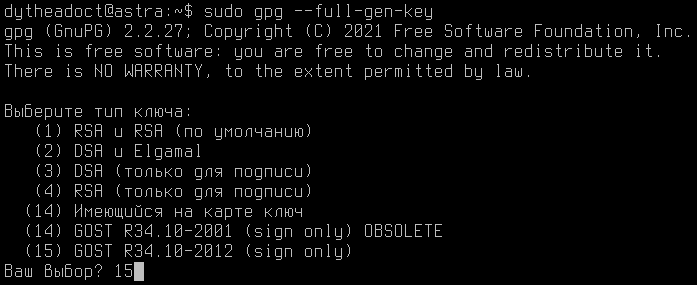
Видно, что fusexmp не позволил запустить скрипт из-за разницы в hash-суммах файлов.

**Таким образом, внутри виртуальной файловой системы, созданной с помощью fusexmp, нельзя запустить изменённые и скриптовые файлы.**

**Мануальная часть**

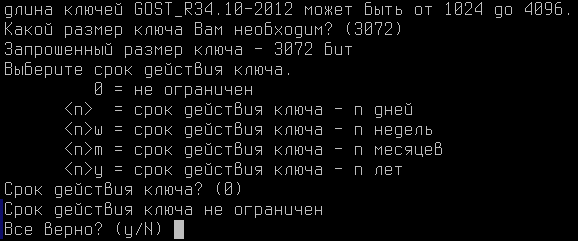
1) В первую очередь необходимо открыть терминал. В терминале ввести команду для генерации ключа подписи, который будет использован для подписания файла с эталонным значением контрольной суммы:

‘sudo gpg --full-gen-key’



2) В списке выбора типа ключа необходимо выбрать вариант (15) GOST R34.10-2012 (sign only), для чего ввести «15» в терминале.

3) В меню выбора срока действия ключа поставить необходимое значение, либо оставить по умолчанию (0), что будет означать неограни-ченный срок действия ключа. После этого будет выведена дата окончания срока действия ключа, либо написано, что срок действия ключа не ограни-чен. Если данные верны, вводите «y», в противном случае вводите «N» и меняете значение срока действия ключа.



4) В следующем пункте создания ключа необходимо ввести ID пользователя. Утилита создаст его из Вашего имени, комментария и адреса электронной почты, которые вы введете. Например: «Ivanov (Ivan Iva-novich) <ivanov@mail.ru>». Соответственно сначала будет предложено ввести Ваше настоящее имя, затем адрес электронной почти и в конце комментарий. После этого на экран будет выведена строка наподобие той, которая была в примере, только с введенными Вами данными. Проверяете правильность ввода. Если какой-то из пунктов введен неверно, его можно изменить, введя соответствующие буквы латинского алфавита: «N» – имя, «C» – комментарий, «E» – адрес электронной почты. Если же все введено верно, то подтверждаете свой выбор, введя «О». Если вы ходите выйти из программы создания ключа, введите «Q».

5) Следующий пункт – создание пароля для защиты закрытого ключа. Именно этот пароль будет использоваться для расшифрования данных закрытого ключа, которые впоследствии будут записаны в файл. После ввода пароля Вам будет предложено подтвердить пароль, следова-тельно, пароль необходимо ввести повторно.

6) Следующим шагом программа начнет генерацию случайных чисел для получения достаточного количества энтропии для дальнейшего создания Вашего ключа. Этот процесс довольно длительный. Чтобы уско-рить его, программа предлагает Вам выполнять некоторые действия: пе-чать на клавиатуре, движение мыши, обращение к дискам.

7) После завершения процесса генерации случайных чисел, про-грамма выдаст итоговую информацию по созданию ключа. Если данный ключ первый, который был создан в системе, то программа создаст табли-цу доверия, и внесет в нее этот ключ как абсолютно доверенный. Далее программа напишет, что открытый и закрытый ключи созданы и подписа-ны. Теперь их можно будет найти в директории: /root/.gnupg/, где откры-тый ключ будет содержаться в файле «pubring.gpg», а закрытый – в файле «secring.gpg». После этого программа выведет информацию о том, что в таблице доверия один подписанный ключ и выведет его идентификатор, дату создания, отпечаток и ID пользователя, который создал этот ключ.

8) Следующим шагом будет расчет контрольной суммы исполь-зуемого скрипта с помощью утилиты «gostsum». Чтобы рассчитать кон-трольную сумму данной утилитой и сохранить значение в файл, необхо-димо ввести команду:

gostsum \*название файла\* > \*название выходного файла\*

9) После создания файла с рассчитанным хешем скрипта его необходимо подписать созданным ключом. Сделать это можно с помощью команды:

gpg --sign \*имя файла с хешем\*

В случае, если в системе несколько ключей для подписания и тот ключ, которым Вы собираетесь подписать скрипт не является ключом по умолчанию для подписывания, то необходимо задать ID пользователя, ключом которого Вы хотите подписать скрипт:

gpg -u \*ID пользователя\* --sign \*имя файла с хешем\*

Утилита потребует пароль для доступа к закрытому ключу пользо-вателя, создавшего данный ключ. Необходимо ввести пароль, который был задан на этапе создания ключа. Если пароль введен верно, то файл будет подписан, в противном случае утилита сообщит, что введен невер-ный пароль и предложит попробовать ввести пароль еще раз.

После этого появится подписанный файл в формате \*имя файла с хешем\*.gpg, который будет содержать данные самого файла с хешем в за-шифрованном виде.

Разработчик должен предоставить пользователю вместе с созданным приложением, написанным на языке сценариев, свой открытый ключ для проверки валидности подписи. Чтобы экспортировать ключ, необходимо воспользоваться командой:

gpg --export --output \*выходной файл\* \*ID ключа\*

На этом первая часть методики заканчивается.

Следующий этап – настройка виртуальной файловой системы и за-пуск через нее скрипта разработчика с проверкой валидности подписи разработчика и сравнения высчитанного хеша приложения с «эталонным». На машине пользователя необходимо установить соответствующие биб-лиотеки для работы с виртуальной файловой системой, подсчитывания контрольной суммы и верификации подписи разработчика.

1) В первую очередь администратор должен импортировать от-крытый ключ разработчика в свою базу ключей. Для этого необходимо воспользоваться командой:

gpg --import \*имя открытого ключа разработчика\*

2) Чтобы расшифровать подписанный файл и сохранить расшиф-рованные данные в выходной файл, необходимо воспользоваться коман-дой:

gpg --decrypt \*имя подписанного файла\* > \*имя выходного файла\*

Утилита «gpg» сама подберет необходимый открытый ключ для расшифрования, расшифрует и поместит расшифрованные данные в вы-ходной файл.

3) Следующий шаг – вычисление контрольной суммы полученно-го файла со скриптом. Сделать это можно по уже известной из первой ча-сти методики команды:

gostsum \*название файла\* > \*название выходного файла\*

4) После этого необходимо сравнить полученную в результате расчета контрольную сумму скрипта с эталонным значением контрольной суммы с помощью любых удобных средств. В случае, если хеши совпада-ют, значит скрипт действительный, не модифицированный и с ним можно работать, иначе его запуск запрещен.

Вторая часть методики предполагает собой автоматизацию с помо-щью виртуальной файловой системы, разворачиваемой с помощью сред-ства разработки «libfuse-dev». Необходимо сконфигурировать программ-ный код приложения FUSE таким образом, чтобы все действия, описанные во второй части методики происходили прозрачно для пользователя и ми-нимально влияли на общую производительность.