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](#_Toc506218633)

[2 Техническое задание 5](#_Toc506218634)

[3 Теоретическая часть 5](#_Toc506218635)

[3.1 Выявление заинтересованных сторон и их интересов 5](#_Toc506218636)

[3.2 IDS Suricata 5](#_Toc506218637)

[4 Конструкторская часть 7](#_Toc506218638)

[4.1 Выбор технических решений, удовлетворяющих интересам ЗС 7](#_Toc506218639)

[4.2 Структура проекта 8](#_Toc506218640)

[4.2.1 Плагин com.bmstu.suricata.settings.manager 8](#_Toc506218641)

[4.2.2 Плагин com.bmstu.suricata.settings.manager.client 10](#_Toc506218642)

[4.2.3 Плагин com.bmstu.suricata.settings.manager.service 11](#_Toc506218643)

[4.3 Диаграмма компонентов 13](#_Toc506218644)

[4.4 Диаграмма классов 14](#_Toc506218645)

[5 Технологическая часть 15](#_Toc506218646)

[5.1 Запуск разработанного приложения 15](#_Toc506218647)

[5.2 Анализ исходного кода с помощью метрик качества 15](#_Toc506218648)

[5.3 Анализ зависимостей в коде системы 17](#_Toc506218649)

[5.4 Тестирование на корректность работы 18](#_Toc506218650)

[5.5 Оценка покрытия кода. 19](#_Toc506218651)

[6 Выводы 20](#_Toc506218652)

[7 Список источников 21](#_Toc506218653)

# 

# Введение

Любой системный администратор всегда сталкивается с необходимостью анализа трафика как в локальной сети в целом, так и на компьютере конкретного пользователя. Для этой цели существуют системы обнаружения вторжений.

Система обнаружения вторжений – это оборудование или программное обеспечение, которое анализирует сетевой трафик на предмет подозрительной активности и выдает предупреждения при обнаружении такой активности в сети. Обнаружение аномалий и выдача предупреждений являются основными функциями СОВ, однако существуют СОВ, способные предпринимать действия при обнаружении злонамеренной активности. Такие системы могут блокировать трафик, отправленный с подозрительных IP-адресов, прежде чем сети будет нанесен ущерб.

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 . 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

Suricata – это система обнаружения вторжений с открытым исходным кодом, являющаяся результатом разработок, проходящих под руководством OISF. Целью OISF является внедрение новых идей безопасности и технологических инноваций в отрасль обнаружения вторжений. При финансовой поддержке Министерства национальной безопасности США была создана многопоточная альтернатива Snort для защиты сетей от передовых вторжений. Многопоточная архитектура Suricata уникальна, поскольку она поддерживает высокопроизводительные многоядерные и многопроцессорные системы. Основными преимуществами многопоточной системы являются обеспечение высокой скорости анализа сетевого трафика и способность распределять нагрузку IDS на основе приоритетов обработки.

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.

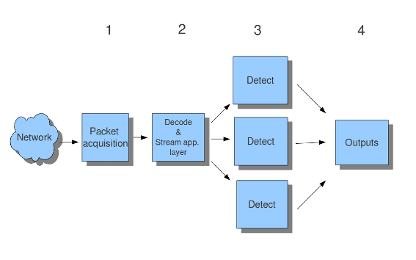


Рисунок 1. Пример работы Suricata

Основные преимущества Suricata:

1. Движок с открытым исходным кодом. Открытый код позволяет сообществу быстрее обнаруживать новые угрозы, нежели это делает одна отдельная организация.
2. Многопоточность: многопоточная архитектура позволяет движку использовать преимущества многоядерных и многопроцессорных архитектур сегодняшних систем.
3. Поддержка IP-Reputation: путем включения подписей и репутаций IP отправителя в движок, Suricata может помечать трафик из недостоверных источников.
4. Автоматическое распознавание протокола. Препроцессоры автоматически идентифицируют протокол, используемый в сетевом потоке, и применяют соответствующие правила, вне зависимости от порта. Автоматическое распознавание так же предотвращает ошибки, совершаемые пользователем.

Suricata была разработана с целью облегчить процесс внедрения системы. Система сопровождается пошаговой документацией и руководством пользователя. Движок Suricata написан на С, с возможностью масштабирования. Хотя Suricata по-прежнему является новым и менее распространенным продуктом по сравнению со Snort, она активно набирает обороты среди всех предприятий и ИТ-пользователей.

# 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 . 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).

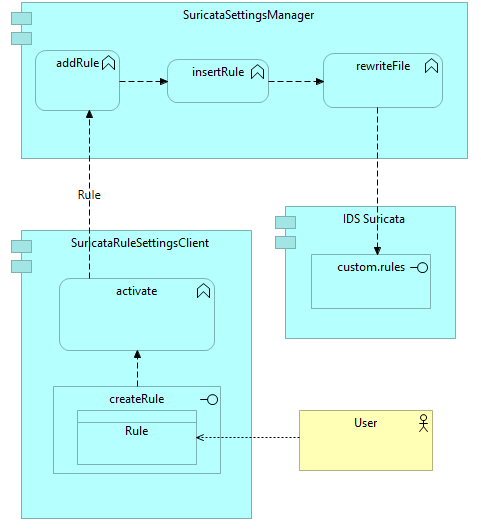


Рисунок 2. Диаграмма компонентов

## 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.

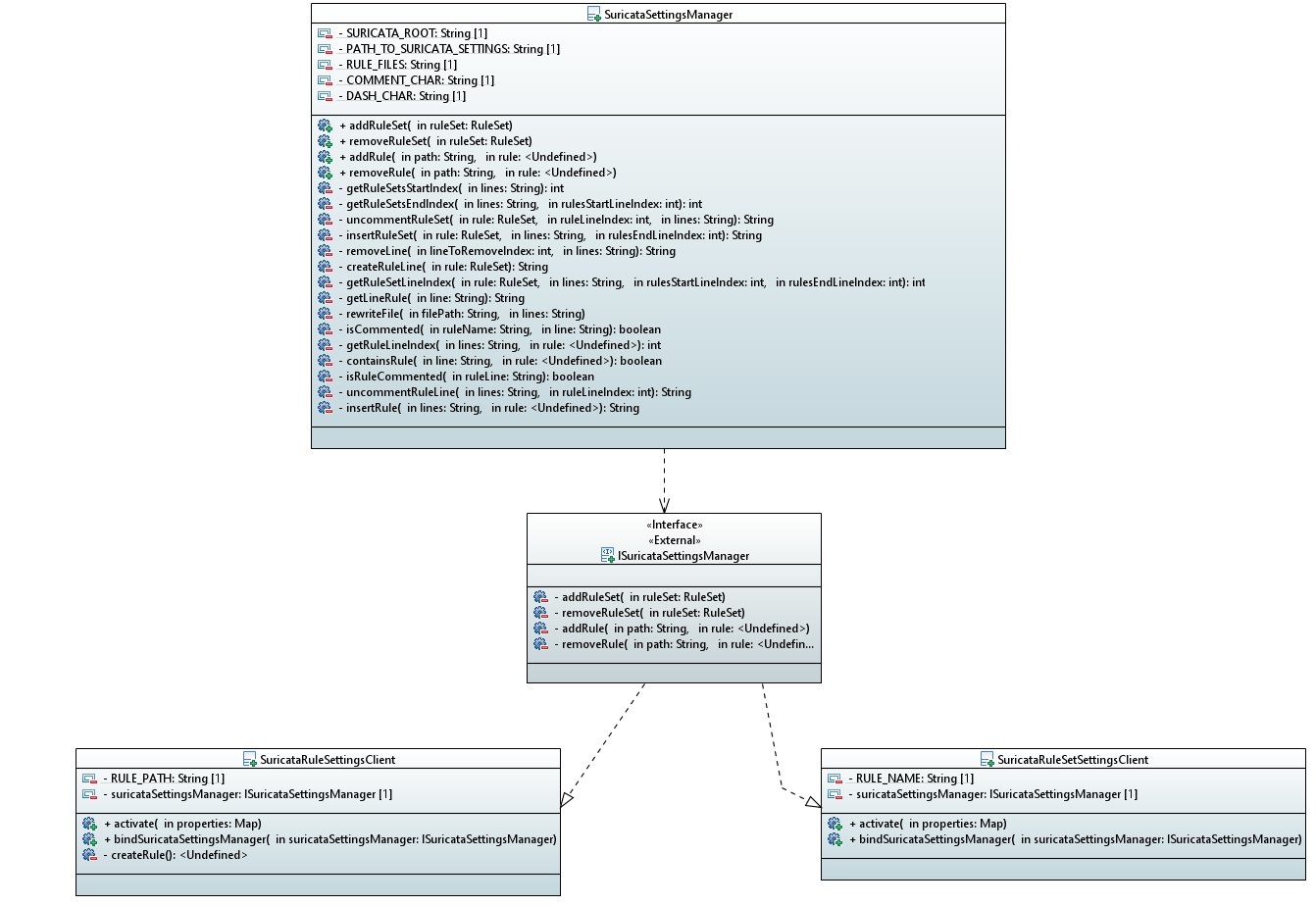


Рисунок 3. Диаграмма классов

# 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.

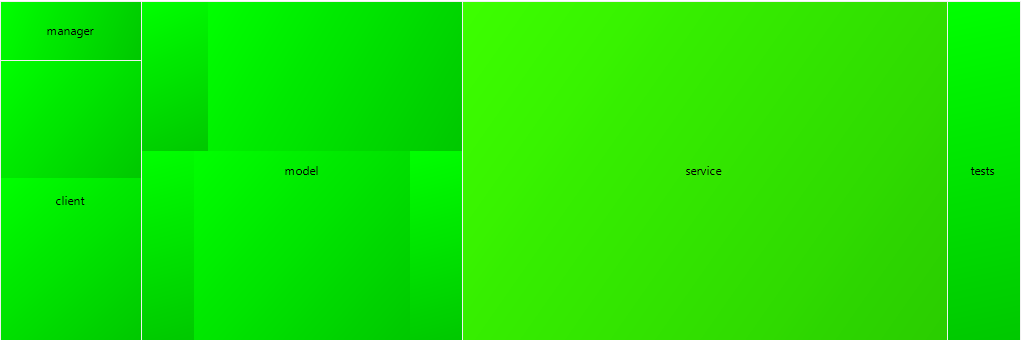


Рисунок 4. Соотношение пакетов по размеру

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).

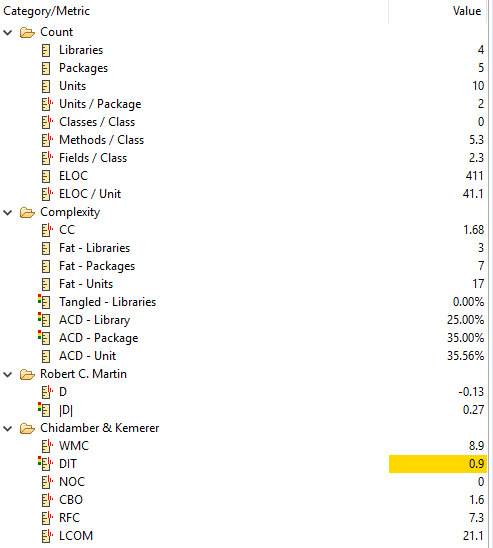


Рисунок 5. Значения метрик

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:

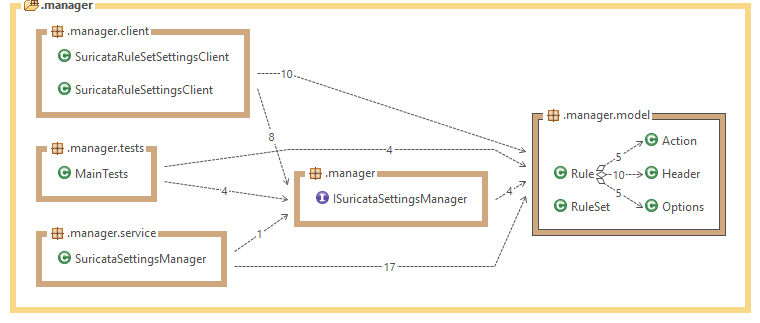


Рисунок 6. Зависимости между пакетами

## 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.

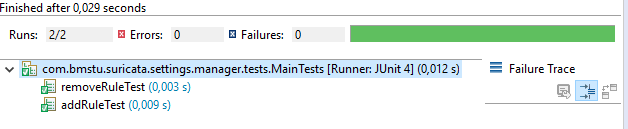


Рисунок 7. Результат работы тестов для проверки на корректность работы

## Оценка покрытия кода.

Покрытие кода – это метрика тестирования программного обеспечения, которая показывает, какие строки исходного кода были протестированы, а какие – нет. Чем больше покрытие, тем выше шанс, что программное обеспечение отработает без ошибок, однако стоит понимать, что высокий процент покрытия требует большего количества усилий и, как следствие, больших затрат ресурсов. Покрытие кода предоставляет важную информацию, позволяющую выявить важные участки кода, на которых стоит сосредоточить свои усилия, так как в дальнейшем эти участки могут быть модифицированы, что, вероятно, приведет возникновению новых ошибок.

Показатели покрытия кода обеспечивают обратную связь, которая позволяет разработчику эффективно фокусироваться на непроверенных областях, а не дублировать усилия.

На рисунке 8 приведен результат оценки покрытия текущего проекта.

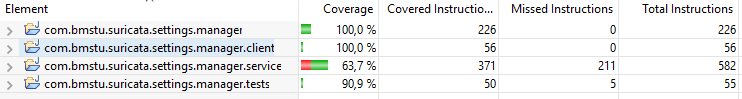


Рисунок . Оценка покрытия кода

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.

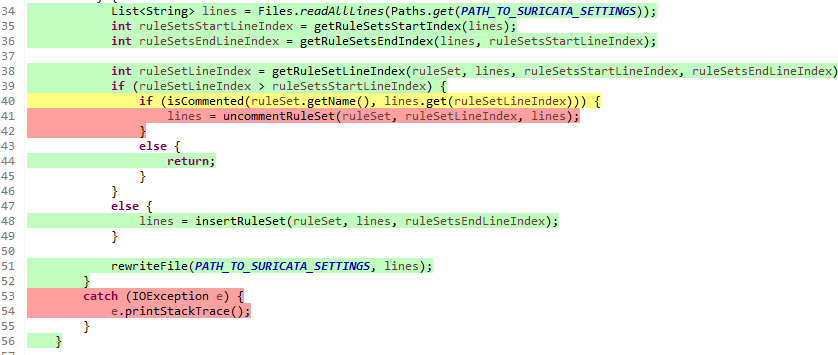


Рисунок 9. Графическое отображение пройденных участков кода

# 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