Федеральное государственное бюджетное образовательное учреждение   
высшего образования

|  |  |
| --- | --- |
| Gerb-BMSTU_01 | «Московский государственный технический университет  им. Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана) |

ФАКУЛЬТЕТ – Информатика и управления

КАФЕДРА – Информационные системы и телекоммуникации

РАСЧЁТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

По курсу

Разработка программного обеспечения

На тему

Разработка OSGI сервиса

настройки правил перехвата пакетов IDS системы Suricata

Студент группы ИУ3-73 (подпись) 13.02.2018 Николаев Д.Л.

Руководитель курсовой работы (подпись) 13.02.2018 Иванов А.М.

Содержание

[1 Введение 4](#_Toc506117055)

[2 Техническое задание 5](#_Toc506117056)

[3 Теоретическая часть 5](#_Toc506117057)

[3.1 Выявление заинтересованных сторон и их интересов 5](#_Toc506117058)

[3.2 IDS Suricata 5](#_Toc506117059)

[4 Конструкторская часть 7](#_Toc506117060)

[4.1 Выбор технических решений, удовлетворяющих интересам ЗС 7](#_Toc506117061)

[4.2 Структура проекта 8](#_Toc506117062)

[4.2.1 Плагин com.bmstu.suricata.settings.manager 8](#_Toc506117063)

[4.2.2 Плагин com.bmstu.suricata.settings.manager.client 10](#_Toc506117064)

[4.2.3 Плагин com.bmstu.suricata.settings.manager.service 11](#_Toc506117065)

[4.3 Диаграмма компонентов 14](#_Toc506117066)

[4.4 Диаграмма классов 15](#_Toc506117067)

[5 Технологическая часть 15](#_Toc506117068)

[5.1 Запуск разработанного приложения 15](#_Toc506117069)

[5.2 Анализ исходного кода с помощью метрик качества 16](#_Toc506117070)

[5.3 Анализ зависимостей в коде системы 18](#_Toc506117071)

[5.4 Тестирование на корректность работы 18](#_Toc506117072)

[5.5 Оценка покрытия кода. 19](#_Toc506117073)

[6 Выводы 20](#_Toc506117074)

[7 Список источников 21](#_Toc506117075)

# 

# Введение

Естественной потребностью системного администратора или специалиста безопасности является некий анализ того, что происходит как на конкретном компьютере конкретного пользователя, так и в локальной сети. Для этой цели существуют так называемые системы обнаружения вторжений (англ. IDS).

Системы обнаружения вторжения – это работающие процессы или устройства, анализирующие активность в сети или системе на предмет неавторизованных или злонамеренных действий. Способы выявления аномалий системой обнаружения вторжения могут быть самыми разыми; однако все системы IDS предназначены для поимки преступников на месте, прежде чем они действительно нанесут ущерб.

IDS защищает компьютер от атаки, незаконного использования и компрометации. Эта система также наблюдает за сетевой активностью, анализирует целостность данных, проводит аудит сетевой и системной конфигурации. В зависимости от методов выявления, которые вы решили внедрять, использование IDS принесет различные прямые и косвенные выгоды.

Целью данной курсовой работы является разработка такого программного обеспечения, которое поможет произвести настройку правил перехвата пакетов системой обнаружения вторжений Suricata.

# Техническое задание

Разработка OSGi сервиса настройки правил перехвата пакетов IDS системы Suricata:

* изучить соответствующую систему;
* спроектировать интерфейс компонента;
* реализовать компонент;
* спроектировать JUnit тесты, провести тестирование;
* описать требования, конструкцию, особенности сборки и запуска в документации.

# Теоретическая часть

## Выявление заинтересованных сторон и их интересов

В таблице ниже представлены результаты выявления и начального анализа заинтересованных сторон (ЗС) и их интересов по отношению к системе.

Таблица 1. Заинтересованные стороны и их интересы по отношению к системе

|  |  |
| --- | --- |
| **Заинтересованные стороны** | **Интересы заинтересованных сторон** |
| Системный администратор | Добавление правил в конфигурационный файл IDS Suricata  Удаление правил из конфигурационного файла IDS Suricata |
| Владелец опенсорсного проекта (project owner) | Быстрая и полная передача исходного кода, настроек, документов.  Возможность в дальнейшем совершенствовать систему, например, добавляя новые модули и возможности |
| Новый разработчик системы | Быстро понять принцип работы системы.  Быстро внести изменения и проверить, что они не нарушают работу существующего функционала. |

## IDS Suricata

Сегодня системы обнаружения и предотвращения вторжений (IDS/IPS, Intrusion detection system / Intrusion prevention system) — необходимый элемент защиты от сетевых атак. Основное предназначение подобных систем — выявление фактов неавторизованного доступа в корпоративную сеть и принятие соответствующих мер противодействия: информирование ИБ-специалистов о факте вторжения, обрыв соединения и перенастройка межсетевого экрана для блокирования дальнейших действий злоумышленника, т. е. защита от хакерских атак и вредоносных программ.

Популярный Snort начал разрабатываться в 1998 году, когда для обработки сетевого трафика обычно использовался сервер с одним 32-битным процессором. Поэтому и была использована единственно актуальная в то время single-threaded архитектура — другая не имела смысла. За более чем 17 лет разработок много изменилось в компьютерном мире: появились многоядерные процессоры, IPv6, увеличилось количество пользовательских приложений, и, главное, стал большим трафик. Это все нашло отражение в Snort: поддержка IPv6, возможность инспектирования уровня приложений, универсальный модуль доступа к данным DAQ и многое другое. Но базовый движок, хотя и научился работать с несколькими ядрами, так и остался однопоточным. И фактически Snort оказался не готов к переходу на настоящий multi-threading.

Когда Snort не справлялся с нагрузкой, то просто увеличивалась мощность CPU или использовалось несколько экземпляров с балансировкой нагрузки между ними. Но проблемы это не решает, поэтому и возникли проекты вроде Gnort, задача которого — повысить эффективность обнаружения атак Snort за счет переноса на GPU кода, отвечающего за проверку регулярных выражений. Это позволило добиться почти двукратного увеличения пропускной способности Snort. Но проблема многопоточности требует полной модернизации всех важных компонентов внутри Snort.

В 2009 году несколько частных компаний создали организацию Open Information Security Foundation (OISF), основной из задач которой было финансирование и разработка многопоточной альтернативы Snort, получившей название Suricata.

Suricata изначально работает в многопоточном режиме, позволяющем оптимально использовать несколько CPU. До релиза 1.3 были некоторые проблемы с масштабируемостью, например, количество ядер больше четырех не давало в тестах прироста скорости. Теперь все проблемы решены и Suricata вполне эффективно работает с 24 и более процессорами. Кроме того, Suricata может использовать вычисления на стороне GPU (CUDA и OpenCL). В итоге эта IDS спокойно справляется на обычном оборудовании с потоками до 10 Гбит/с.

Главная особенность Suricata — то, что, кроме своих уникальных наработок, она использует практически все, что уже наработано для Snort. Одним из преимуществ Suricata является возможность обработки 7-го уровня OSI, что повышает ее способность обнаруживать вредоносные программы для приложений. Движок автоматически определяет и парсит протоколы (IP, TCP, UDP, ICMP, HTTP, TLS, FTP, SMB, SMTP и другие), поэтому в правилах можно строго не привязываться к номеру порта, как это сделано в Snort, достаточно указать протокол и действие. Дальше модули Suricata сами уже разберутся с трафиком и обнаружат протокол, даже если используется нестандартный порт.

Учитывая, что наиболее интенсивной работой, выполняемой системой обнаружения вторжений, является детектирование пакетов, разработчики Suricata решили использовать потоки для обнаружения. На рисунке 1 приведен пример создания трех потоков детектирования. Suricata может получать сетевой трафик с карты сетевого интерфейса или из ранее записанного сетевого трафика из файла, хранящегося в формате PCAP. Трафик передается через модуль декодирования, где он сначала декодируется в соответствии с его протоколом, затем потоки повторно собираются перед распределением между модулями обнаружения сигнатуры.

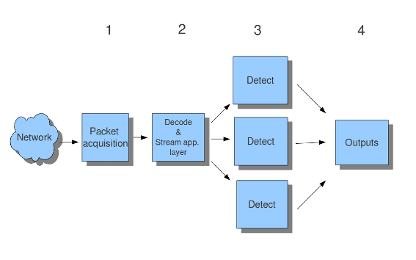


Рисунок 1. Пример работы Suricata

# Конструкторская часть

## Выбор технических решений, удовлетворяющих интересам ЗС

В таблице ниже представлены результаты выбора технических решений, позволяющие удовлетворить интересы заинтересованных сторон по отношению к системе.

Таблица 2. Технические решения, удовлетворяющие интересам ЗС

|  |  |
| --- | --- |
| **Интересы заинтересованных сторон** | **Технические решения** |
| Добавление правил в конфигурационный файл IDS Suricata  Удаление правил из конфигурационного файла IDS Suricata | Новое правило заносится одним из классов разработанной системы в конфигурационный файл.  Существующее правило удаляется одним из классов разработанной системы в конфигурационном файле.  Корректность работы будет проверяться в процессе разработки с помощью юнит тестов. |
| Быстрая и полная передача исходного кода, настроек, документов.  Возможность в дальнейшем совершенствовать систему, например, добавляя новые модули и возможности | Код и настройки разрабатываемого менеджера будут находиться в системе GitHub. Контроль версий будет производиться с использованием системы Git.  Для модульного тестирования будет использована система JUnit. |
| Быстро понять принцип работы системы.  Быстро внести изменения и проверить, что они не нарушают работу существующего функционала. | Написаны аннотации с помощью Javadoc  Исходный код будет структурирован по пакетам. Повторно-используемые методы будут вынесены в родительские классы. |

## Структура проекта

Разберем работу написанной системы, для этого опишем состав каждого плагина, из которых состоит наша система:

Таблица 3. Разработанные плагины и их описание

|  |  |
| --- | --- |
| **Плагин** | **Содержимое** |
| com.bmstu.suricata.settings.manager | Содержит интерфейс сервиса и модель клиентов разрабатываемого ПО |
| com.bmstu.suricata.settings.manager.client | Содержит реализацию клиентов |
| com.bmstu.suricata.settings.manager.service | Содержит реализацию сервиса |

### Плагин com.bmstu.suricata.settings.manager

Данный плагин содержит 2 пакета:

* com.bmstu.suricata.settings.manager
* com.bmstu.suricata.settings.manager.model

Пакет com.bmstu.suricata.settings.manager содержит интерфейс ISuricataSettingsManager, который включает в себя 4 метода:

Таблица 4. Методы интерфейса ISuricataSettingsManager и их описание

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| **void** addRuleSet(RuleSet ruleSet) | Добавляет список правил в конфигурационный файл Suricata. На вход принимает список правил, который не может быть пуст. |
| **void** removeRuleSet(RuleSet ruleSet) | Удаляет список правил из конфигурационного файла Suricata. На вход принимает список правил, который не может быть пуст. |
| **void** addRule(String path, Rule rule) | Добавляет правило в список правил, который может быть найден по заданному пути. На вход принимает путь к корневой папке Suricata со списком правил и само правило. Оба аргумента не могут быть пустыми. |
| **void** removeRule(String path, Rule rule) | Удаляет правило из списка правил, который может быть найден по заданному пути. На вход принимает путь к корневой папке Suricata со списком правил и само правило. Оба аргумента не могут быть пустыми. |

Согласно спецификации Suricata каждое правило перехвата состоит из действия, заголовка и опций. Пакет com.bmstu.suricata.settings.manager.model содержит 5 классов, необходимых для реализации модели правила:

Таблица 5. Классы пакета com.bmstu.suricata.settings.manager.model и их описание

|  |  |
| --- | --- |
| **Класс** | **Состав и описание** |
| Action | **private** String name – имя действия  **public** Action(String name) – конструктор, задает имя полученное в списке аргументов  **public** String getName() – возвращает имя действия |
| Header | **private** String protocol – протокол правила. Не пустой.  **private** String destination – пункт назначения. Не пустой.  **private** String destinationPort – порт назначения. Не пустой.  **private** String direction – направление. Не пустое.  **private** String source - источник правила. Не пустой.  **private** String sourcePort – порт источника. Не пустой.  **public** String getProtocol() – возвращает протокол.  **public** String getDestinationPort() – возвращает порт назначения.  **public** String getDirection() – возвращает направление правила.  **public** String getSource() – возвращает источник.  **public** String getSourcePort() – возвращает порт источника. |
| Options | **private** Collection<String> values – список опций  **public** Options(Collection<String> values) – конструктор, задает опции полученные в списке аргументов  **public** Collection<String> getValue() – возвращает список опций |
| Rule | **private** Action action – название действия  **private** Header header - заголовок  **private** Options options – опции  **public** Rule(Action action, Header header, Options options) – конструктор, задает параметры полученные в списке аргументов  **public** Action getAction() – возвращает имя действия  **public** Header getHeader() – возвращает заголовок  **public** Options getOptions() – возвращает опции  **public** String toString() – преобразует правило, разбитое на фрагменты в единую строку |
| RuleSet | **private** String name – имя списка правил  **public** RuleSet(String name) – конструктор, задает имя списка, указанное в аргументах.  **public** String getName() – возвращает имя списка правил. |

### Плагин com.bmstu.suricata.settings.manager.client

Данный плагин содержит пакет com.bmstu.suricata.settings.manager.client, который включает в себя 2 класса, используемых для создания правила и добавления правил в конфигурационный файл:

Таблица 6. Классы пакета com.bmstu.suricata.settings.manager.client и их описание

|  |  |
| --- | --- |
| **Класс** | **Состав и описание** |
| SuricataRuleSetSettingsClient | **private** **static** **final** String ***RULE\_NAME*** = "http-events.rules" – задает имя правила.  **private** ISuricataSettingsManager suricataSettingsManager - создает экземпляр интерфейса.  **public** **void** activate(Map<String, Object> properties) – добавляет список правил в экземпляр suricataSettingsManager.  **public** **void** bindSuricataSettingsManager(ISuricataSettingsManager suricataSettingsManager) – инициализирует экземпляр suricataSettingsManager значением, полученным в списке аргументов. |
| SuricataRuleSettingsClient | **private** **static** **final** String ***RULE\_PATH*** = "rules\\custom.rules" – задает путь к конфигурационному файлу с правилами  **private** ISuricataSettingsManager suricataSettingsManager - создает экземпляр интерфейса.  **public** **void** activate(Map<String, Object> properties) – добавляет/удаляет правило в экземпляр suricataSettingsManager.  **public** **void** bindSuricataSettingsManager(ISuricataSettingsManager suricataSettingsManager) - инициализирует экземпляр suricataSettingsManager значением, полученным в списке аргументов.  **private** Rule createRule() – создает необходмое пользователю правило |

### Плагин com.bmstu.suricata.settings.manager.service

Содержит пакет com.bmstu.suricata.settings.manager.service, который включает в себя класс SuricataSettingsManager, реализующий основной функционал разработанного сервиса. Содержит следующие поля и методы:

**private** **static** **final** String ***SURICATA\_ROOT*** = "C:\\Suricata\\" – задает путь к корневой папке сурикаты

**private** **static** **final** String ***PATH\_TO\_SURICATA\_SETTINGS*** = ***SURICATA\_ROOT*** + "suricata.yaml" – задает путь к файлу с настройками сурикаты

**private** **static** **final** String ***COMMENT\_CHAR*** = "#" – задает символ, которым обозначается комментарий

**private** **static** **final** String ***DASH\_CHAR*** = "-" – символ черты

**public** **void** addRuleSet(RuleSet ruleSet) – добавляет список правил в файл. Работает следующим образом: считывается весь файл настроек suricata.yaml, при этом находится строка «rule-files», ее индекс помечается как начальный индекс списка правил, затем ждем, пока метод дойдет до конца файла и не наткнется на строку с длинной равной 0 – индекс этой строки будет концом списка правил. Затем считывается индекс текущей строки, и если она соответствует нашему правилу и закомментирована, то мы ее раскомментируем, иначе допишем наше правило в конец.

**public** **void** removeRuleSet(RuleSet ruleSet) – удаляет список правил из файла. Сначала считываем файл, выделяем начальный и конечный индексы списка правил, и удаляем требуемую строку из файла.

**Public** **void** addRule(String path, Rule rule) - добавляет правило в файл. Так же считываем весь файл, получаем номер нужной строки > 0 и если правило существует и закомменчено, то снимаем комментарий, иначе добавляем правило в конец.

**public** **void** removeRule(String path, Rule rule) – удаляем правило из файла. Считывается файл, получаем номер нужной строки, если индекс строки > 0 и правило не закомменчено, то удаляем его.

**private** **int** getRuleSetsStartIndex(List<String> lines) – позволяет определить начальный индекс списка правил. В цикле прогоняется текст файла и если строка начинается с «rule-files», то ее индекс возвращается.

**private** **int** getRuleSetsEndIndex(List<String> lines, **int** rulesStartLineIndex) – позволяет определить индекс конца списка правил. В цикле прогоняется текст файла, если строка пустая (длина равно нулю), то индекс строки возвращается.

**private** List<String> uncommentRuleSet(RuleSet rule, **int** ruleLineIndex, List<String> lines) – снимает комментарий требуемой строки списка правил, путем замены этой строки (в виде комментария) на наше правило.

**private** List<String> insertRuleSet(RuleSet rule, List<String> lines, **int** rulesEndLineIndex) – вставляет правило в конец списка. Создается массив размером текущий список + 1 строка, в него переносятся существующие правило и в конец добавляется нужное правило.

**private** String createRuleLine(RuleSet rule) – создает строку для списка правил.

**private** **int** getRuleSetLineIndex(RuleSet rule, List<String> lines, **int** rulesStartLineIndex, **int** rulesEndLineIndex) – возвращает индекс искомой строки. В цикле, от начального до конечного индекса, если текст строки совпадает с текстом правила, то индекс этой строки возвращается.

**private** **void** rewriteFile(String filePath, List<String> lines) – осуществляет перезапись в файл.

**private** **boolean** isCommented(String ruleName, String line) – флаг, возвращает true, если строка в начале содержит символ «#».

**private** **int** getRuleLineIndex(List<String> lines, Rule rule) – возвращает индекс строки правила. В цикле, пока встречается символ «#» он обрезается, затем идет проверка, что строка не пустая и содержит правило; в этом случае возвращается индекс строки.

**private** **boolean** containsRule(String line, Rule rule) – флаг, возвращает true, если строка содержит искомое правило.

**private** **boolean** isRuleCommented(String ruleLine) – флаг, возвращает true, если строка начинается с символа «#» (комментарий).

**private** List<String> uncommentRuleLine(List<String> lines, **int** ruleLineIndex) – снимает комментарий с правила. Если индекс строки совпадает с индексом строки правила, убирается символ «#».

**private** List<String> insertRule(List<String> lines, Rule rule) – вставляет созданное пользователем правило в с список правил.

## Диаграмма компонентов

На рисунке 2 представлена диаграмма компонентов нашего проекта. Пользователь, взаимодействуя с методом createRule, задает требуемое правило Rule. Методом activate, используя метод addRule класса SuricataSettingsManager, наше правило добавляется в экземпляр этого класса (при этом в добавлении принимают участие методы insertRule, который непосредственно добавляет правило в наш менеджер, и rewriteFile, который заносит правило в сам файл).

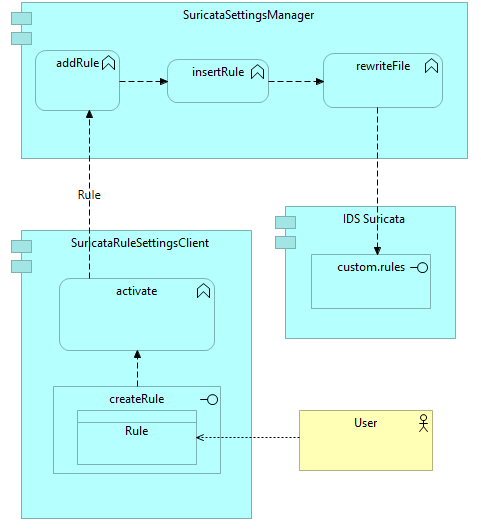


Рисунок 2. Диаграмма компонентов

## Диаграмма классов

На рисунке 3 представлена диаграмма классов нашего проекта. Весь проект состоит из пяти пакетов. Один и пакетов содержит тесты, а другой модель правила и списка правил, все это было описано выше и не отображено на нижней диаграмме.

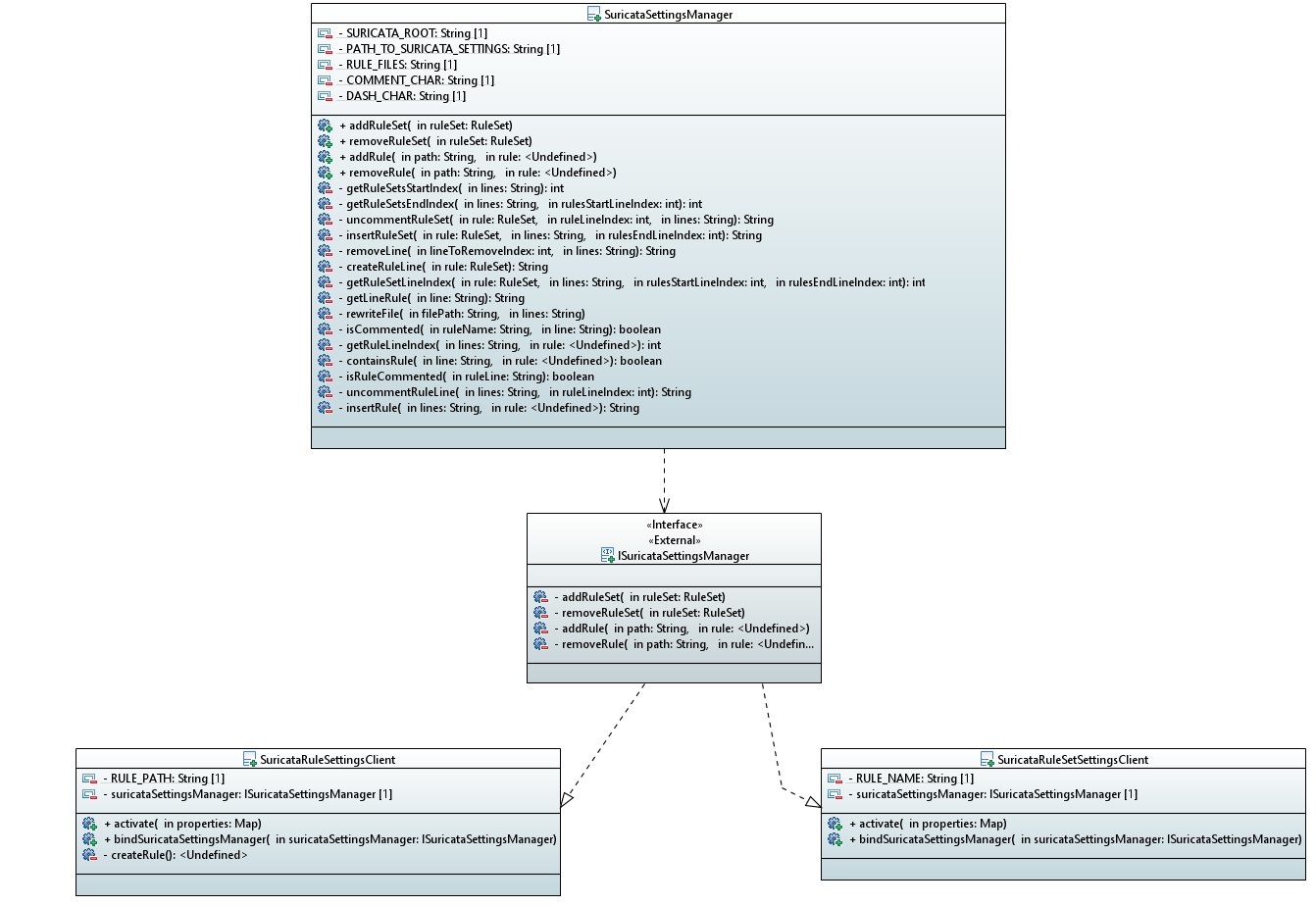


Рисунок 3. Диаграмма классов

# Технологическая часть

## Запуск разработанного приложения

Исходный код проекта доступен в репозитории GitHub [3].

Сервис SuricataSettingsManager взаимодействует с двумя клиентами SuricataRuleSettingsClient и SuricataRuleSetSettingsClient. Запуск приложения осуществляется с помощью конфигурационного файла, который содержит настройки, необходимые для успешного запуска приложения.

## Анализ исходного кода с помощью метрик качества

На рисунке 4 показано соотношение пакетов проекта по их размеру. Видно, что самый большой размер имеет пакет service.

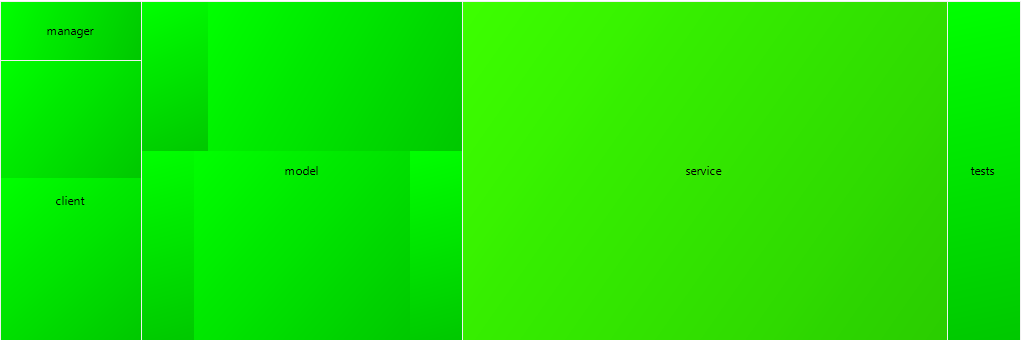


Рисунок 4. Соотношение пакетов по размеру

Далее на рисунке 5 отображен список всех метрик по разделам. Всего имеется четыре раздела:

* метрики количества (Count);
* метрики сложности (Complexity);
* метрики Роберта Мартина (Robert C. Martin);
* метрики Чидамбера-Кемерера (Chidamber & Kermerer).

Первый раздел с метриками количества (Count) содержит следующие метрики:

* количество классов верхнего уровня (Unit);
* среднее число внутренних классов на класс (Classes / Class);
* среднее число методов в классе (Methods / Class);
* среднее число полей в классе (Fields / Class);
* число строчек кода (ELOC);
* число строчек кода на модуль (ELOC / Unit).

Второй раздел с метриками сложности (Complexity) содержит всего три различных метрики:

* средняя циклическая сложность (CC);
* метрика Fat (Fat);
* средняя зависимость компонентов между модулями (ACD - Unit).

Третий раздел с метриками Роберта Мартина содержит следующие метрики:

* нормализованное расстояние от основной последовательности (D);
* абстрактность (A);
* нестабильность (I);
* число афферентных соединений (Ca);
* число эфферентных соединений (Ce).

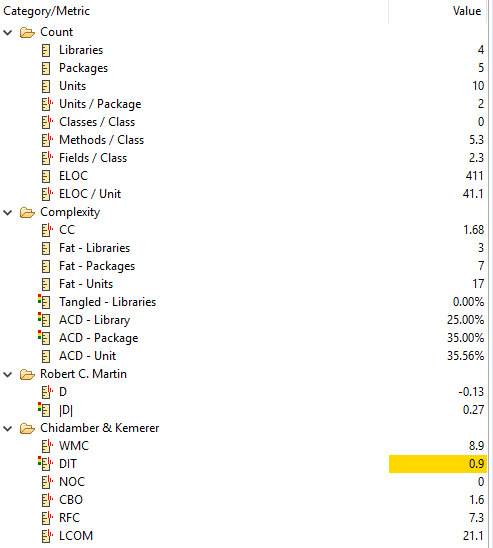


Рисунок 5. Значения метрик

Последний раздел с метриками Чидамбера-Кемерера содержит следующие метрики:

* средняя длина метода на класс (WMC);
* средняя глубина наследования (DIT);
* среднее количество классов-наследников (NOC);
* среднее число соединений класса (CBO);
* среднее число методов, которые потенциально могут быть выполнены в ответ на сообщение, полученное объектом этого класса (RFC);
* отсутствие единства методов (LCOM).

## Анализ зависимостей в коде системы

Рисунок 6 описывает зависимости между пакетами разработанного программного обеспечения:

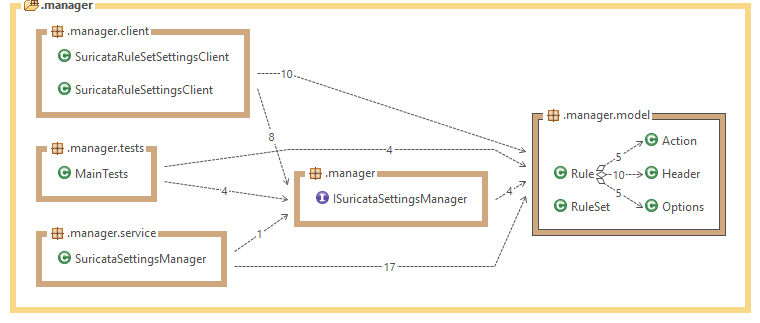


Рисунок 6. Зависимости между пакетами

## Тестирование на корректность работы

Для последующего тестирования разработанного программного обеспечения была поставлена задача написания Junit тестов.

В ходе выполнения курсовой работы было написано 2 теста:

* 1 тест проверяет возможность добавления правила в конфиг-файл. Тест будет пройден, если значение будет не Null
* 2 тест проверяет возможность удаления правила из конфиг-файла.

Связь Junit с OSGI происходит с помощью ServiceTracker. ServiceTracker позволяет получить сервисы из OSGI. Для его использования тест необходимо запускать как plugin test, иначе система не будет подгружать необходимые плагины.

Для проверки используется метод assertNotNull. Если сервис не доступен, то выводится соответствующее сообщение.

Результаты тестирования Junit приведены на рисунке 7.

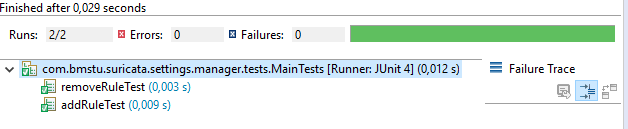


Рисунок 7. Результат работы тестов для проверки на корректность работы

## Оценка покрытия кода.

Покрытие кода – мера, используемая при тестировании программного обеспечения. Она показывает процент исходного кода программы, который был выполнен в процессе тестирования. Цель состоит в том, чтобы получить набор тестов для регрессионного тестирования, тщательно проверяющих весь исходный код. Покрытие близкое к 100% гарантирует безошибочную работу программного обеспечения, однако на практике достижение столь высокого значения требует больших затрат ресурсов, поэтому в большинстве случаев происходит проверка только значимых участков кода, а также проверяется код, который в дальнейшем может быть модифицирован, и тем самым станет вызывать новые ошибки. На рисунке 8 приведен результат оценки покрытия текущего проекта.

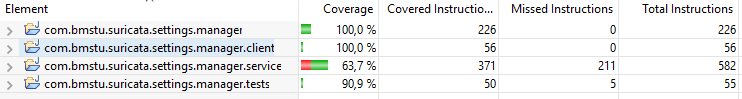


Рисунок 8. Оценка покрытия кода

Зеленые участки – были пройдены при тестировании;

Желтые участки – были частично затронуты при тестировании. Такими участками обычно являются условные операторы;

Красные участки – код не был пройден при тестировании.

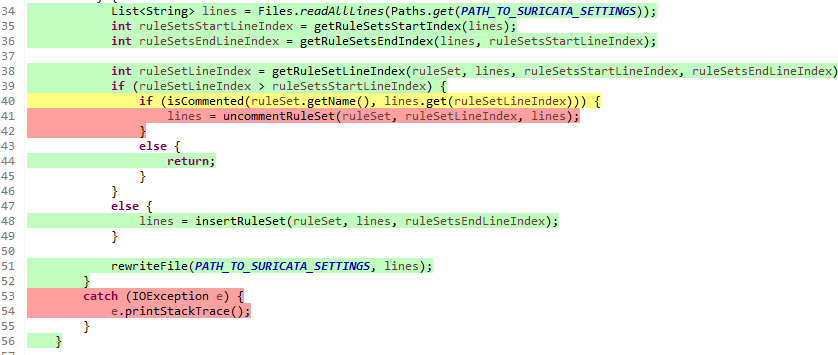


Рисунок 9. Графическое отображение пройденных участков кода.

# Выводы

В ходе выполнения курсовой работы был спроектирован и реализован OSGI компонент настройки правил перехвата пакетов IDS Suricata. Работа была выполнена с помощью технологий OSGI и Junit в среде разработки Eclipse.

Была изучена IDS Suricata и ее преимущества перед другими системами обнаружения вторжений, а также разработана архитектура приложения, структура классов, проведены необходимые тесты работоспособности программного продукта.

Разработанная программа устойчиво выполняет все свои функции.

# Список источников

[1]  Suricata : [Электронный ресурс]. Режим доступа: https://xakep.ru/2015/06/28/suricata-ids-ips-197/

[2] IDS/IPS - системы обнаружения и предотвращения вторжений и хакерских атак: [Электронный ресурс]. Режим доступа: http://www.altell.ru/solutions/by\_technologies/ids/

[3] GitHub – SuricataSettingsManager [Электронный ресурс]. Режим доступа: https://github.com/PanMan1/SuricataSettingsManager/

[4] OSGI modulation – Tutorial : [Электронный ресурс]. Режим доступа: http://www.vogella.com/tutorials/OSGi/article.html

[5] OSGI and Equinox : J. McAffer, 2010. – 328 с.

[6] OSGI services – Tutorial : [Электронный ресурс]. Режим доступа: http://www.vogella.com/tutorials/OSGiServices/article.html