

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Дальневосточный федеральный университет»**

|  |  |
| --- | --- |
|  | **ШКОЛА ЕСТЕСТВЕННЫХ НАУК** |

**Кафедра информационной безопасности**

Пузан Сергей Александрович

|  |
| --- |
| РАЗРАБОТКА ПЛАТФОРМЫ МОНИТОРИНГА И СКАНИРОВАНИЯ ВРЕДОНОСНЫХ ФАЙЛОВ |

**КУРСОВАЯ РАБОТА**

по дисциплине «Защита в операционных системах»

Специальность 10.05.01 «Компьютерная безопасность»

Очной формы обучения

|  |
| --- |
| Студент группы С8117 ммзи \_\_\_\_\_\_\_\_\_  (подпись) |
| Руководитель  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) (и.о.ф) |
| Регистрационный № \_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись И.О.Фамилия  « \_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021г. | Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) (и.о.ф)  «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021г. |

г. Владивосток

2021

Оглавление

[Введение 3](#_Toc75383479)

[1. Классификация вредоносных файлов, их свойства. 5](#_Toc75383480)

[2. Методы анализа вредоносных файлов 8](#_Toc75383481)

[2.1 Статический анализ 8](#_Toc75383482)

[2.2 Динамический анализ 9](#_Toc75383483)

[3. Разработка платформы анализа и мониторинга вредоносных файлов 10](#_Toc75383484)

[3.1 Выбор необходимых для работы инструментов 10](#_Toc75383485)

[3.2 Общая схема работы платформы 12](#_Toc75383486)

[3.3 Настройка основной системы 15](#_Toc75383487)

[3.4 Настройка гостевой системы под управлением Windows 10 16](#_Toc75383488)

[3.5 Настройка гостевой системы под управлением Ubuntu 16.04 16](#_Toc75383489)

[3.6 Формирование команд на сканирование для выбранных антивирусных продуктов 17](#_Toc75383490)

[Заключение 19](#_Toc75383491)

[Список использованной литературы 20](#_Toc75383492)

[Приложение 21](#_Toc75383493)

# Введение

Любая инфраструктура, реализуемая с использованием информационных технологий, в той или иной степени подвержена угрозе нарушения информационной безопасности, причем атакам злоумышленников подвергаются как компании коммерческого, так и государственного секторов. Зачастую авторы вредоносных программ даже не нацеливают свое ПО против конкретных компаний – их целью является нанесение максимального вреда и придание огласке такого рода происшествий.

Подверженность различных инфраструктур атакам вредоносных программ объясняется не только наличием уязвимостей в используемом ими программном обеспечении (далее – ПО), но и в неправильной или недостаточной его настройке. Распространенным решением по защите компьютерной системы от различных видов вредоносного ПО является использование антивирусных программ, однако на данный момент ни один антивирусный продукт не может гарантировать абсолютную защиту от вредоносных файлов.

Одним из способов по уменьшению рисков заражения и получению более точных результатов анализа файлов является объединение нескольких технологий в одну платформу. Существуют сервисы, использующие на своей основе практически все известные антивирусные программные модули сканирования, что в конечном счете позволяет проанализировать отчеты о сканировании экземпляра от разных продуктов и сформировать наиболее точный вердикт. Основная проблема, встающая на пути использования подобных сервисов – конфиденциальность данных. В силу различных ограничений многие организации не могут свободно выгружать имеющиеся у них файлы на проверку.

В связи с этим целью данной работы является разработка платформы для локального мониторинга и анализа экземпляров вредоносных файлов. Для достижения цели поставлены следующие задачи:

1. Изучить основные классы вредоносных файлов, их свойства;
2. Изучить основные методы анализа файлов;
3. Подготовить системные окружения для безопасного анализа вредоносного ПО;
4. Разработать программу, реализующую управление задачами по сканированию файлов.

# Классификация вредоносных файлов, их свойства.

Вредоносное ПО можно классифицировать по достаточно большому спектру признаков: способам размножения, особенностям алгоритма работы и сложности кода, деструктивным возможностям, использованию интернет-технологий, способам запуска, средам обитания, способам проникновения в систему, времени воздействия и целевым операционным системам [3, 7]. Так как не существует единой классификации для всех видов и семейств вредоносных объектов, для примера приведем лишь некоторые примеры из классификации, представленной на сайте компании «Лаборатория Касперского» [6]. В соответствии с их классификацией вредоносные программы делятся на:

* Черви
  + Net-Worm
  + Email-Worm
  + IM-Worm
* Вирусы
* Троянские программы
  + Backdoor
  + Rootkit
  + Trojan-Spy
  + Trojan-Downloader
  + Trojan-Dropper
  + Trojan-Mailfinder
  + Trojan-DDoS
  + Trojan-Clicker
  + Trojan-Ransom
* Вредоносные утилиты
  + Exploit
  + VirTool
  + HackTool
  + Constructor
  + AdWare

В первую очередь программы делятся по способам их распространения – наиболее опасными являются черви, так как они могут распространяться через сети – по каналам обмена сообщениями, файлами, в одноранговых (P2P) сетях и т.д. Среди червей есть особый класс так называемых сетевых червей (Net-Worm). Их особенность заключается в том, что для распространения им не нужен пользователь, т.е. они могут самостоятельно распространяться и запускаться на уязвимых компьютерах. В основном это становится возможно при наличии критической уязвимости у компьютеров, подключенных к сети, позволяющей послать специально сформированный пакет, который позволит коду червя проникнуть на компьютер жертвы и запустит его на выполнение.

Следующими по способу распространения являются вирусы – они способны распространяться только в пределах локальных ресурсов компьютера. Распространение вирусных программ происходит следующим образом: вирус ищет необходимые ему файлы, открывает их с правами записи и дописывает к ним свой код. Далее, если пользователь запустит уже зараженный файл на своем компьютере, он будет пытаться заражать остальные документы. Вирусы зачастую дописывают себя в начало файла – в секцию, выполняющуюся самой первой при его запуске. В исполняемых файлах такой секцией является точка входа.

Далее – троянские программы. В их функционал не заложены механизмы самораспространения и они не могут заразить собой другие файлы. В классификации приведены не все виды, так как некоторые из них представляют собой лишь улучшенные или более узконаправленные их варианты. Ниже дадим краткое пояснение этим видам.

***Backdoor***

Backdoor - это класс вредоносных программ, позволяющий злоумышленнику получить контроль на компьютером жертвы и посылать на него различные команды. Специфика этих команд определяется автором трояна, однако при желании он может получить практически полный доступ к конечному узлу.

***Rootkit***

Зачастую в качестве таких программ выступают драйверы, предназначенные для сокрытия определенной активности в системе. В основном используются в связке с другим вредоносным ПО для уменьшения вероятности его обнаружения.

***Trojan-Spy***

К этому классу относятся вредоносные программы, ворующие личные данные жертвы и пересылающие их злоумышленнику. Пересылка может осуществляться практически любыми способами.

***Trojan-Downloader***

Это программы, скрыто загружающие из сети на компьютер жертвы другие вредоносные файлы и запускающие их на выполнение.

***Trojan-Dropper***

Класс программ-дропперов отличается тем, что они не загружают вредоносные программы из интернета, а распаковывают их из своего тела. В качестве вредоносного наполнения может служить не только исполняемый файл, но и самый обыкновенный скрипт, написанный на практически любом языке программирования. В таком случае в теле троянца иногда можно найти даже строчки кода, которые он впоследствии поместит в файл и запустит его на выполнение.

***Trojan-Mailfinder***

Класс программ, описывающий деятельность вредоносов, занимающихся поиском файлов, которые могут содержать в себе адреса электронных почт (к примеру, адресные книги). После сбора адресов она отправляет данные злоумышленнику.

***Trojan-DDoS***

К этому классу относятся вредоносные программы, проводящие атаки типа «отказ в обслуживании» на заранее указанный компьютер жертву.

***Trojan-Clicker***

Используются злоумышленниками либо для искусственной накрутки посещаемости сайтов, либо для привлечения потенциальных жертв заражения другими вредоносными файлами. Такие программы нередко представляют собой небольшой отрывок кода, встроенный в веб-страницы, который обращается к определенным веб-ресурсам в скрытом окне.

***Trojan-Ransom***

К ним относятся вредоносные программы-вымогатели, вынуждающие своих жертв платить выкуп за восстановление поврежденной, украденной информации, за возвращение доступа к заблокированным функциям компьютера и т.д.

Зачастую вредоносные программы сочетают в себе сразу большое количество функций, позволяющих им как распространяться по сети, так и, к примеру, проводить DDoS-атаки, поэтому конечный класс и вид программы определяется сначала ее способом распространение, а затем по вредоносной нагрузке. Так, например программа, загружающая на компьютер жертвы другую вредоносную программу, меняющая все ярлыки на рабочем столе, крадущая личные данные пользователя и распространяющаяся по локальным ресурсам компьютера в рамках данной классификации будет обнаружена антивирусом как “Virus”.

# 2. Методы анализа вредоносных файлов

Анализ вредоносных файлов можно разделить на два этапа – статический и динамический.

## 2.1 Статический анализ

Статический анализ подразумевает под собой исследование программы на наличие вредоносного кода без ее запуска. В него входит поиск необычных для чистых файлов артефактов, строк, ресурсов, чтение дизассемблированного кода программы, сверка сигнатур с базами данных, а также эвристический анализ (выполняется непосредственно антивирусом).

Эвристический анализ — это метод поиска программных дефектов в коде программы, подразумевающий под собой сопоставление фрагментов кода с имеющейся базой данных дефектов безопасности [4]. Здесь под дефектами безопасности понимаются недостатки в реализации программы, потенциально влияющие на состояние защищенности информации. Такой подход может быть очень полезен, к примеру, для обнаружения вредоносных программ с высокой степенью полиморфизма. Авторы некоторых вирусов разрабатывают их таким образом, чтобы вредоносный код формировался в процессе их исполнения и видоизменялся от раза к разу. Таким образом, невозможно создать универсальную сигнатуру для обнаружения подобного вируса – у каждого файла зараженная область будет практически полностью внешне отличаться, что не позволит антивирусу обнаружить ее по своим вирусным базам.

## 2.2 Динамический анализ

Под динамическим анализом понимают исследование поведения программы после ее непосредственного запуска. Такой подход осуществляется с помощью различного ПО, предназначенного для мониторинга процессов, служб, исходящего и входящего интернет-трафика, анализа загрузки памяти и т.д. Динамический анализ позволяет аналитику более широко понять принцип работы программы. В некоторых случаях авторы вредоносного ПО используют настолько сложные алгоритмы упаковки, шифрования, адресации и обфускации собственного кода и встроенных в них ресурсов, что иногда не представляется возможным достоверно определить класс и вредоносную нагрузку такой программы.

Так как динамический анализ подразумевает под собой запуск исследуемого объекта, в целях защиты компьютера от возможного вредоносного воздействия такого ПО используют различные методы, позволяющие изолировать его от внешнего мира. В случае, если с программой работает непосредственно вирусный аналитик, ее запускают на виртуальной машине с предустановленным ПО, необходимого для анализа.

Для автоматизации процесса анализа запущенной программы были созданы так называемые «песочницы» - изолированные среды, выполняющие определенную последовательность действий по запуску и мониторингу деятельности программы и собирающие всю информацию о ней в файлах-логах. Ранее эта технология была популярна среди антивирусного ПО, однако со временем стала реже ими использоваться ввиду высокой ресурсоемкости таких систем. На данный момент «песочницы» являются удобным инструментом для динамического анализа вредоносных программ, так как позволяют запускать в очередь на сканирование сразу несколько файлов и значительно сокращают время обработки результатов мониторинга. Аналитику лишь остается ознакомиться с результатами, которые выдаст ему изолированная среда, чтобы сделать соответствующие выводы.

Конечно, как утверждалось ранее, ни одна автоматизированная система анализа не позволяет быть стопроцентно уверенным в том, что тот или иной файл не является вредоносным, т.к. существует вероятность как ложной тревоги [4], так и того, что вредоносное ПО не будет обнаружено. К счастью, использование нескольких технологий статического и динамического анализа может позволить значительно сократить шанс подобного инцидента.

# 3. Разработка платформы анализа и мониторинга вредоносных файлов

## 3.1 Выбор необходимых для работы инструментов

Для достижения поставленной цели в данной работе были использованы следующие инструменты:

* VMWare Workstation 16 Player – средство виртуализации операционных систем, на которых будет проводиться дальнейший анализ;
* Windows 10 – ОС, на которой будет проводиться статический анализ файлов путем сканирования с помощью инструментов антивирусного ПО;
* Ubuntu 16.04 – ОС семейства Linux, используемая для работы «песочницы». На ней непосредственно будет производиться динамический анализ файлов;
* Kaspersky Security Cloud – бесплатный антивирус Лаборатории Касперского;
* ESET Internet Security 14.1.20 – пробная версия антивируса компании ESET;
* Защитник Windows – встроенное антивирусное ПО компании Microsoft;
* Cuckoo Sandbox – «песочница», используемая для динамического анализа файлов;
* VirtualBox - средство виртуализации операционных систем, используемое в рамках динамического анализа «песочницей» Cuckoo Sandbox;
* ElasticSearch+Kibana – платформа для быстрого и удобного поиска по результатам сканирования.

Приведенные выше антивирусы были выбраны по тем причинам, что не во всех бесплатных версиях антивирусных продуктов различных компаний есть сканирующий модуль, позволяющий инициировать операцию сканирования с использованием командной строки или стороннего ПО. В нашем случае у всех выбранных антивирусов присутствует функция, позволяющая проводить сканирование файлов с указанием определенных флагов прямо из командной строки.

Cuckoo Sandbox – один из бесплатных и наиболее распространенных продуктов по автоматизированному динамическому анализу вредоносного ПО. Помимо того, что он выдает полный отчет о работе файла, продукт также имеет собственный модуль по анализу результатов мониторинга и вынесению вердиктов. В связи с тем, что последняя выпущенная версия ПО была протестирована только на Ubuntu 16.04, было принято решение использовать именно эту версию ОС в целях стабильности работы «песочницы». Для своей работы Cuckoo Sandbox использует средство виртуализации VirtualBox, поэтому установлено оно будет непосредственно на Ubuntu 16.04.

## 3.2 Общая схема работы платформы

Реализация платформы для анализа и мониторинга вредоносных файлов подразумевает наличие трех связанных между собой узлов – одной хост-машины, с которой производится загрузка объектов для сканирования и проверка результатов, и двух гостевых виртуальных машин. В частности, виртуальные машины могут быть заменены и на реальные узлы, однако такое решение будет более уязвимым, так как случайно запущенный вредоносный файл на реальном компьютере может практически полностью вывести его из строя, и восстановление его работоспособности займет намного больше времени, нежели откат к сделанному заранее снимку состояния виртуальной машины.

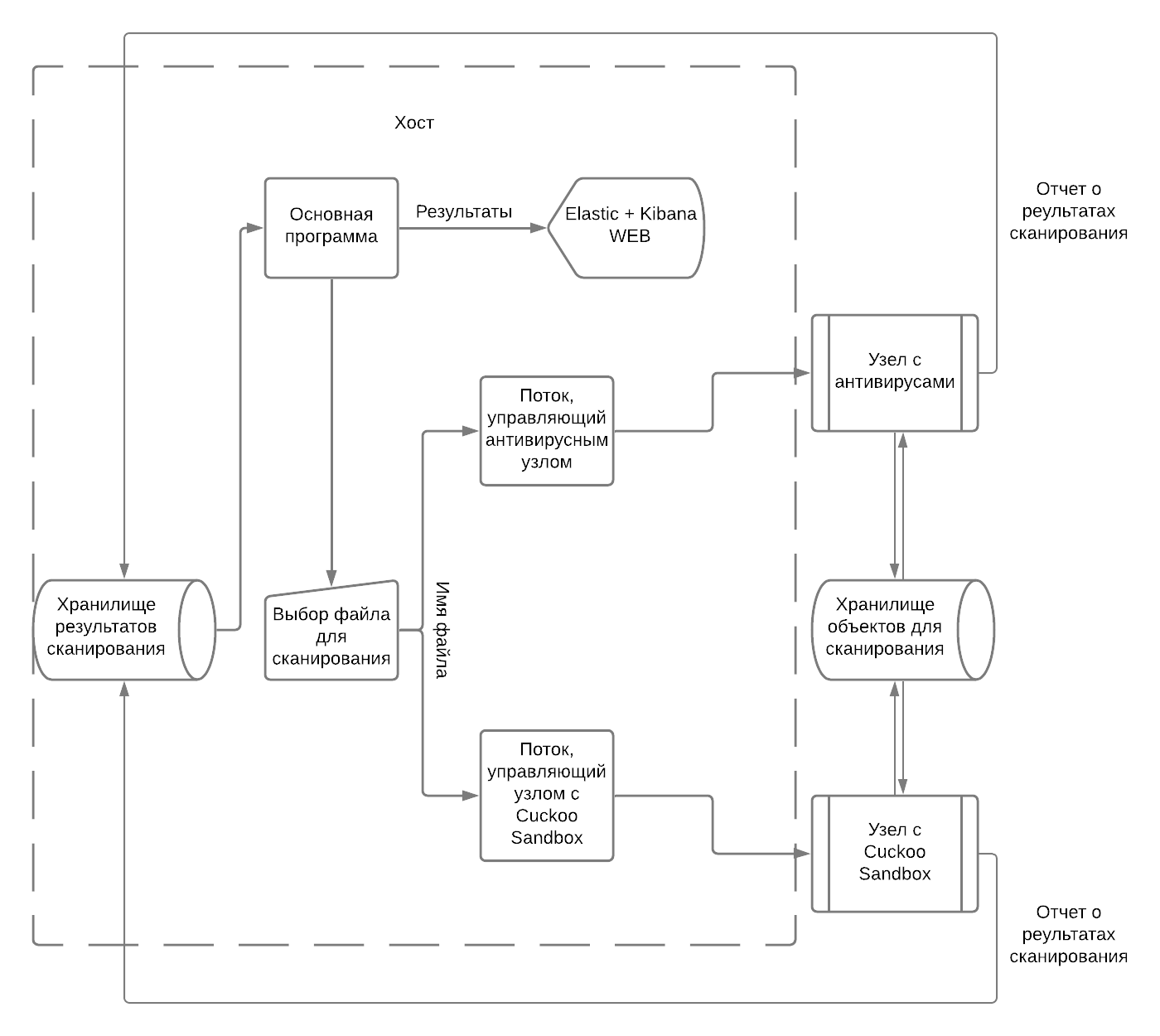
Приведем общую схему платформы: 

Рисунок 1. Общая схема платформы

Первый шаг – выбор файла для сканирования. Он осуществляется непосредственно пользователем и подразумевает под собой введение его имени. Далее программа создает два потока: в одном она формирует команды для машины с антивирусами, а во втором просто посылает имя файла для сканирования в CuckooSandbox и ожидает выполнения сканирования. Поток, управляющий работой антивирусов, сформирован таким образом, чтобы с главной машины можно было выбирать, какие антивирусные продукты мы хотим использовать для сканирования, и при этом не вносить изменений в программный код на гостевой машине.

Второй шаг – создание задач на сканирование. Клиентская часть программы на узле с антивирусами копирует файл в отдельную папку, выполняет посланные с основной машины команды на сканирование определенными продуктами с определенными параметрами и по их выполнении сообщает хосту о завершении работы, переходя в режим ожидания следующих команд. Узел с CuckooSandbox работает следующим образом – клиентская часть программы при помощи REST API для CuckooSandbox формирует POST запрос на сканирование, указывая имя файла. Далее она ежесекундно отправляет Cuckoo GET запрос, чтобы проверить готовность отчета о сканировании. Как только будет получено сообщение с кодом 200 – программа сразу копирует этот отчет в общую папку для отчетов, соответствующе переименовывая его. После того, как на серверную часть программы подаются сигналы о выполнении задачи, в тех же потоках происходит обработка результатов сканирования, и затем они передаются в тело функции main. В ней на последнем шаге формируется запрос на добавление записи в ElasticSearch и программа переходит в режим ожидания следующего файла для сканирования.

В итоге аналитик имеет возможность проанализировать результаты отчета в удобной форме, настраивая необходимые ему поля прямо в браузере. Пример отчета выглядит следующим образом: 

Рисунок 2. Пример отчета о сканировании файлов, загруженный в Elastic

Тут же можно в виде выпадающего списка более детально ознакомиться с результатами сканирования:

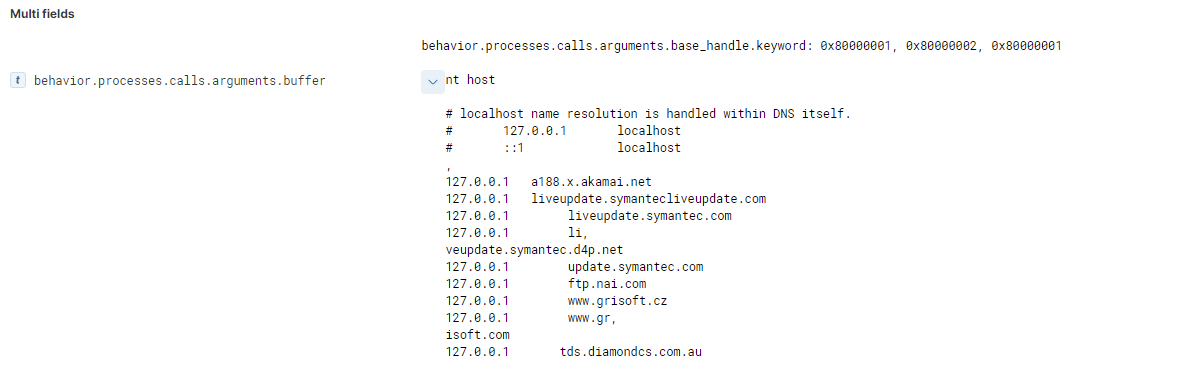


Рисунок 3. Пример содержания одного из полей отчета о сканировании.

## 3.3 Настройка основной системы

Так как в рамках работы будут использоваться вредоносные экземпляры, необходимо максимально полно исключить вероятность их непроизвольного запуска. Для этого все файлы должны перемещаться внутри основной системы в архиве, защищенном паролем.

При создании виртуальной машины под управлением Ubuntu необходимо включить средства вложенной виртуализации Intel VT или AMD-V в зависимости от используемого в компьютере процессора. Эти средства позволят в дальнейшем Cuckoo Sandbox запускать собственную виртуальную машину для анализа. Еще одно важное условие – объем оперативной памяти и количество ядер процессора, выделяемых системе. Даже для анализа не более чем одного файла за раз необходимо достаточно большое количество памяти ввиду того, что мы запускаем виртуальную машину внутри виртуальной машины. В рамках работы для Ubuntu было выделено 5120мб оперативной памяти и 4 ядра процессора.

Для удобства обмена файлами и отчетами с виртуальными машинами необходимо создать директорию с функцией общего доступа. В данном случае функцию общего доступа достаточно настроить непосредственно в средстве виртуализации, а не глобально на компьютере. Таким образом, доступ к этой папке будут иметь только установленные виртуальные машины. Так как в папке с экземплярами могут оказаться вредоносные файлы, необходимо с помощью настроек групповых политик запретить запуск файлов из этой папки для всех пользователей системы. Для этого в редакторе групповых политик создается правило, запрещающее исполнение файлов из этой папки и указывается список расширений, попадающих под это правило. В качестве дополнительной меры предосторожности стоит всем предназначенным для сканирования файлам изменять расширение на неподходящее для запуска. Для удобства положим, что все файлы должны иметь расширение «.to\_scan\_». Это расширение также необходимо добавить в правило о запрете на запуск из общей папки.

## 3.4 Настройка гостевой системы под управлением Windows 10

В первую очередь необходимо применить тот же комплекс мер по предотвращению запуска файла, что и на основной системе. Далее следует установка антивирусного ПО.

Для того, чтобы избежать конфликтов во время работы антивирусов, необходимо после их установки полностью выключить все модули сканирования и мониторинга событий в системе, а также добавить папки с объектами для сканирования в исключения. Помимо этого, чтобы работало сканирование с использованием командной строки, необходимо добавить пути к исполняемым файлам антивирусов в переменную среду PATH. Также для работы понадобится Python3. Финальным шагом в настройке будет создание папки для объектов сканирования, к примеру, на рабочем столе. Это делается по той причине, что антивирусы не позволяют выполнять сканирование удаленных дисков путем запуска из командной строки, а в данном случае папка с общим доступом видна гостевой системе как удаленный жесткий диск.

## 3.5 Настройка гостевой системы под управлением Ubuntu 16.04

Несмотря на то, что вместе с Ubuntu поставляется достаточно большой набор инструментов, для работы Cuckoo Sandbox необходимо установить довольно большое количество зависимостей [9]. После установки этих зависимостей необходимо создать нового пользователя без прав на выполнение root-команд. Это требование необходимо соблюдать в целях безопасности работы «песочницы», так как некоторые вредоносные программы могут быть оснащены модулями обнаружения и попытки эксплуатирования уязвимостей. Даже если подобное произойдет, зловреду будет намного сложнее нанести ущерб гостевой машине из-под пользователя с низким уровнем привилегий [9]. Созданный пользователь должен иметь возможность создавать дампы интернет-трафика и быть членом группы vboxusers для создания и запуска виртуальных машин.

Далее непосредственно подготовка виртуальной машины для «песочницы». В этом случае аналитик сам определяет, какая система ему нужна в качестве среды для тестирования. На момент выполнения работы наиболее подходящей является Windows 7, так как она менее ресурсоемка по сравнению с Windows 10 и при этом не является слишком устаревшей для тестирования современных вредоносных программ. Подготовить виртуальную машину для Cuckoo можно двумя способами – вручную, либо с использованием инструмента vmcloak. Он позволяет намного быстрее выполнить установку операционной системы с необходимым набором ПО и не требует от пользователя ничего, кроме списка этого ПО.

Установку виртуальной машины и самой Cuckoo необходимо проводить с использованием виртуального окружения. Это делается для того, чтобы избежать конфликтов с другими, глобально установленными пакетами в Python. После установки Cuckoo и виртуальных машин с помощью vmcloak необходимо указать в файлах конфигурации «песочницы» ip-адрес и имя виртуальной машины, а также включить функцию использования интернета на виртуальной машине, чтобы иметь возможность анализировать сетевую активность вредоносной программы.

## 3.6 Формирование команд на сканирование для выбранных антивирусных продуктов

Так как у бесплатных антивирусных решений нет своего интерфейса прикладного программирования (API), взаимодействовать с ними напрямую с помощью написанной нами программы не получится. В рамках работы для управления сканированиями будут использованы консольные команды. Определим необходимые для нас условия сканирования: антивирус должен просканировать указанный файл, сохранить событие в качестве отчета в определенную папку и при этом не предпринимать никаких действий после сканирования.

Для используемых в рамках работы антивирусов получились следующие команды:

* Антивирус Касперсого: «avp.com SCAN /i0 "Path\_to\_file"», где флаг /i0 включает режим «только отчет»;
* Антивирус ESET: «ecls /no-quarantine /adv-heur “Path\_to\_file”», где флаг /no-quarantine отключает реакцию на обнаружение файла, а /adv-heur включает глубокий эвристический анализ;
* Windows Defender: «MpCmdRun.exe -Scan -ScanType 3 -DisableRemediation -File “Path\_to\_file”, где -ScanType указывает, что мы сканируем конкретный файл, а -DisableRemediation отключает реакцию на обнаружение вредоносного файла.

# Заключение

Существует большое количество критериев для классификации вредоносных файлов, однако наиболее общей является классификация по способам их распространения. Таким образом, можно выделить 4 класса программ: черви, вирусы, троянские программы и вредоносные утилиты.

Для анализа файлов применяют два основных метода – статический и динамический. Статический метод анализа подразумевает под собой исследование программы на наличие вредоносного кода, артефактов без ее непосредственного запуска, а динамический – мониторинг поведения файла после запуска.

В рамках работы было подготовлено следующее окружение: настроена основная и две виртуальные машины под управлением Windows 10 и Ubuntu 16.04, созданы необходимые правила в редакторе групповых политик для предотвращения непроизвольного запуска программ, подготовлены общие папки для обмена файлами.

Реализованная программа позволяет не только управлять задачами по сканированию файлов, но также выполняет функцию диспетчера отчетов, направляя их в соответствующий модуль для удобства чтения аналитиком.

# Список использованной литературы

1. Аветисян, А. И. Технологии статического и динамического анализа уязвимостей программного обеспечения / А. И. Аветисян, А. А. Белеванцев, И. И. Чукляев // Вопросы кибербезопасности. - 2014. - №3 (4). – С. 20-28.
2. Бородавкин, Д.А. Система вынесения вердиктов по отчетам Cuckoo Sandbox // Решетневские чтения. - 2015. - № 19. – С. 271-273.
3. Ерёменко С. П., Классификация вредоносных программ / С. П. Ерёменко, А. И. Сапелкин, С. Б. Хитов // Вестник Санкт-Петербургского университета ГПС МЧС России. – 2016. - № 3. – С. 55-61.
4. Марков, А. С. Эвристический анализ безопасности программного кода / А. С. Марков, В. А. Матвеев, А. А. Фадин, В.Л. Цирлов // Вестник МГТУ им. Н.Э. Баумана. - 2016. – Т.1. - № 106. – С. 98-111.
5. Сильнов, Д. С. О ложных срабатываниях средств защиты информации / Д. С. Сильнов, О. В. Тараканов // Прикладная информатика. - 2015. – Т.2 № 56. – С. 68-79.
6. Типы детектируемых объектов [Электронный ресурс]. – Режим доступа: https://encyclopedia.kaspersky.ru/knowledge/the-classification-tree/.
7. Классификация компьютерных вирусов [Электронный ресурс]. – Режим доступа: https://math-it.petrsu.ru/users/semenova/Informatika/DOC/MUM\_65/Lections/Comp\_bez\_Lec.pdf.
8. Вредонос под наблюдением. Как работают сендбоксы и как их обойти [Электронный ресурс]. – Режим доступа: https://xakep.ru/2020/06/08/malware-sandbox/.
9. Cuckoo Sandbox Setup for People in a Hurry [Электронный ресурс]. – Режим доступа: https://hatching.io/blog/cuckoo-sandbox-setup/.

# Приложение

Приложение 1 – Программный код управляющей части программы

import threading  
import socket  
from cuckoo\_main import cuckoo\_thread  
from av\_main import av\_thread  
from elasticsearch import Elasticsearch  
  
reports = [{}, {}]  
  
av\_machine\_ip = '192.168.58.128'  
av\_machine\_port = 4433  
  
cuckoo\_port = 4433  
cuckoo\_machine\_ip = '192.168.58.129'  
  
try:  
 av\_sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 av\_sock.connect((av\_machine\_ip, av\_machine\_port))  
 print('Binded av sock')  
except Exception:  
 print(Exception)  
 no\_av\_task = True  
  
try:  
 cuckoo\_sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 cuckoo\_sock.connect((cuckoo\_machine\_ip, cuckoo\_port))  
 print('Binded cuckoo sock')  
except Exception:  
 print(Exception)  
 no\_cuckoo\_task = True  
  
es = Elasticsearch()  
  
while True:  
 no\_av\_task = False  
 no\_cuckoo\_task = False  
 print('Файл берется из общей с виртуальными машинами папки. Введите его имя:')  
 filename = input()  
  
 if not no\_av\_task:  
 av\_task = threading.Thread(target=av\_thread, args=(filename, av\_sock, reports, ))  
 av\_task.start()  
  
 if not no\_cuckoo\_task:  
 cuckoo\_task = threading.Thread(target=cuckoo\_thread, args=(filename, cuckoo\_sock, reports, ))  
 cuckoo\_task.start()  
 if not no\_av\_task:  
 av\_task.join()  
 if not no\_cuckoo\_task:  
 cuckoo\_task.join()  
  
 data = reports[0]  
 data.update(reports[1])  
 res = es.index(index='malware\_testing', body=data)  
 print(res['result'])  
  
 if filename == 'QUIT':  
 break  
  
print('Connection closed')

Приложение 2 – Программная реализация функции, управляющей антивирусным узлом с хост машины

import base64  
import time  
import av\_report\_parser  
  
HEADER = 4  
FORMAT = 'utf-8'  
  
  
AV\_list = {'AVP': 'SCAN;/i0',  
 'ESET': '/no-quarantine;/adv-heur',  
 'WinAV': '-Scan;-ScanType 3;-DisableRemediation;-File'}  
  
REPORTS\_PATH = 'D:\\машины\\Shared\\Reports\\'  
  
  
def av\_thread(filename, sock, report\_handle):  
 do\_parse = True  
  
 print(type(sock))  
 if filename == 'QUIT':  
 msg = 'QUIT'  
 base64\_msg = base64.b64encode(msg.encode(FORMAT))  
 msg\_len = len(base64\_msg)  
 sock.send(('{:<}'.format(msg\_len, HEADER)).encode(FORMAT))  
 time.sleep(0.05)  
 sock.send(base64\_msg)  
 sock.close()  
 return "QUIT"  
  
 for av in AV\_list:  
  
 msg = f'{av};{AV\_list[av]};{filename}.{av};{filename}'  
 base64\_msg = base64.b64encode(msg.encode(FORMAT))  
 msg\_len = len(base64\_msg)  
 sock.send(('{:<}'.format(msg\_len, HEADER)).encode(FORMAT))  
 time.sleep(0.1)  
 sock.send(base64\_msg)  
 response = sock.recv(HEADER).decode(FORMAT)  
 if response == 'ER-2':  
 print('[AV]: File not found error')  
 do\_parse = False  
 break  
  
 if do\_parse:  
 results = av\_report\_parser.parse\_results([filename + '.AVP', filename + '.ESET', filename + '.WinAV'], REPORTS\_PATH)  
 with open('Total reports.txt', 'a', encoding='utf-8') as file:  
 report\_handle[0] = {}  
 for elem in results:  
 report\_handle[0].update({str(elem): str(results[elem])})  
 print("[AV]: " + str(results))

Приложение 3 – Программная реализация функции, управляющей узлом с Cuckoo Sandbox с хост машины

import base64  
import time  
import json  
  
HEADER = 4  
FORMAT = 'utf-8'  
  
REPORT\_PATH = 'D:\\машины\\Shared\\Reports\\'  
  
  
def cuckoo\_thread(filename, sock, report\_handle):  
  
 msg = f'{filename}'  
 base64\_msg = base64.b64encode(msg.encode(FORMAT))  
 msg\_len = len(base64\_msg)  
 sock.send(('{:<}'.format(msg\_len, HEADER)).encode(FORMAT))  
 time.sleep(0.1)  
 sock.send(base64\_msg)  
 while True:  
 response = sock.recv(HEADER).decode(FORMAT)  
 if response:  
 break  
 if response == 'ER-2':  
 print('[CUCKOO]: File not found error')  
 return response  
 else:  
 time.sleep(3)  
 with open(REPORT\_PATH + filename + ".CUCKOO", "r") as report:  
 report\_handle[1] = json.loads(report.read())  
 return "Next"  
 sock.close()

Приложение 4 – Программная реализация клиента, запускающего задачи на сканирование антивирусами со стороны гостевой машины

import subprocess  
import socket  
import base64  
import os  
import shutil  
import codecs  
  
# SHARED FOLDER SAMPLES PATH  
NET\_PATH = '"Z:\\Shared\\Samples\\'  
  
# VM FOLDER SAMPLES PATH  
SAMPLE\_PATH = 'C:\\Users\\AVHost\\Desktop\\Samples\\'  
  
# SHARED FOLDER REPORTS PATH  
REPORT\_PATH = '"Z:\\Shared\\Reports\\'  
  
CWD = os.path.abspath(os.curdir)  
  
HEADER = 4  
FORMAT = 'utf-8'  
  
sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
sock.bind(('0.0.0.0', 4433))  
  
sock.listen(3)  
  
print('Binded sock')  
  
kaspersky = None  
eset = None  
winav = None  
flag = False  
prev\_filename = None  
  
  
def process\_msg(con, addr):  
 msg\_len = con.recv(HEADER)  
 msg\_len = msg\_len.decode(FORMAT)  
 if msg\_len:  
 msg = base64.b64decode(con.recv(int(msg\_len))).decode('utf-8')  
 return msg  
  
  
def do\_scan(name, params, file):  
 params[0] = name  
 try:  
 print(' '.join(elem for elem in params))  
 subprocess.call(' '.join(elem for elem in params), stdout=file, stderr=file, shell=True)  
 return 'Next'  
 except Exception:  
 return 'Er-r'  
  
  
def cpy\_f(filename):  
 global SAMPLE\_PATH  
 global NET\_PATH  
 global ts  
 try:  
 open(NET\_PATH[1:] + filename)  
 try:  
 open(SAMPLE\_PATH + filename)  
 print('Such already file exists. Creating a new instance.')  
 shutil.copyfile(NET\_PATH[1:] + filename, SAMPLE\_PATH + filename)  
 except FileNotFoundError:  
 print('Copying file.')  
 shutil.copyfile(NET\_PATH[1:] + filename, SAMPLE\_PATH + filename)  
 return 1  
 except FileNotFoundError:  
 print('No such file')  
 return -1  
  
  
while True:  
 con, addr = sock.accept()  
 while con:  
 res = process\_msg(con, addr)  
 if res == 'QUIT':  
 flag = True  
 break  
 params = res.split(';')  
  
 curr\_filename = params[len(params) - 1]  
 state = 1  
 if curr\_filename != prev\_filename:  
 state = cpy\_f(curr\_filename)  
 prev\_filename = curr\_filename  
  
 if state == -1:  
 con.send('ER-2'.encode(FORMAT))  
 continue  
  
 params[len(params) - 1] = SAMPLE\_PATH + curr\_filename  
 params[len(params) - 2] = REPORT\_PATH + params[len(params) - 2] + '"'  
  
 try:  
 file = codecs.open(os.path.join(CWD, params[len(params) - 2][1:-1]), 'wb', encoding='utf-8-sig')  
 except FileNotFoundError:  
 file = codecs.open(params[0] + '.templog', 'wb', encoding='utf-8-sig')  
 params.remove(params[len(params) - 2])  
 if res[0:3] == 'AVP':  
 con.send(do\_scan('AVP.com', params, file).encode(FORMAT))  
  
 elif res[0:4] == 'ESET':  
 con.send(do\_scan('ecls', params, file).encode(FORMAT))  
  
 elif res[0:5] == 'WinAV':  
 con.send(do\_scan('"%ProgramFiles%\Windows Defender\MpCmdRun.exe"', params, file).encode(FORMAT))  
 if os.path.exists(f"{SAMPLE\_PATH + prev\_filename}"):  
 os.remove(f"{SAMPLE\_PATH + prev\_filename}")  
 else:  
 print('No sample exists')  
 file.close()  
  
 if flag:  
 sock.close()  
 break

Приложение 5 – Программная реализация клиента, запускающего задачи на сканирование с помощью Cuckoo Sandbox со стороны гостевой машины

import requests  
import socket  
import base64  
import time  
  
# SHARED FOLDER SAMPLES PATH  
NET\_PATH = '/mnt/hgfs/Shared/Samples/'  
  
# SHARED FOLDER REPORTS PATH  
REPORT\_PATH = '/mnt/hgfs/Shared/Reports/'  
  
# CUCKOO REPORTS PATH  
CUCKOO\_REPORTS = '/home/cuckoo/.cuckoo/storage/analyses/'  
  
HEADER = 4  
FORMAT = 'utf-8'  
  
REST\_URL\_CREATE\_TASK = "http://localhost:8090/tasks/create/file"  
  
REST\_URL\_TASK\_REPORT = "http://localhost:8090/tasks/report/"  
  
API\_KEY = {"Authorization": "Bearer iSj7lKm0u3HY\_-xk2vHJaQ"}  
  
sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
sock.bind(('0.0.0.0', 4433))  
  
sock.listen(3)  
  
  
def process\_msg(con, addr):  
 msg\_len = con.recv(HEADER)  
 msg\_len = msg\_len.decode(FORMAT)  
 if msg\_len:  
 msg = base64.b64decode(con.recv(int(msg\_len))).decode('utf-8')  
 return msg  
  
  
def check\_task\_status(id):  
 r = requests.get(REST\_URL\_TASK\_REPORT + str(id), headers=API\_KEY)  
 return r  
  
  
def do\_scan(SAMPLE\_FILE):  
 print("Start scanning")  
 with open(SAMPLE\_FILE, "rb") as sample:  
 files = {"file": (SAMPLE\_FILE, sample)}  
 r = requests.post(REST\_URL\_CREATE\_TASK, headers=API\_KEY, files=files)  
 task\_id = r.json()["task\_id"]  
 iterator = 300  
 while True:  
 res = check\_task\_status(task\_id)  
 if '200' in str(res):  
 return task\_id  
 time.sleep(1)  
 iterator -= 1  
 if iterator == 0:  
 break  
  
  
def cpy\_f(filename, task\_id):  
 global NET\_PATH  
 global REPORT\_PATH  
 try:  
 print('Copying new report.')  
 src = CUCKOO\_REPORTS + str(task\_id) + "/reports/report.json"  
 dst = REPORT\_PATH + filename + ".CUCKOO"  
 length = 16 \* 1024 \* 1024  
 with open(src, 'r') as fsrc:  
 with open(dst, 'w') as fdst:  
 while 1:  
 buf = fsrc.read(length)  
 print(buf)  
 if not buf:  
 break  
 fdst.write(buf)  
  
  
  
  
 except Exception:  
 print('Error coyping')  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
  
 while True:  
 print("Client ready for incoming files")  
 con, addr = sock.accept()  
 while con:  
 res = process\_msg(con, addr)  
 if res == None:  
 continue  
 if res == 'QUIT':  
 flag = True  
 break  
 print("Got message from server: " + str(res))  
 task\_id = do\_scan(NET\_PATH + str(res))  
 if task\_id == "ER-2":  
 con.send(("ER-2").encode(FORMAT))  
 else:  
 cpy\_f(str(res), task\_id)  
 con.send(("Next").encode(FORMAT))  
 flag = True  
  
 if flag:  
 sock.close()  
 break

Приложение 6 – Программная реализация парсера результатов сканирования антивирусными продуктами

def parse\_results(files, rep\_path):  
 results = {}  
 for i in range(len(files)):  
 with open(rep\_path + files[i], 'r', encoding='utf-8') as f:  
 if '.AVP' in files[i][-4:]:  
 line = f.readline()  
 while 'Time Finish:' not in line:  
 line = f.readline()  
 total\_objects = avp\_get\_num(f.readline())  
 ok\_objects = avp\_get\_num(f.readline())  
 detected\_objects = avp\_get\_num(f.readline())  
 suspicious\_objects = avp\_get\_num(f.readline())  
 skipped\_objects = avp\_get\_num(f.readline())  
 if ok\_objects == total\_objects:  
 results.update({'AVP': 'Clean'})  
 elif detected\_objects > 0:  
 results.update({'AVP': 'Malicious'})  
 elif suspicious\_objects > 0:  
 results.update({'AVP': 'Suspicious'})  
 elif skipped\_objects > 0:  
 results.update({'AVP': 'Clean, some objects skipped'})  
 else:  
 print('wrong smth')  
 elif '.ESET' in files[i][-5:]:  
 line = f.readline()  
 while 'Всего: ' not in line:  
 line = f.readline()  
 detected\_objects = eset\_get\_num(f.readline())  
 if detected\_objects > 0:  
 results.update({'ESET': 'Malicious'})  
 else:  
 results.update({'ESET': 'Clean'})  
 elif '.WinAV' in files[i][-6:]:  
 for \_ in range(2):  
 f.readline()  
 line = f.readline()  
 if 'found no threats' in line:  
 results.update({'WinAV': 'Clean'})  
 else:  
 results.update({'WinAV': 'Malicious'})  
 else:  
 print('wrong smth BIG')  
 return results  
  
  
def avp\_get\_num(line):  
 delim\_ind = line.find(':')  
 return int(line[delim\_ind + 1:].strip())  
  
  
def eset\_get\_num(line):  
 delim\_ind = line.find('—')  
 com\_ind = line.find(',')  
 return int(line[delim\_ind+1:com\_ind].strip())