Министерство науки и высшего образования РФ

Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой
_____О.В. Непомнящий
«___»_____2023 г.

БАКАЛАВРСКАЯ РАБОТА

090301 Информатика и вычислительная техника

Система анализа пограничного трафика на основе глубокого анализа пакетов

Руководитель	_____	_____	доцент, канд. техн. наук	Ф.А. Казаков
	подпись	*дата*	*должность, ученая степень*	
Выпускник	_____	_____		Ю.А. Коптев
	подпись	*дата*		
Нормоконтролёр	_____	_____		Ф.А. Казаков
	подпись	*дата*		

Красноярск 2023

Министерство науки и высшего образования РФ

Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой
_____О.В. Непомнящий
«___»_____2023 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
в форме бакалаврской работы

Студенту Коптеву Юрию Андреевичу
фамилия, имя, отчество

Группа ЗКИ18-07Б Направление (специальность) 09.03.01
номер код

Информатика и вычислительная техника
наименование

Тема выпускной квалификационной работы: Система анализа пограничного трафика на основе глубокого анализа пакетов

Утверждена приказом по университету № 7667/С от 17.05.2023 г.

Руководитель ВКР: Ф.А. Казаков, канд. техн. наук, доцент, директор Центра РиЭМКИВС
инициалы, фамилия, учёная степень, должность, место работы

Исходные данные для ВКР:

1) структурная схема системы;

2) справочный материал.

3)

Перечень разделов ВКР:

1) обзор существующих систем глубокого анализа трафика;

2) архитектура предлагаемой системы;

3) описание функционирования и основные алгоритмы анализа создаваемой системы;

4) результаты тестирования системы.

Перечень графического материала: Презентация Power Point, структурная схема системы.

Руководитель ВКР _____ Ф.А. Казаков
подпись инициалы, фамилия

Задание принял к исполнению _____ Ю.А. Коптев
подпись инициалы, фамилия

«22» 12 2022 г.
дата

РЕФЕРАТ

Выпускная квалификационная работа по теме «Система анализа пограничного трафика на основе глубокого анализа пакетов» содержит 64 страницы текстового документа, 36 рисунков, 13 использованных источников, 10 слайдов презентации.

Цель – создание автоматизированного рабочего места специалиста по информационной безопасности на основе виртуального сервера.

Для достижения поставленных целей необходимо выполнить следующие задачи:

– создание виртуального сервера на базе гипервизора Hyper-V:

– установка и настройка системы обнаружения вторжений Snort;

– установка и настройка генератора правил безопасности для Snort;

– проведение тестов системы посредством эмуляции атак;

– создание и модификация правил безопасности для предотвращения проникновения.

В результате выпускной квалификационной работы произведен анализ предметной области, по результатам которого для реализации системы выбрано следующее программное обеспечение:

– средой развертывания виртуального сервера был выбран гипервизор Hyper-v;

– программным обеспечением для проведения глубокого анализа пакетов было выбрано программное обеспечение Snort и библиотека захвата пакетов Npcap;

– генератор правил на основе web-приложения для программного обеспечения Snort, развернутый при помощи технологии Docker-контейнера Snorpy;

– произведена установка и настройка выбранного программного обеспечения;

– произведены тестовые запуски системы, просканированы эталонные файлы, изучены полученные log-файлы;

– созданы новые правила, для более тонкого тестирования системы.

Полученная в результате работы система на базе виртуального сервера может служить в качестве системы оповещения об атаках, системой мониторинга, системой дополнительного управления системы типа «Firewall», а также система обладает высокой степенью портативности и отличается малым временем развертывания.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Объектом разработки данной выпускной квалификационной работы выступает система анализа пограничного траффика на основе глубокого анализа пакетов (DPI).

Задачи работы:

– создание виртуального сервера на базе гипервизора Hyper-V;

– установка и настройка системы обнаружения вторжений Snort;

– установка и настройка генератора правил безопасности для Snort;

– проведение тестов системы посредством эмуляции атак;

– создание и модификация правил безопасности для предотвращения проникновения.

В первом разделе выпускной квалификационной работы мы рассматривается принцип действия системы глубокого анализа трафика, описываются существующие решения и плюсы и минусы каждого из них. Дополнительно произведен краткий обзор использующихся технологий, взаимодействующих с системами глубокого анализа пакетов.

Во втором разделе выпускной квалификационной работы произведен обзор выбранных решений, подробное описание используемого программного обеспечения и подробный обзор действий, необходимых для запуска и настройки системы.

Третий раздел посвящен описанию функционирования основных элементов системы глубокого анализа пакетов. Обзор принципов работы элементов системы и описание синтаксиса ключевых команд для работы с используемым программным обеспечением.

В четвертом разделе выпускной квалификационной работы произведен запуск системы, тестирование её в различных режимах. Представлены результаты произведенных тестов с использованием эталонного файла.

Актуальность данной работы обусловлена необходимостью качественного роста информационной безопасности в связи с происходящими мировыми

событиями. Например, из-за эпидемиологической ситуации 2019-2021 года в мире многократно возросло количество сотрудников на удаленном режиме работы.

На текущий момент времени сотрудники сферы информационной безопасности не в состоянии обеспечить достаточную степень защищенности корпоративных данных на домашних компьютерах работников, что закономерно ведет за собой количественное увеличение попыток нелегитимного проникновения во внутреннюю сеть компании и количество утечек корпоративных данных.

Одним из вариантов решения проблем с атаками на корпоративную сеть с домашних компьютеров сотрудников могут служить межсетевые экраны, использующие технологии DPI, которые помимо функции сканирования пакетов также позволяют собирать статистику.

На основании этой статистики специалист по информационной безопасности способен обнаруживать нехарактерный для пользователя трафик и предотвращать попытки проникновения.

1 Анализ предметной области

1.1 Обзор технологии глубокого анализа пакетов

Технология глубокого анализа пакетов (далее DPI) представляет собой систему анализирующую пакеты трафика на верхних уровнях модели OSI.

Сетевая модель OSI(Open Systems Interconnection) [6] – модель сетевого стека, разработанная в конце 1970-х годов. Модель классифицирует уровни взаимодействия сетевых протоколов и оборудования, уровни классификации согласно модели OSI представлены на рисунке 1.

Данные	**Прикладной** (доступ к сетевым службам)	Осуществляет взаимодействие между пользователем и сетью. Взаимодействует с приложениями на стороне клиента.	HTTP, FTP, Telnet, SSH, SNMP
Данные	**Представления** (представление и кодирование данных)	Осуществляет преобразование данных в нужную форму, шифрование/кодирование, сжатие.	MIME, SSL
Данные	**Сеансовый** (управление сеансом связи)	Управляет созданием/поддержанием/завершением сеанса связи.	L2TP, RTCP
Блоки	**Транспортный** (безопасное и надежное соединение точка-точка)	Предназначен для доставки данных без ошибок, потерь и дублирования в той последовательности, как они были переданы. Выполняет сквозной контроль передачи данных от отправителя до получателя.	TCP, UDP
Пакеты	**Сетевой** определение пути и IP (логическая адресация)	Его основными задачами являются маршрутизация – определение оптимального пути передачи данных, логическая адресация узлов. Кроме того, на этот уровень могут быть возложены задачи по поиску неполадок в сети (протокол ICMP). Сетевой уровень работает с пакетами.	IP, ICMP, IGMP, BGP, OSPF
Кадры	**Канальный** MAC и LLC (физическая адресация)	Отвечает за доступ к среде передачи, исправление ошибок, надежную передачу данных. *На приеме* полученные с физического уровня данные упаковываются в кадры после чего проверяется их целостность. Если ошибок нет, то данные передаются на сетевой уровень. Если ошибки есть, то кадр отбрасывается и формируется запрос на повторную передачу. Канальный уровень подразделяется на два подуровня: **MAC (Media Access Control)** и **LLC (Locical Link Control)**. MAC регулирует доступ к разделяемой физической среде. LLC обеспечивает обслуживание сетевого уровня. На канальном уровне работают коммутаторы.	IEEE 802.3, IEEE 802.11, PPP, DHCP, ARP
Биты	**Физический** (кабель, сигналы, бинарная передача данных)	Определяет вид среды передачи данных, физические и электрические характеристики интерфейсов, вид сигнала. Этот уровень имеет дело с битами информации.	IEEE 802.11, ISDN

Рисунок 1 – Уровни модели OSI

Модель OSI разработана для:

– согласования работы устройств разработанных разными производителями;

– обеспечения взаимодействия сетей, которые используют разные среды распространения сигнала.

Модель построена по иерархическому признаку, в котором каждый из уровней предоставляет сервис уровню вышестоящему и пользуется сервисом нижестоящего уровня. Обработка данных начинается с седьмого (прикладного)

уровня, после чего данные, проходя через все уровни модели попадают на физический уровень и отправляются в канал связи. На принимающей стороне данные принятые на физическом уровне обрабатываются в обратном порядке.

Модель OSI вводит два основополагающих понятия – «Протокол» и «Интерфейс»:

– протокол – набор правил, на основании которых уровни различных открытых систем взаимодействуют между собой;

– интерфейс – совокупность методов и средств, по которым взаимодействуют элементы открытых систем.

Еще одним немаловажным термином, позволяющим понять, каким образом пакеты данных подготавливаются к передаче, является Инкапсуляция [8]. Инкапсуляция – это метод, при помощи которого происходит упаковка пакетов данных, во время которой независимые друг от друга заголовки пакетов абстрагируются от заголовков уровней нижестоящих, путем их включения в уровни вышестоящие (Рисунок 2).

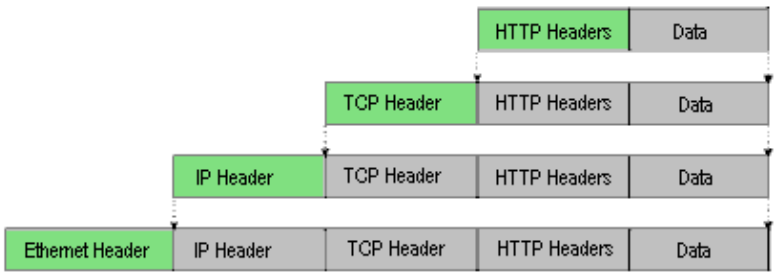

Рисунок 2 – Пример инкапсулирования заголовка

Рассмотрим конкретный пример. Допустим, мы хотим попасть с персонального компьютера на сайт. Алгоритм действия компьютера будет выглядеть следующим образом:

– подготовка http-запроса на получение ресурсов веб-сервера, на котором хранится необходимая нам страница сайта;

– на прикладном уровне к данным браузера добавляется HTTP-заголовок;

– на транспортном уровне к пакету прибавляется TCP-заголовок, содержащий в себе номера портов отправителя и получателя (Для HTTP это будет порт 80);

– на сетевом уровне будет сформирован IP- Заголовок, который содержит в себе IP -адрес отправителя и IP -адрес получателя;

– в момент перед передачей на канальном уровне добавится Ethernet-заголовок, который содержит в себе MAC-адреса (физические адреса) отправителя и получателя.

По окончанию всех этих действий пакет в виде битов информации отправляется по физическому уровню сети. Принимающая сторона производит вышеуказанные процедуры в обратном порядке. Web-сервер будет проверять соответствующий заголовок на каждом уровне. В случае если проверка пройдет удачно, то заголовок будет отброшен и пакет перейдет дальше по уровню, в случае если проверка будет провалена, то весь пакет будет отброшен.

На основании вышеизложенных данных мы можем более углубленно рассмотреть системы DPI их функционал и применение.

Системы DPI [3] помимо уже изложенных функций способны осуществлять поведенческий анализ трафика. Данный анализ позволяет распознать приложения, которые не используют для обмена данными известные заголовки и структуры данных. Один из ярких примеров таких приложений может выступать приложение Bittorrent. Для идентификации подобных приложений используется анализ последовательностей пакетов, которые обладают одинаковыми признаками, такими как пары «порт источника – порт назначения», размеры пакетов, частота открытия новых сессий и т.д., по эвристическим (поведенческим) моделям, соответствующим таким приложениям.

Очевидным является тот факт, что эвристические модели создают разработчики DPI-систем, а следственно каждая система обладает своим набором поведенческих шаблонов для анализа, а значит и точность детектирования также разнится.

Одними из самых широко известных производителей подобных систем являются:

– Allot Communications;

– Procera Networks;

– Cisco;

– Sandvine.

Всё большей популярностью начинают пользоваться решения с интегрированными в маршрутизаторы DPI-системы, к таким решениям прибегают такие компании как Cisco, Juniper, Ericsson и некоторые другие. Такое решение принято считать компромиссным, к сожалению, оно не позволяет получить весь спектр возможностей доступный полноценным, иначе говоря, Standalone DPI-системам. Однако для большинства задач хватает и таких решений.

Важная отличительная способность DPI-систем это возможность аналитики трафика за счет сбора различной статистики с разбиением её по приложениям, тарифным планам, по регионам, по типам абонентских устройств, что может быть очень актуально для компаний предоставляющих сотовую, спутниковую, либо же проводную связь.

Системы DPI, как правило, устанавливаются на границе сетей, в разрыв существующих линий связи внешних интерфейсов пограничных маршрутизаторов (Рисунок 3). Благодаря чему весь трафик который покидает или поступает в контролируемую сеть поступает через DPI-систему, а следственно система может производить мониторинг и контроль трафика исходя из заранее прописанных правил. Для решения специфических задач можно установить систему и ниже по сети, ближе к конечному пользователю. Это может быть полезно, например, для контроля внутренних каналов сети.

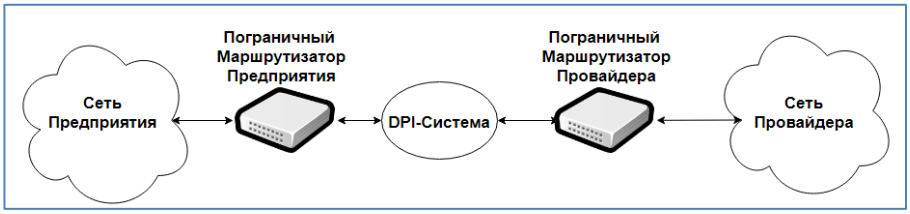

Рисунок 3 – Схема пограничного соединения

1.2 Основные проблемы DPI-систем

Основной проблемой всех существующих на текущий момент DPI-систем является необходимость однозначного определения того или иного потока данных к одному из приложений использующих сеть. Это необходимо для того, что бы система мониторинга могла распознать оба направления потока трафика, как от приложения к хосту, так и от хоста к приложению. Иначе говоря, входящий и исходящий трафик в пределах одного потока должны пройти через одно и то же устройство. Если же система видит только одно направление потока в рамках сессии, то она не имеет возможности сопоставить данный поток с какой-либо известной категорией трафика и следующими из этого проблемами.

В связи с этим, когда поднимается вопрос контроля линий связи внешних интерфейсов пограничных маршрутизаторов, возникает проблема контроля асимметричного трафика, который в крупных сетях являет собой регулярное явление, нежели исключение. Различные разработчики решают эту проблему разными путями [13]:

– компания Cisco использует половину сессии, пытаясь по ней определить тип сетевого соединения. При данной методике, очевидно, страдает точность определения приложений, в особенности тех, которые для однозначного определения требуют анализ поведенческих моделей. Также в данном способе появляются ограничения, накладываемые на варианты управления таким трафиком;

– Sandvine для решения проблемы с асимметрией трафика используют инкапсуляцию. Они инкапсулируют асимметричный трафик в broadcast-фреймы и пересылают его по всем устройствам DPI-системы в домене. В итоге данной сессии устройство – отправитель запроса сможет обнаружить обе части асимметричного трафика, на основании полученных данных и будет осуществляться анализ и контроль трафика. Недостатком данной схемы можно однозначно назвать серьезные требования к пропускной способности сети, которая

должна быть тем больше, чем большее количество асимметричного трафика присутствует в сети.

Procera И Allot для решения этой проблемы использовали одно из лучших решений на рынке. Решение является развитием идеи компании Sandvine, но вместо инкапсулирования асимметричного трафика и отправки его broadcast-фреймом разработчики пересылают явно характеризующие асимметричный трафик метаданные. За счет подобной оптимизации требования к каналам связи снижаются вплоть до 95% относительно реализации Sandvine.

Вторая важная проблема DPI-систем – это частота обновления файлов сигнатур, на основании которых проводится анализ трафика. Если некоторые из поставщиков DPI-систем делают регулярные обновления файлов-сигнатур и обновляют их раз в неделю, то другие поставщики подобных услуг могут обновлять эти файлы раз в квартал. Разумеется критические обновления выпускаются всеми поставщиками в примерно одно и то же время. Чаще всего поставщики положительно относятся к пожеланиям заказчиков добавить какой-либо протокол в список поддерживаемых и всячески помогают в этом. Следует обратить внимание, что в большинстве стран существуют свои локальные продукты, пользующиеся большой популярностью, например в России такой программой может быть «Mail.ru агент».

1.3 Применение DPI-систем

Во время эксплуатации DPI-системы оператор имеет возможность контролировать утилизацию подключенных через DPI каналов на уровне приложений. До внедрения подобных систем задачи реализации QoS (Quality of Service) решались средствами построения очередей на основании маркировки трафика служебными битами в IP-заголовках [2]. Это позволяло выделить наиболее приоритетный трафик, такие как VPN-сервисы, службы IPTV, службы SIP и подобные им. Приоритетному трафику гарантировалась определенная, предусмотренная заранее, пропускная способность в любой момент времени.

Остальной же трафик фактически оставался бесконтрольным, что давало приложениям по типу Bittorrent забрать себе весь свободный канал в ущерб других приложений. С использованием DPI-систем у контролирующего звена управления сети появилась возможность ограничивать подобные приложения, отводя им меньшую часть трафика в загруженные часы и возвращая им доступ в часы простоя сети.

Другим типичным примером может быть работа с SIP у некоторых мобильных операторов. До внедрения технологий DPI провайдеры либо полностью блокировали подобный трафик, либо выделяли ему минимально возможный канал, что вело за собой соответствующую деградацию качества связи.

Способы контроля сервисов при помощи технологии DPI базируются на двух основных подходах – per-service и per-subscriber. В первом случае конкретному приложению выделяется персональная полоса трафика. Во втором – привязка приложения к выделенной полосе трафика осуществляется для конкретного пользователя или группы пользователей независимо от других [1].

Еще одна из функций, особо актуальная для компаний предоставляющих доступ в сеть интернет или мобильных операторов. Эта функция не относится напрямую к сфере информационной безопасности, но тем не менее мы должны о ней упомянуть. Эта функция – маркетинг. DPI-системы позволяют провайдерам более тесно взаимодействовать с пользователями, привлекать дополнительную аудиторию и удерживать клиентов. Благодаря DPI-системам можно:

– разделять по сегментам клиентскую базу, предлагать для каждого сегмента таргетированные услуги и тарифы;

– бороться с несанкционированной раздачей доступа в глобальную сеть;

– гибко настраивать предложения под каждого пользователя.

Также при помощи DPI можно повышать качество предоставляемого подключения к сети интернет

– приоритизировать и разграничивать трафик;

– организовывать равномерное прохождение трафика;

– повышать качество и скорость соединения;

– производить профилактику перегрузок сети;

DPI-системы позволяют блокировать подозрительные и запрещенные сайты, например из реестра запрещенных сайтов Роскомнадзора. Важно учитывать, что для выполнения этой функции к DPI-системе будут представлены крайне высокие требования по производительности: они должны будут анализировать огромные массивы данных в режиме реального времени.

1.4 Сравнение предлагаемого программного обеспечения

На момент разработки выпускной квалификационной работы лидерами на рынке подобного программного обеспечения выступают системы Snort и Suricata. Обе системы представляют собой системы предотвращения проникновений распространяемых на основе лицензии GPLv2, обладают открытым исходным кодом [9].

Хронологически первой системой была разработана Snort, это является одновременно и плюсом, и минусом системы.

Плюсы:

– ввиду более длительного срока жизни как программного продукта работает более стабильно, большая часть ошибок в коде программы уже исправлены;

– более обширное общество пользователей;

– большее количество руководств и литературы по данному программному обеспечению.

Минусы:

– устаревшая среда разработки (Ядро программы исполнено на языке C++);

– невозможность эффективного использования многопоточных вычислений.

Система Suricata была разработана людьми, отделившимися от компании-разработчика Snort, и следственно хронологически существует меньше, что также несет в себе как положительные, так и отрицательные стороны.

Плюсы:

– стремительно развивающееся программное обеспечение;

– благодаря новому ядру отлично взаимодействует с многопоточными вычислительными системами;

– расширенный функционал относительно Snort.

Минусы:

– разработчики пока не исправили все ошибки, произошедшие во время разработки программного обеспечения;

– меньшая база пользователей, следственно меньшая обратная связь с разработчиком;

– малое количество литературы и руководств по работе с программным обеспечением.

Сравнив и проанализировав плюсы и минусы представленного программного обеспечения, остановимся на программном обеспечении Snort. Программное обеспечение Snort считается более стабильной системой, работой над которой занимается большее количество разработчиков.

2 Архитектура предлагаемой системы

2.1 Подготовка виртуального сервера

В данной выпускной квалификационной работе мы рассматриваем процесс создания системы анализа пограничного траффика на основе глубокого анализа пакетов.

Для создания системы нам потребуется развернуть виртуальный сервер, на котором впоследствии будут выполняться все необходимые процедуры. Средой виртуализации в нашем случае выступит среда Hyper-V, производства компании Microsoft [5]. Hyper-V – это система аппаратной виртуализации для систем на базе 64-битной архитектуры. Hyper-V разработана на основе гипервизора. Удобство Hyper-V заключается в легкости установки, для развертывания оболочки Hyper-V на компьютере с установленной операционной системой Windows 10 необходимо открыть системное окно «Включение или отключение компонентов Windows». Далее следует найти пункт «Hyper-V» и установить служебный флажок (Рисунок 4).

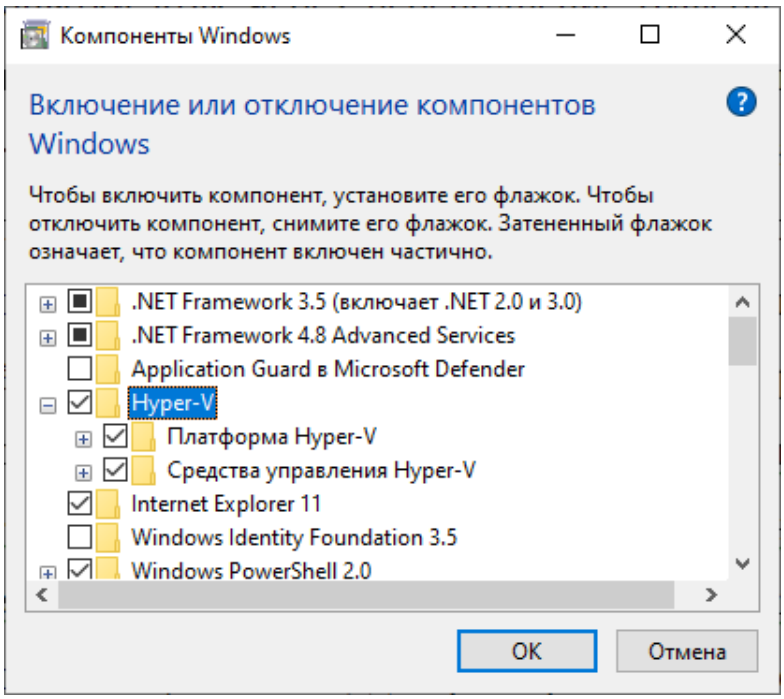

Рисунок 4 – Окно «Компоненты Windows»

После того как операционная система подготовит необходимый пакет установщика, появится системное сообщение о необходимости перезагрузки, после перезагрузки оболочка Hyper-V уже будет развернута, и готова к работе (Рисунок 5)

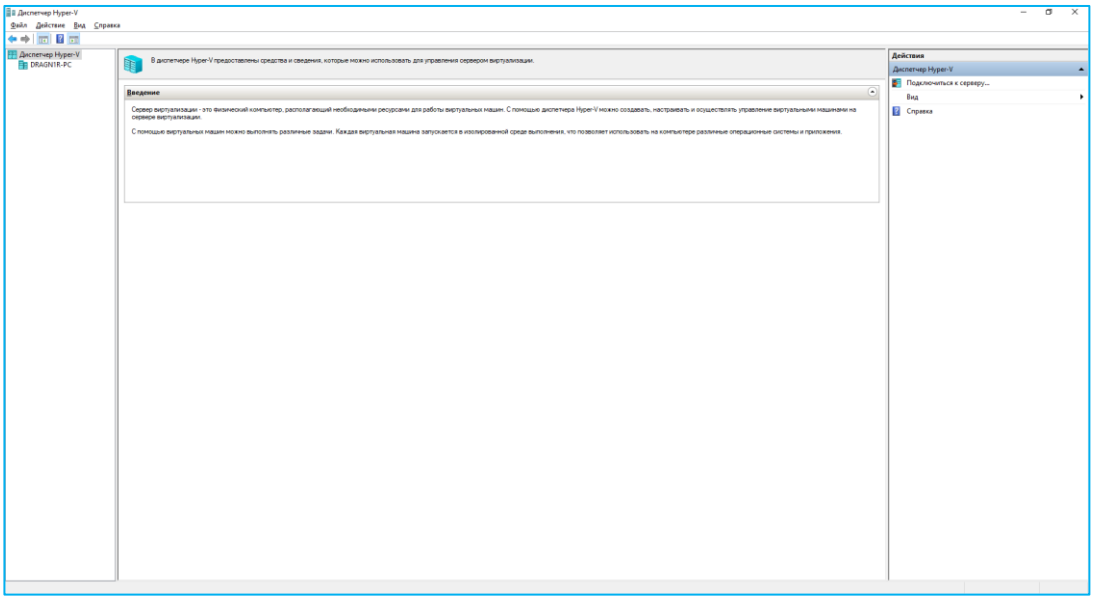

Рисунок 5 – Оболочка Hyper-V

По окончанию всех вышеописанных действий нам необходимо создать новый виртуальный сервер (Рисунок 6).

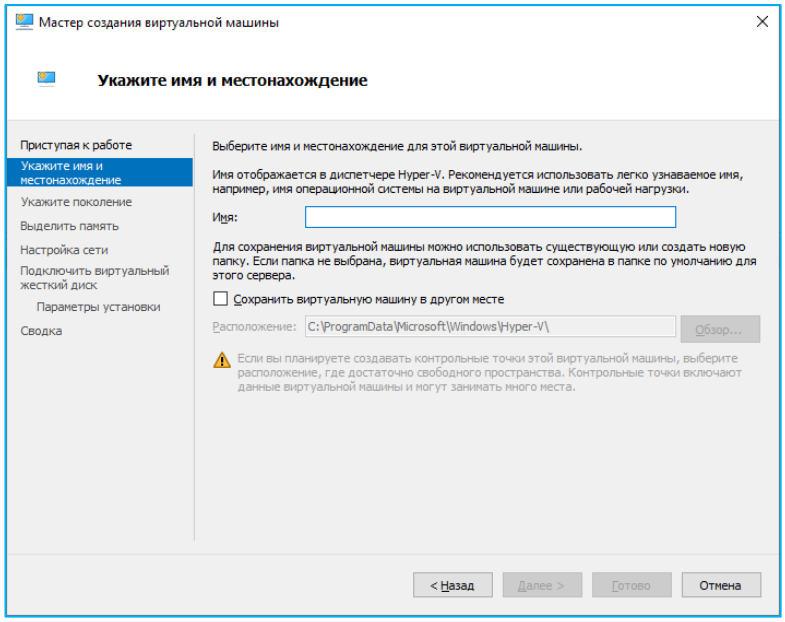

Рисунок 6 – Мастер создания виртуальной машины в среде Hyper-V

2.2 Подготовка программного обеспечения

После развертывания виртуального сервера, необходимо подготовить его к установке программного обеспечения Snort. Для этого необходимо установить операционную систему. Мною была выбрана операционная система Windows 10 от компании Microsoft.

Закончив установку операционной системы и произведя первичную настройку рабочей среды, следует установить пакет Microsoft Visual C++ Redistributable для операционных систем под управлением 64-битных архитектур, версии 14.34 или старше. Он необходим для корректной работы программного обеспечения, ядро которой было написано на языке C++, которое мы будем использовать в дальнейшем.

Анализ пограничного трафика на основе глубокого анализа пакетов осуществляется при помощи программного обеспечения на базе приложения Snort. Snort – это свободно распространяемая система предотвращения и обнаружения вторжений с открытым исходным кодом.

Система Snort способна в реальном времени выполнять регистрацию пакетов, а также осуществлять анализ траффика проходящего через систему.

Система Snort (Рисунок 7) изначально разрабатывалась Мартином Рёшем, далее развивалась и поддерживалась компанией, основанной первым разработчиком программы – Sourcefire, которая была поглощена группой компаний Cisco в 2013 году [10].

Программа Snort осуществляет ведение протоколов, поиск содержимого, анализ, а также активно применяется для активного блокирования или же пассивного обнаружения нападений, попыток зондирования. Примером таких атак могут быть попытки атак через переполнение буфера, сканирование портов в скрытом режиме, попытки атак на web-приложения, зондирование SMB протокола, и даже попытки определения операционной системы на защищаемом устройстве или устройствах. Основное назначение программы Snort – это предотвращение сетевых атак и/или блокирование попыток проникновения.

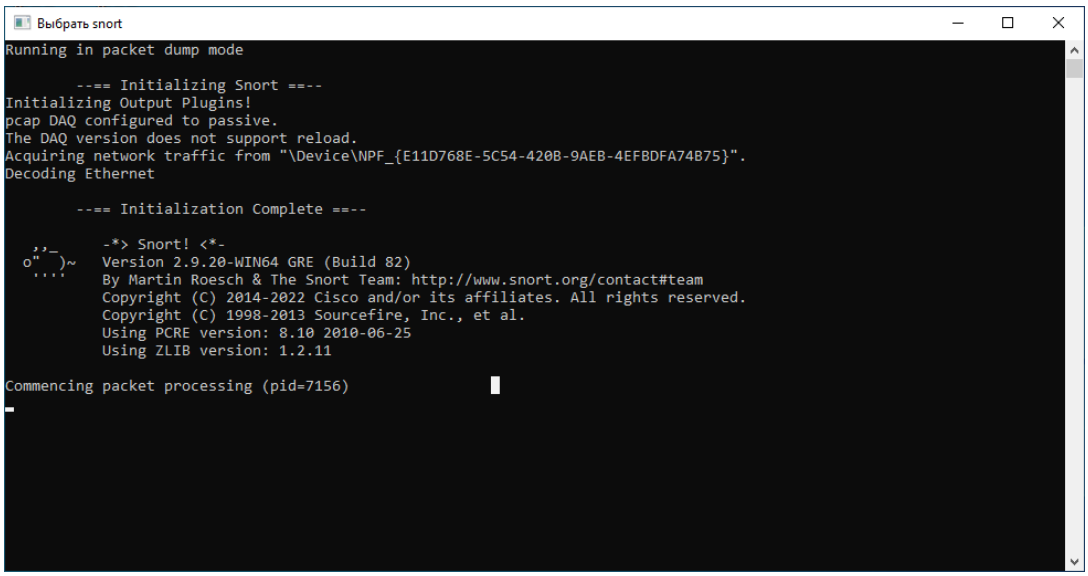

Рисунок 7 – Рабочее окно программы Snort

Система Snort распространяется под лицензией GPLv2, а ядро проекта разработано на языке C++. Данный тип лицензии подразумевает, что автор или коллектив авторов передает программное обеспечение в общественную собственность.

Для начала работы с программным обеспечением Snort необходимо произвести инсталляцию продукта. Пакет установки находится в открытом доступе и расположен на сайте в разделе Downloads. После скачивания пакета установки производим установку программного обеспечения (Рисунок 8).

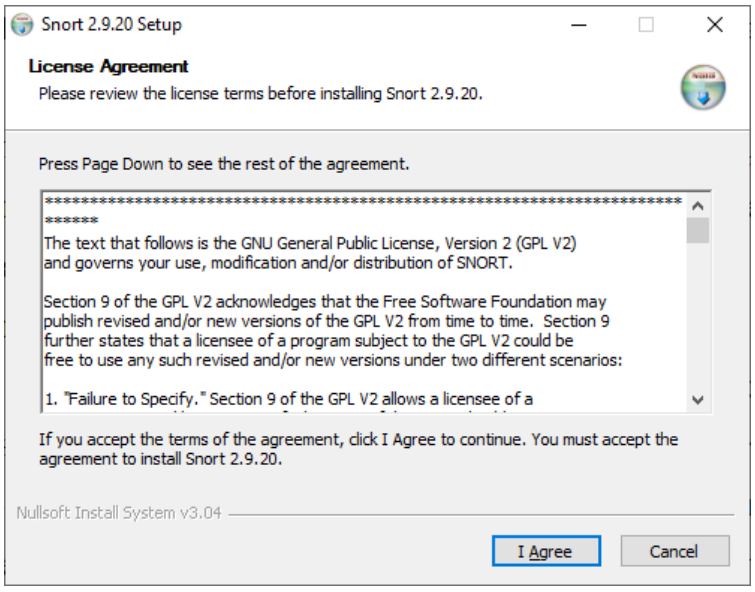

Рисунок 8 – Окно установки ПО Snort

В процессе установки необходимо указать путь установки, в котором в дальнейшем будут располагаться установленные файлы программы. В конце процесса установки программного обеспечения система уведомит (Рисунок 9) о необходимости установки дополнительного вспомогательного программного обеспечения для программы Snort.

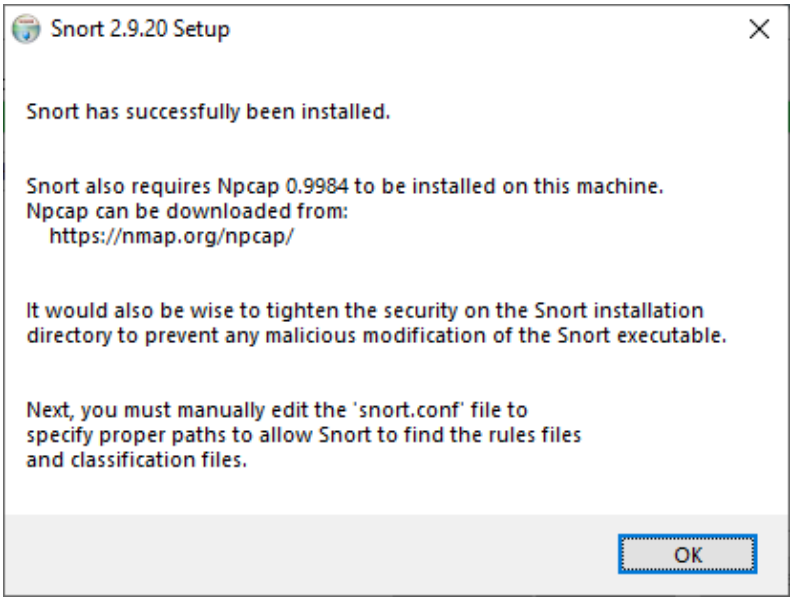

Рисунок 9 – Установщик информирует о необходимости установки дополнительного ПО

Npcap – это библиотека для анализа пакетов проекта Nmap для семейства операционных систем Windows [12]. Npcap является программным инструментом для взаимодействия с драйвером сетевого интерфейса. Функционал программы Npcap подразумевает обеспечение доступа к захвату и передачи сетевых пакетов в обход стека протоколов. Благодаря Npcap удается оперативно выявлять сетевые проблемы и заниматься блокированием атак, цель которых получить нелегитимный доступ к интерфейсу сетевой карты.

Большинство сетевых программ используют так называемые сокеты для получения доступа к сетевым ресурсам, это предоставляет программам доступ на низком уровне, но уже обработанным операционной системой. Некоторые приложения требуют данные, которые до этого не были обработаны операционной системой. В данном конкретном случае Npcap позволит программному

обеспечению Snort беспрепятственно и своевременно получать информацию непосредственно с виртуальной сетевой карты виртуального сервера. Произведем установку программы (Рисунок 10).

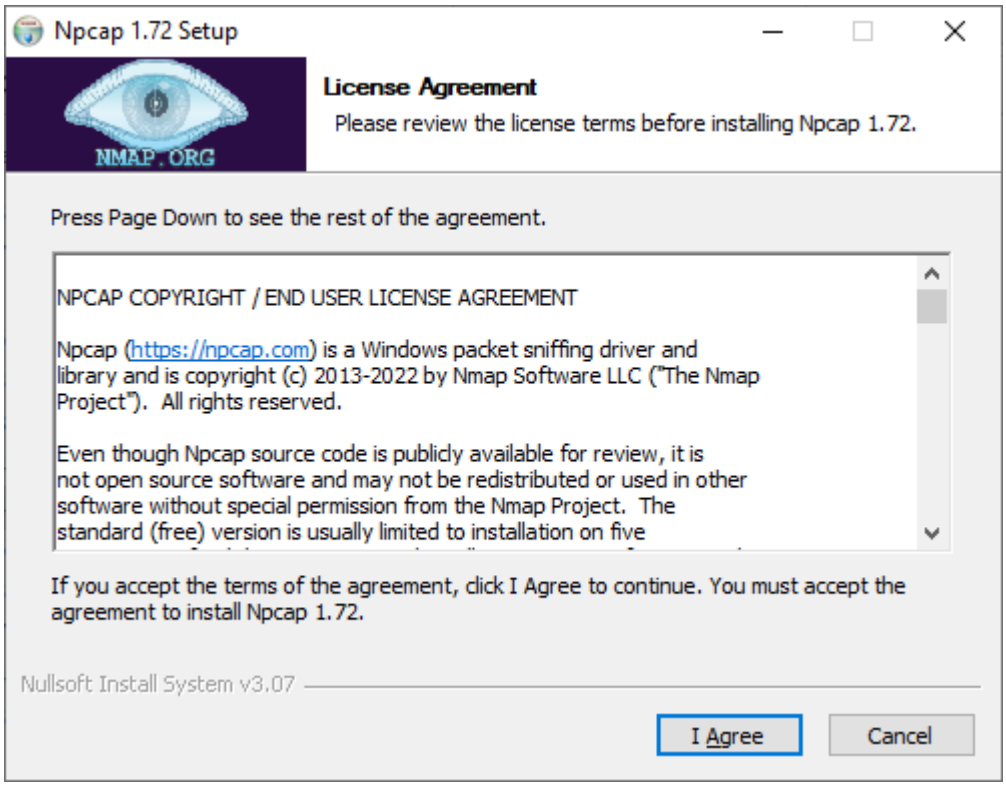

Рисунок 10 – Окно установщика Npcap

После установки операционной системы, вспомогательных библиотек, основного и вспомогательного программного обеспечения следует приступить к настройке конфигурационных файлов программы Snort.

2.3 Настройка программного обеспечения Snort

Архитектура программы Snort имеет в своем составе три основных компонента, опишем их:

– дешифратор пакетов – подготавливает пакеты перехваченные программой в форму типа данных, в последствие обрабатываемые механизмом обнаружения в понятном ему виде. Дешифратор пакетов позволяет регистрировать пакеты типа Ethernet, SLIP и PPP пакеты;

– механизм обнаружения – занимается анализом и обработкой пакетов представленных ему Дешифратором пакетов. Анализ и обработка производятся на базе заранее подготовленных правил для приложения Snort. Универсальность программы Snort заключается в вариативности наборов правил, а также возможностью подключения или отключения дополнительных модулей путем добавления или удаления тех или иных правил;

– регистратор – позволяет пользователю программного обеспечения регистрировать полученную в ходе работы информацию в лог-файл, который в последствии возможно анализировать в формате понятном и удобном пользователю.

При работе с программным обеспечением Snort пользователю доступны три основные режима [6]:

– режим пакетного снифера – программа, работающая в этом режиме читает и дешифрует все доступные сетевые пакеты и формирует дамп. В данном режиме работы программа считывает только заголовки пакетов, также доступен режим чтения пакетов в формате заголовка и содержимого пакета, но для этого потребуется изменить ключ запуска программы;

– режим регистратора пакетов – в этом режиме программа записывает проходящие через указанный сетевой интерфейс пакеты и декодирует их в формат ASCII, удобный и понятный для пользователя;

– режим обнаружения вторжения – в данном режиме поступающие сигнальные данные регистрируется механизмом обнаружения, после чего пользователю программного обеспечения поступает сигнал соответствующий правилам, прописанным в файлах правил.

После того как мы описали основные механизмы работы программы, можно приступать к настройке самого приложения. Основной файл настройки находится в папке установки в подпапке etc, он называется snort.conf. Не смотря на расширение файла он представляет собой обычный текстовый файл, который можно открыть любым текстовым редактором, распознающим формат

файлов типа .txt. Для удобства предлагаю использовать бесплатно распространяемую программу Notepad++ или её аналог (Рисунок 11).

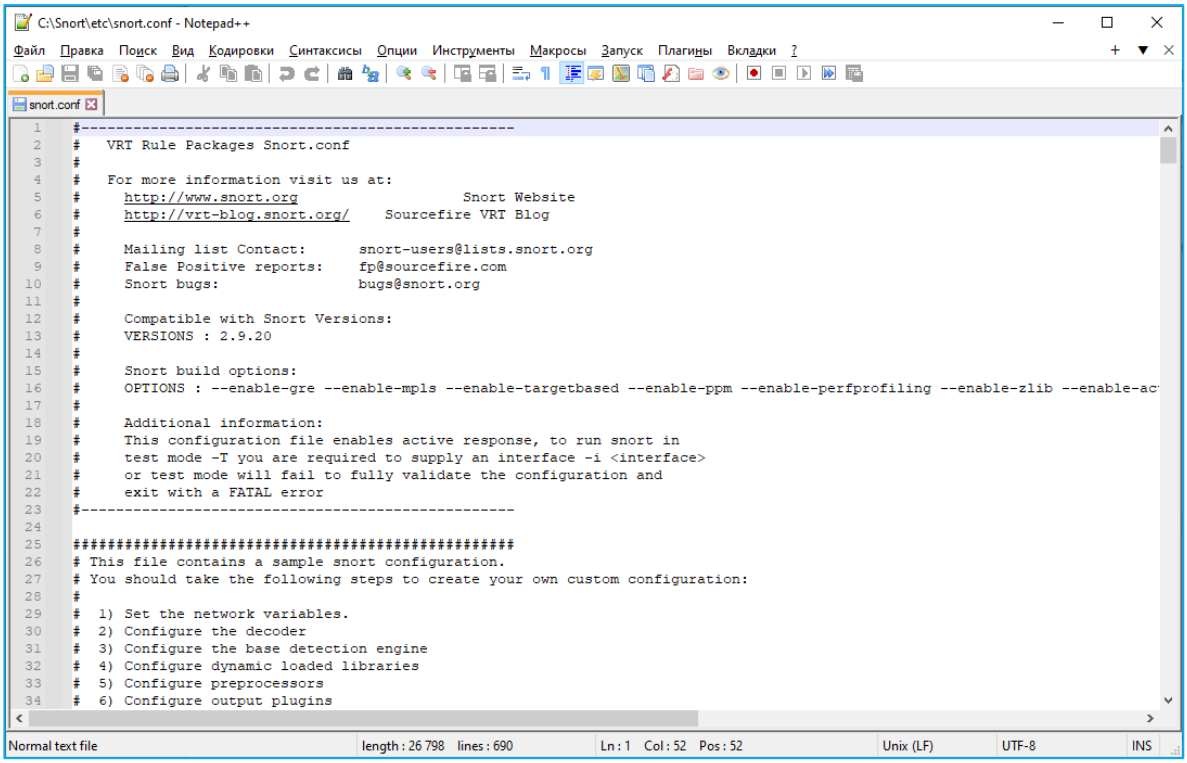

Рисунок 11 – Файл конфигурации открытый при помощи Notepad++

Первично указать программе пути, для файлов настроек и правил, для корректного взаимодействия с ними. Каждый путь записывается в соответствующую ему переменную. Символ «#» используется в конфигурационном файле как индикатор комментария строки, следственно строки начинающиеся с этого символа игнорируются компилятором при сборке приложения (Рисунок 12).

```
104  var RULE_PATH ../rules
105  var SO_RULE_PATH ../so_rules
106  var PREPROC_RULE_PATH ../preproc_rules
107
108  # If you are using reputation preprocessor set these
109  # Currently there is a bug with relative paths, they are relative to where snort is
110  # not relative to snort.conf like the above variables
111  # This is completely inconsistent with how other vars work, BUG 89986
112  # Set the absolute path appropriately
113  var WHITE_LIST_PATH ../rules
114  var BLACK_LIST_PATH ../rules
```

Рисунок 12 – Первоначальный вид переменных – указателей путей

– переменная RULE_PATH указывает путь на файлы правил, по которым программа Snort анализирует поступающий трафик;

– переменная SO_RULE_PATH указывает путь на файлы в которых описаны правила для файлов общего доступа;

– переменная PREPROC_RULE_PATH указывает путь на файлы подготовки исполнительного препроцессора;

– переменные WHITE_LIST_PATH и BLACK_LIST_PATH указывают на файлы списков, хранящих в себе информацию о белом и черном листе доступа соответственно.

Исправим путь внутри переменных в соответствии с фактическим местоположением файлов правил для программы Snort (Рисунок 13).

```
104   var RULE_PATH c:\snort\rules
105   var SO_RULE_PATH c:\snort\so_rules
106   var PREPROC_RULE_PATH c:\snort\preproc_rules
107
108   # If you are using reputation preprocessor set these
109   var WHITE_LIST_PATH c:\snort\rules
110   var BLACK_LIST_PATH c:\snort\rules
```

Рисунок 13 – Исправленные файлы путей

После исправления используемых путей необходимо указать приложению путь к папке, в которой приложение будет создавать и хранить лог-файлы. Допишем это строкой «config logdir: c:\snort\log» в конце конфигурационного файла.

В связи с тем, что изначально программа разрабатывалась для семейства операционных систем на базе Linux, большая часть путей указанных в конфигурационном файле указана на основании вида файловой системы операционных систем на базе Linux (Рисунок 14).

```
248
249   # path to base preprocessor engine
250   dynamicengine /usr/local/lib/snort_dynamicengine/libsf_engine.so
251
252   # path to dynamic rules libraries
253   dynamicdetection directory /usr/local/lib/snort_dynamicrules
254
```

Рисунок 14 – Первоначальный вид конфигурационного файла

Далее необходимо указать программе путь, по которому находятся файлы описывающие функционал основного ядра программы и директорию для динамических правил, используемых программой (Рисунок 15).

```
# path to dynamic preprocessor libraries
dynamicpreprocessor directory C:\Snort\lib\snort_dynamicpreprocessor

# path to base preprocessor engine
dynamicengine C:\Snort\lib\snort_dynamicengine\sf_engine.dll
```

Рисунок 15 – Исправленные пути к ядру программы и динамическим правилам

Еще одним пунктом подготовки конфигурационного файла является исправление пути к каждому отдельному файлу правил, использующихся программой.

Причина, по которой эта необходимость возникла также заключается в том, что первоначальной операционной системой подготовленной к работе с программой является не операционная система Windows.

В связи с этим нам необходимо в каждой строке пути заменить символ «/» на символ «\», для того что бы программа корректно воспроизводила путь для каждого конкретного файла правил (Рисунки 16, 17).

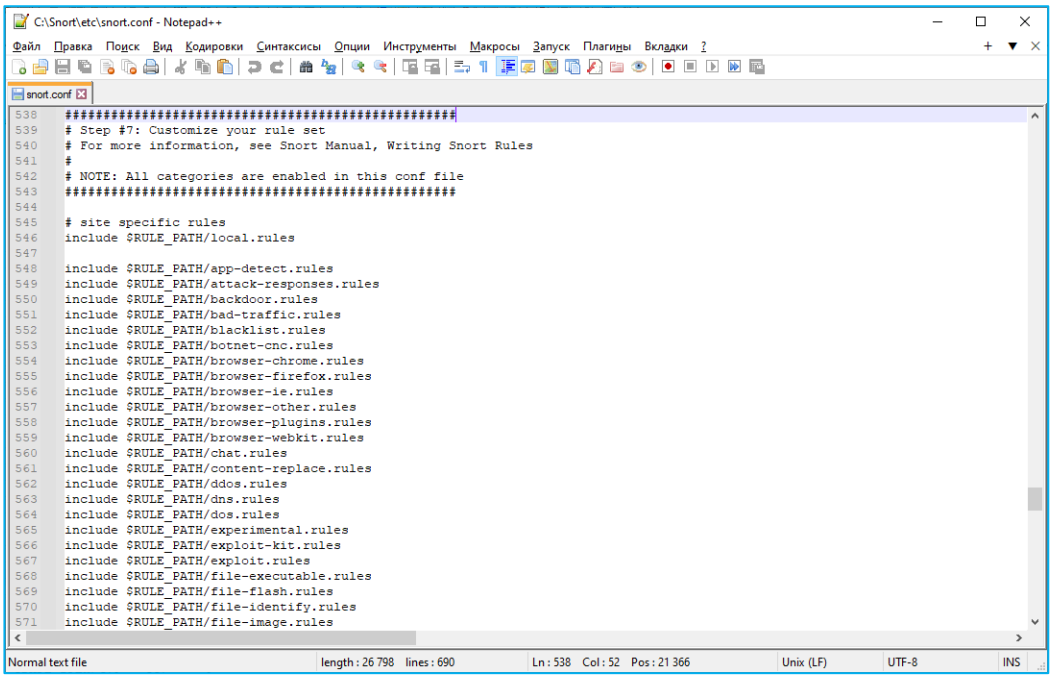

Рисунок 16 – Изначальный вид конфигурационного файла

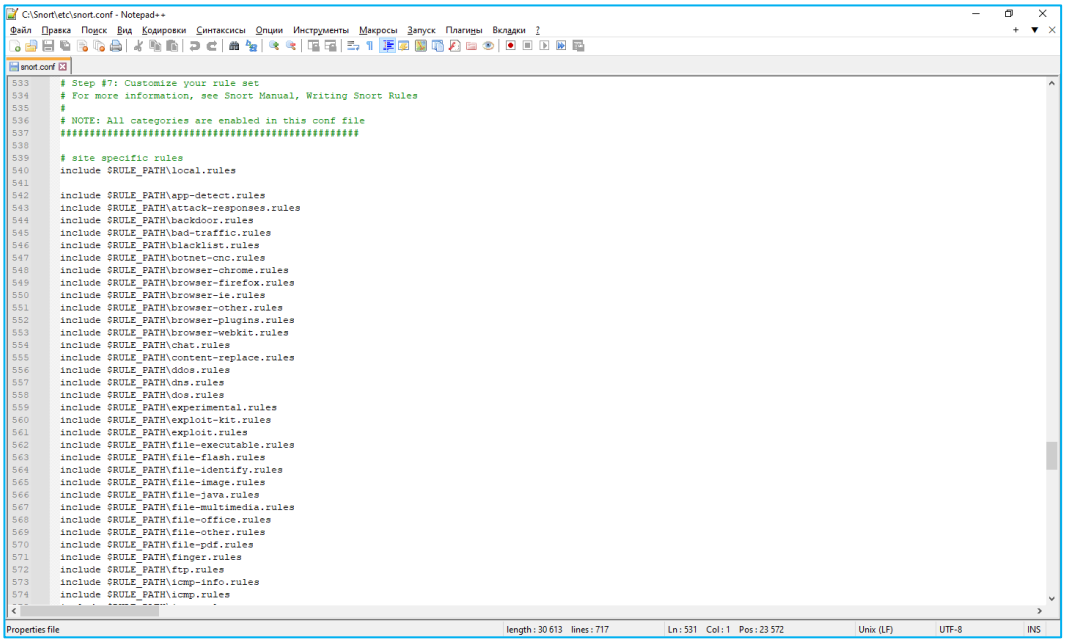

Рисунок 17 – Исправленный конфигурационный файл

Последним пунктом изменений конфигурационного файла будет установка адресов сетей (Рисунок 18), которые программа будет считать внутренними и внешними. Для упрощения проведения тестовых работ укажем диапазоном адресов домашней сети подсеть виртуального сервера. Адресами внешней сети будут считаться все другие адреса, не входящую в эту сеть.

```
# Setup the network addresses you are protecting
ipvar HOME_NET 172.23.79.207/20

# Set up the external network addresses. Leave as "any" in most situations
ipvar EXTERNAL_NET !$HOME_NET
```

Рисунок 18 – Указание диапазона адресов для внутренней и внешних сетей

На этом работу с конфигурационным файлом можно считать законченной, итоговый вид конфигурационного файла приведен в приложении А.

Следующим этапом будет установка правил для программы Snort. Файлы правил также находятся в свободном доступе на сайте разработчика. После скачивания архива с файлами правил требуется разместить их, согласно указанным ранее в конфигурационном файле путям. Правила представляет собой файл, в имени которого записана краткая информация о том, к какому виду взаимодействий относится данное правило, все файлы имеют расширение вида .rules. Файлы правил (Рисунок 19) содержат внутри описание событий или видов трафика, которые должны быть обработаны как угроза.

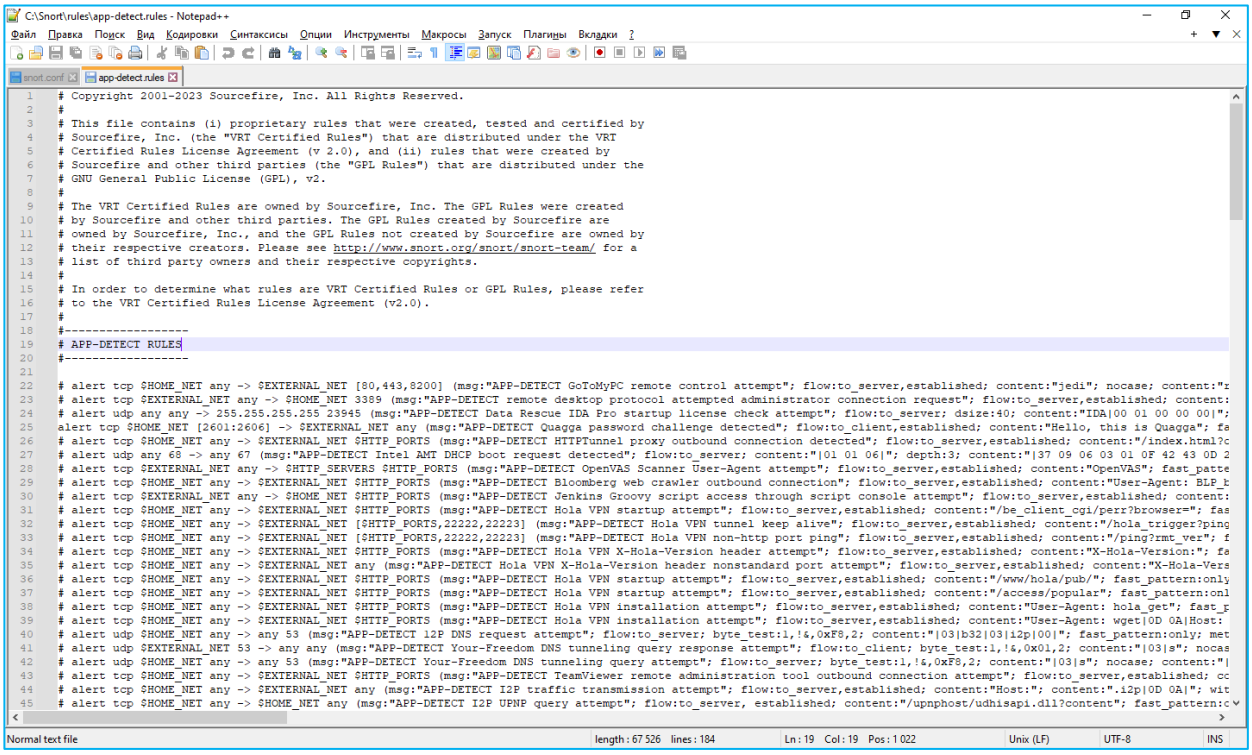

Рисунок 19 – Стандартный вид файла правил детектирующего вредоносные или компрометированные приложения

Произведя все вышеописанные действия необходимо произвести тест программы на наличие ошибок в произведенных действиях. Для этого необходимо запустить командную строку операционной системы Windows, при помощи команды «cd» изменить рабочую директорию на директорию исполняемых файлов программы Snort (Рисунок 20).

Рисунок 20 – Перемещение в рабочую директорию

Управление приложением Snort осуществляется при помощи запуска приложения через консоль при помощи контрольных ключей. Первостепенной задачей будет установление необходимого порядкового номера сетевого устройства, через которое будет происходить обмен данными виртуального сервера с внешней сетью. Запустим приложение с ключом «–W» для просмотра доступных сетевых интерфейсов (Рисунок 21).

Исходя из данных, предоставленных нам программой, делаем вывод о том, что необходимый для наших задач интерфейс располагается под индексом 4, зафиксируем этот результат, он потребуется нам в дальнейшем.

Убедимся, что все действия, выполненные нами ранее корректны, и программа запустится без ошибок. Для этого запустим программу с ключом –T (Рисунок 22), что переведет её в режим самотестирования, также необходимо

будет указать путь к конфигурационному файлу и индекс интерфейса, с которым программе предстоит работать.

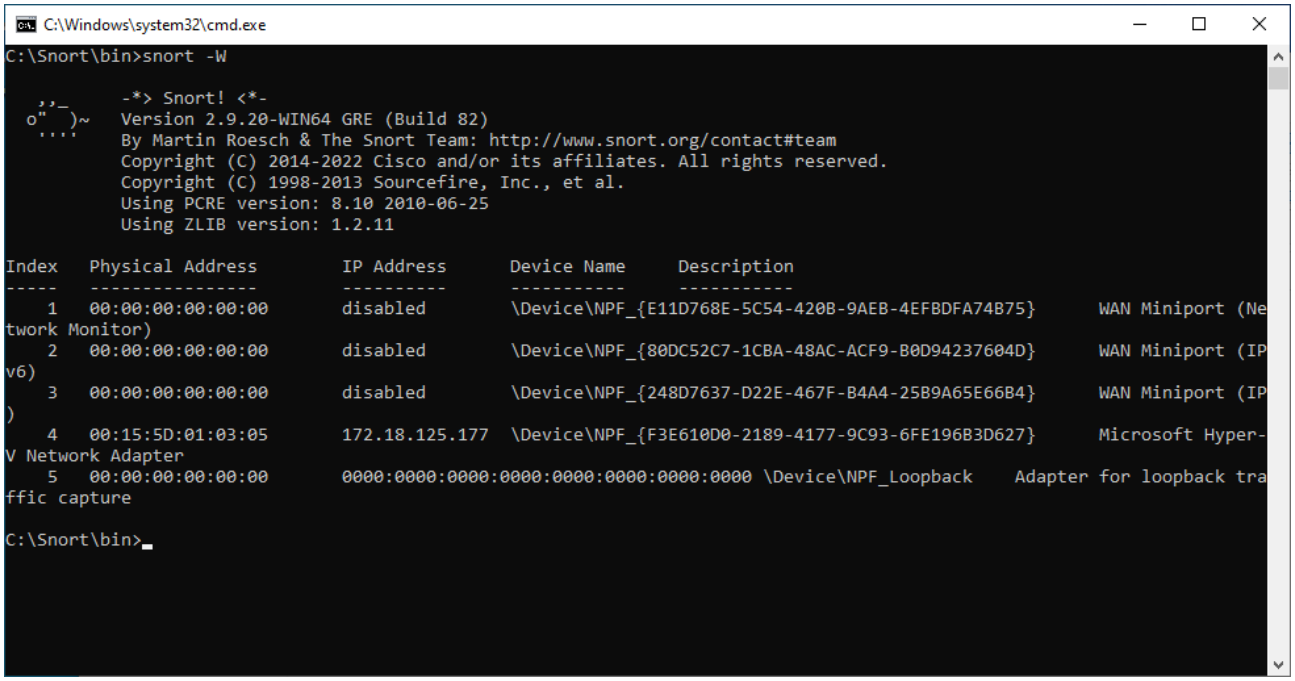

Рисунок 21 – Результат запуска программы Snort с ключом –W

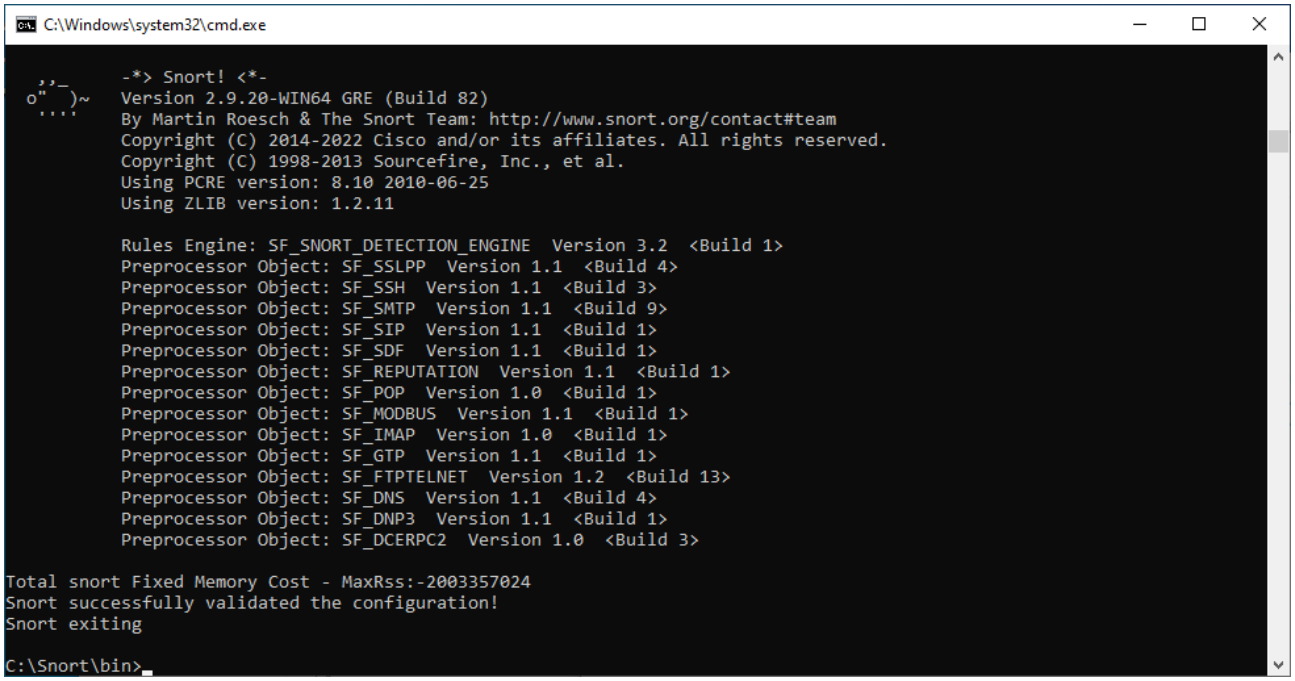

Рисунок 22 – Программное сообщение об успешном тестировании конфигурации

Далее создадим правило, для проверки корректности взаимодействия программы Snort с сетевой картой. Для этого в каталоге с файлами правил необходимо найти файл Local.rules (Рисунок 23) и добавить в него следующие параметры:

```
# Copyright 2001-2023 Sourcefire, Inc. All Rights Reserved.
#
# This file contains (i) proprietary rules that were created, tested and certified by
# Sourcefire, Inc. (the "VRT Certified Rules") that are distributed under the VRT
# Certified Rules License Agreement (v 2.0), and (ii) rules that were created by
# Sourcefire and other third parties (the "GPL Rules") that are distributed under the
# GNU General Public License (GPL), v2.
#
# The VRT Certified Rules are owned by Sourcefire, Inc. The GPL Rules were created
# by Sourcefire and other third parties. The GPL Rules created by Sourcefire are
# owned by Sourcefire, Inc., and the GPL Rules not created by Sourcefire are owned by
# their respective creators. Please see http://www.snort.org/snort/snort-team/ for a
# list of third party owners and their respective copyrights.
#
# In order to determine what rules are VRT Certified Rules or GPL Rules, please refer
# to the VRT Certified Rules License Agreement (v2.0).
#
#-------------
# LOCAL RULES
#-------------

alert icmp any any -> any any (msg:"Testing ICMP"; sid:1000001)
alert tcp any any -> any any (msg:"Testing TCP"; sid:1000002)
alert udp any any -> any any (msg:"Testing UDP"; sid:1000003)

```

Рисунок 23 – Создание тестового правила

Описав правило, запустим программу в рабочем режиме , с выводом оповещений в консоль для непосредственного контроля происходящих событий (Рисунок 24). После выполнения команды откроем еще одно окно командной строки операционной система Windows, в котором выполним команду Ping на произвольный доступный IP-адрес.

```
C:\Windows\system32\cmd.exe - snort  -i 4 -c c:\snort\etc\snort.conf -A console
05/09-16:21:25.937779  [**] [1:1000001:0] Testing ICMP [**] [Priority: 0] {ICMP} 8.8.8.8 -> 172.18.125.177
05/09-16:21:26.953820  [**] [1:1000001:0] Testing ICMP [**] [Priority: 0] {ICMP} 8.8.8.8 -> 172.18.125.177
05/09-16:21:27.969360  [**] [1:1000001:0] Testing ICMP [**] [Priority: 0] {ICMP} 8.8.8.8 -> 172.18.125.177
05/09-16:21:28.985498  [**] [1:1000001:0] Testing ICMP [**] [Priority: 0] {ICMP} 8.8.8.8 -> 172.18.125.177
05/09-16:21:30.000338  [**] [1:1000001:0] Testing ICMP [**] [Priority: 0] {ICMP} 8.8.8.8 -> 172.18.125.177
05/09-16:21:31.017155  [**] [1:1000001:0] Testing ICMP [**] [Priority: 0] {ICMP} 8.8.8.8 -> 172.18.125.177
05/09-16:21:32.031695  [**] [1:1000001:0] Testing ICMP [**] [Priority: 0] {ICMP} 8.8.8.8 -> 172.18.125.177
05/09-16:21:33.047310  [**] [1:1000001:0] Testing ICMP [**] [Priority: 0] {ICMP} 8.8.8.8 -> 172.18.125.177
05/09-16:21:34.062884  [**] [1:1000001:0] Testing ICMP [**] [Priority: 0] {ICMP} 8.8.8.8 -> 172.18.125.177
05/09-16:21:35.078506  [**] [1:1000001:0] Testing ICMP [**] [Priority: 0] {ICMP} 8.8.8.8 -> 172.18.125.177
05/09-16:21:36.094010  [**] [1:1000001:0] Testing ICMP [**] [Priority: 0] {ICMP} 8.8.8.8 -> 172.18.125.177
05/09-16:21:37.111232  [**] [1:1000001:0] Testing ICMP [**] [Priority: 0] {ICMP} 8.8.8.8 -> 172.18.125.177
05/09-16:21:38.125195  [**] [1:1000001:0] Testing ICMP [**] [Priority: 0] {ICMP} 8.8.8.8 -> 172.18.125.177
05/09-16:21:39.140812  [**] [1:1000001:0] Testing ICMP [**] [Priority: 0] {ICMP} 8.8.8.8 -> 172.18.125.177
```

Рисунок 24 – Отчет программы о регистрируемых событиях

Завершаем работу программы. Следующим этапом тестирования станет проверка на доступность и создание лог-файла. Для этого повторим предыдущую команду, добавив указание программе содержащее в себе ключ записи в файл и указание пути к файлу. Далее повторно откроем еще одно окно командной строки операционной система Windows в котором выполним команду Ping на произвольный доступный IP-адрес. По окончанию тестового запуска рекомендуется проверить доступность полученного лог-файла (Рисунок 25).

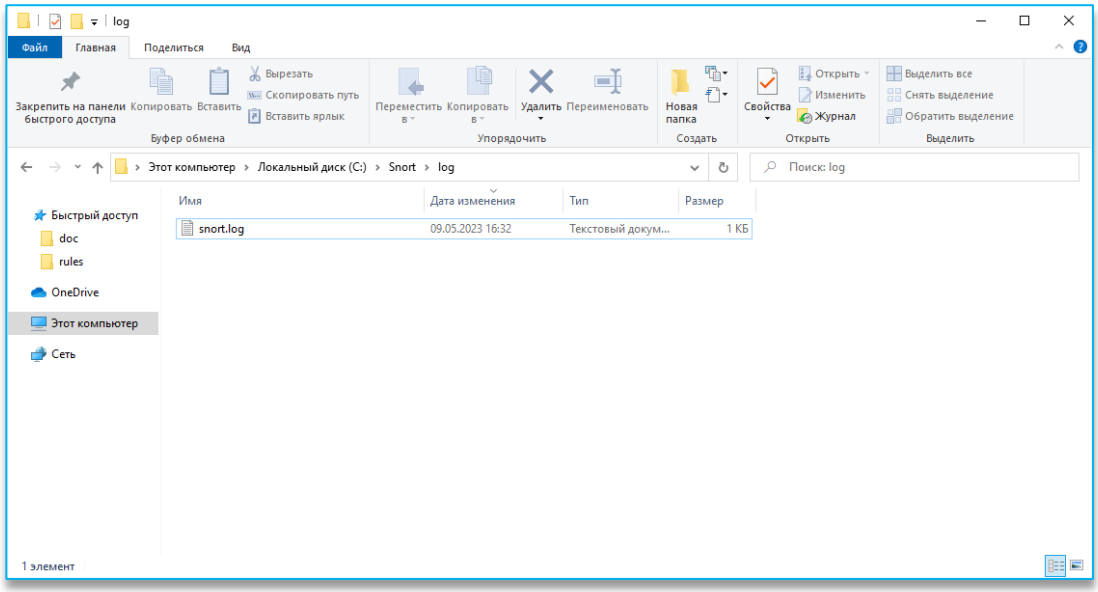

Рисунок 25 – Лог-файл созданный программой Snort

Исходя из всех полученных нами данных, а также произведенных тестовых запусков, сделаем вывод о том, что система установлена, настроена и готова к работе.

2.4 Итоговый вид предлагаемой системы

После завершения всех подготовительных процессов опишем итоговую архитектуру системы и принцип работы. На запущенном виртуальном сервере, развернутом при помощи гипервизора Hyper-v, установлена операционная система Windows 10. Установлено программное обеспечение Snort, взаимодействующее с виртуальной сетевой картой при помощи библиотеки Npcap. Сред-

ствами Docker развернуто вспомогательное приложение Snorpy для генерации правил Snort.

Сетевой трафик, проходящий через виртуальную сетевую карту, захватывается библиотекой захвата Npcap, библиотека Npcap передает трафик в изначальном виде программному обеспечению Snort для анализа.

Первичный анализ трафика выполняется препроцессорами программного обеспечения Snort, вторичный анализ, более продвинутый, происходит при использовании правил, описанных в rules-файлах.

При необходимости установки дополнительных файлов правил используется локально развернутая веб-форма Snorpy. Архитектурная схема представлена ниже на рисунке 26.

Для корректного работы системы в реальной сетевой инфраструктуре необходима рабочая станция либо сервер, с поддержкой гипервизора Hyper-V.

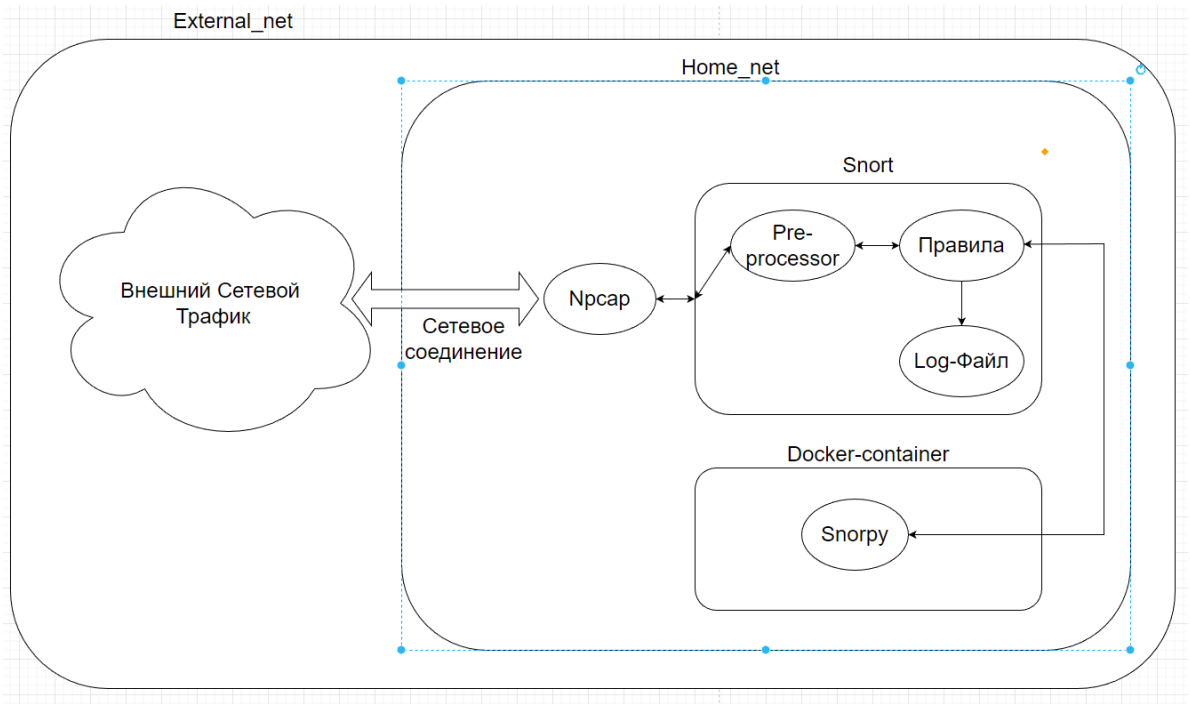

Рисунок 26 – Архитектурная схема системы

В разрыв сети, подключенной к пограничному коммутатору, устанавливается сетевой сплиттер, который дублирует весь трафик на сетевой интерфейс представленной системы.

Схема подключения представлена на рисунке 27.

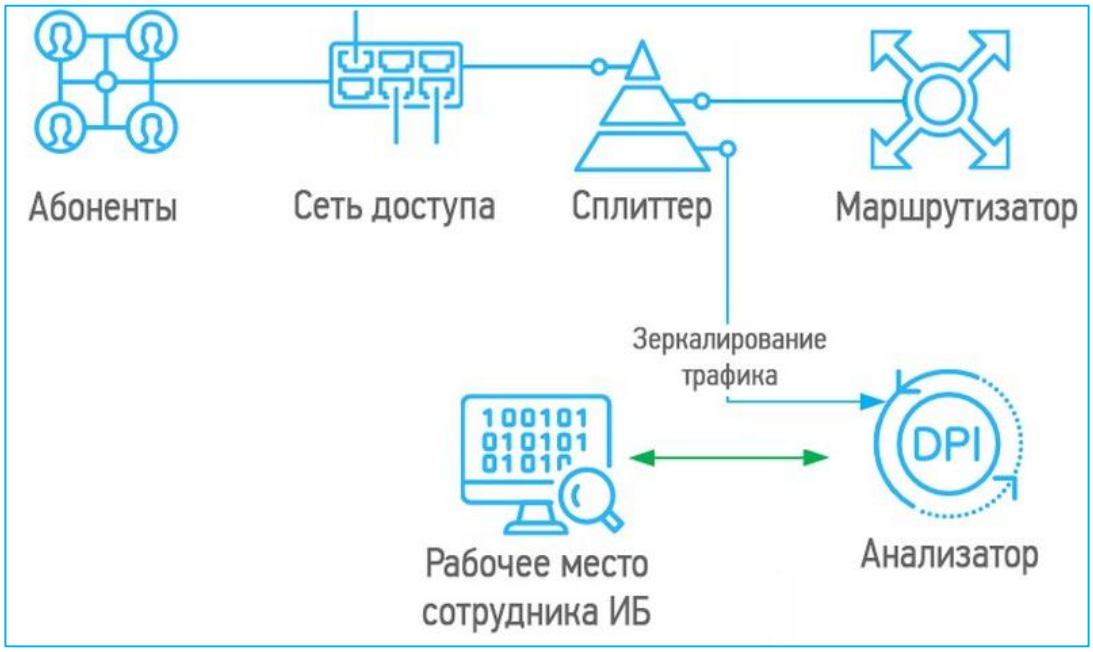

Рисунок 27 – Схема подключения системы к сегменту сети

3 Описание функционирования и основные алгоритмы анализа создаваемой системы

3.1 Описание функционирования Snort

Основой функционирования программного обеспечения Snort является работа с предварительно заданными шаблонами вредоносного трафика, эти шаблоны называются правилами (rules). Правила позволяют определить какой тип трафика регистрируемого программой является вредоносным, а какой безопасен и не несет угрозы системе. В некотором смысле этот принцип напоминает работу современных антивирусных программ, которые работают, опираясь на базы информации о вирусах, и тем самым регистрируют их сигнатуры.

Таким же образом, как и антивирусные программы, программное обеспечение Snort необходимо регулярно обновлять.

Из вышеперечисленного можно сделать вывод о том, что программное обеспечение Snort обладает уязвимостью перед всеми новыми типами атак, не известными разработчику и обширной группе пользователей Snort.

Поскольку программное обеспечение Snort распространяется на базе лицензии GPLv2 [2], текущий базовый пакет правил для Snort являет собой компиляцию всех созданных пользователями и разработчиками правил, доступных в открытом доступе. Список правил регулярно пополняется как пользователями, так и разработчиком. Пользовательские правила проходят проверку разработчиком, прежде чем попадают на сайт в список рекомендуемых.

Правила программного обеспечения Snort имеют достаточно простой синтаксис (Рисунок 28), опишем его ниже.

```
<action> <protocol> <first host> <first port> <direction> <second host> <second port> (<rule options>;)
```

Рисунок 28 – Синтаксис правила

– параметр «action» – действие, предписанное к исполнению при срабатывании правила. Примерами действий могут быть действия alert или log, первое возвращает сигнал тревоги, как это указано в настройках программного обеспечения Snort, второе записывает событие в log-файл;

– параметр «protocol» – указывает с каким протоколом будет взаимодействовать правило, основные протоколы, использующиеся при написании правил это TCP, UDP, IP и ICMP;

– параметры «first host» и «second host» – указывают на адреса обрабатываемые правилом, это могут быть как конкретные адреса в сети, так и подсети или переменные $EXTERNAL_NET или $HOME_NET отвечающие за внешнюю и внутреннюю сеть соответственно;

– параметры «first port» и» second port» – указывают на порты, обрабатываемые правилом;

– параметр «direction» – отражает направление движения траффика, обозначается символами «->»или «<-»;

– параметр «rule options» – хранит в себе комментарий и sid-номер события, вызываемого правилом. Параметр создан в основном для облегчения работы пользователя программного обеспечения Snort. Например, по sid-номеру возможно осуществление поиска по log-файлу, для выявления конкретных событий за время наблюдения.

Примером готового тестового правила может выступать следующее выражение: «alert icmp any any -> 192.168.1.1 any (msg: «ping»;)». Это правило ожидает ICMP-пакет с любого узла, направленные на адрес 192.168.1.1, и при появлении такого пакета отправляет сообщение о том, что на указанный адрес поступает Ping-запрос. Более сложные правила могут содержать переменные, как например диапазоны ip-адресов для внешней или внутренней сети.

Кроме правил программное обеспечение Snort позволяет настраивать предпроцессоры, которые сканируют траффик до того как начинается обработка правил. Основная задача предпроцессора – работа с обычным или известным траффиком, например сканирование портов или ping-пакеты, которые могут

снижать скорость обработки через обычные правила, которые требуют больше ресурсов.

Файл Snort.conf представляет собой конфигурационный файл, который описывает пути для директорий с правилами, пути файлов препроцессоров, задает переменные отвечающие за обозначение внутренней и внешней сети.

Основой взаимодействия с программным обеспечением Snort является командная строка, в нашем случае – командная строка Windows. Несмотря на опции указанные в конфигурационном файле, опции запуска приложения в командной строке имеют больший приоритет. Поскольку основа взаимодействия с программой – командная строка, разработчик предоставляет нам возможность обширного использования префиксов запуска, для решения локальных задач, изменения файла конфигурации для которых было бы избыточным, ввиду, например, едино разовой необходимости исполнения команды или выполнения проверки. Примеры таких префиксов:

– префикс A – может принимать значения: fast, full, console или none. Fast предназначена для быстрого генерирования алертов. Этот параметр рекомендуют использовать не только разработчики, но и специалисты в информационной безопасности. Full – самый медленный способ, используется при необходимости;

– префикс b – журналировать пакеты в формате tcpdump. (Это наиболее быстрый и производительный вариант);

– префикс h – выводит экран помощи;

– префикс I – обозначает используемый для прослушивания интерфейс;

– префикс N – отключает ведение log-файла.

После того, как указаны эти опции, можно использовать так называемые опции фильтров BPF. Это довольно интересная возможность, позволяющая исключать определенные хосты для того, чтобы игнорировать их трафик. Это быстрый способ, так как Snort при этом вообще не «видит» пакеты, так как они сбрасываются на интерфейс BPF.

Пример использования команды, которая позволит игнорировать все пакеты от IP-адреса 127.0.0.1: «snort <опции_командной_строки> not host 127.0.0.1».

Пример команды, позволяющей игнорировать весь ICMP трафик: «snort<опции_командной_строки>``not ( (icmp[0] = 8 or icmp[0] = 0) and host )``».

3.2 Генератор правил Snorpy

Генератор правил для программного обеспечения Snort призван облегчить работу по созданию персональных правил. Генератор правил Snorpy распространяется по лицензии GNU General Public License v2.0 и представляет собой веб-форму (Рисунок 29), которая автоматически генерирует правила для программного обеспечения Snort. Веб-форма доступна на сейте разработчика приложения, либо в качестве docker – контейнера. Развернув docker-контейнер, пользователь получит локальную версию веб-формы, доступную, например, при отключении от глобальной сети Интернет.

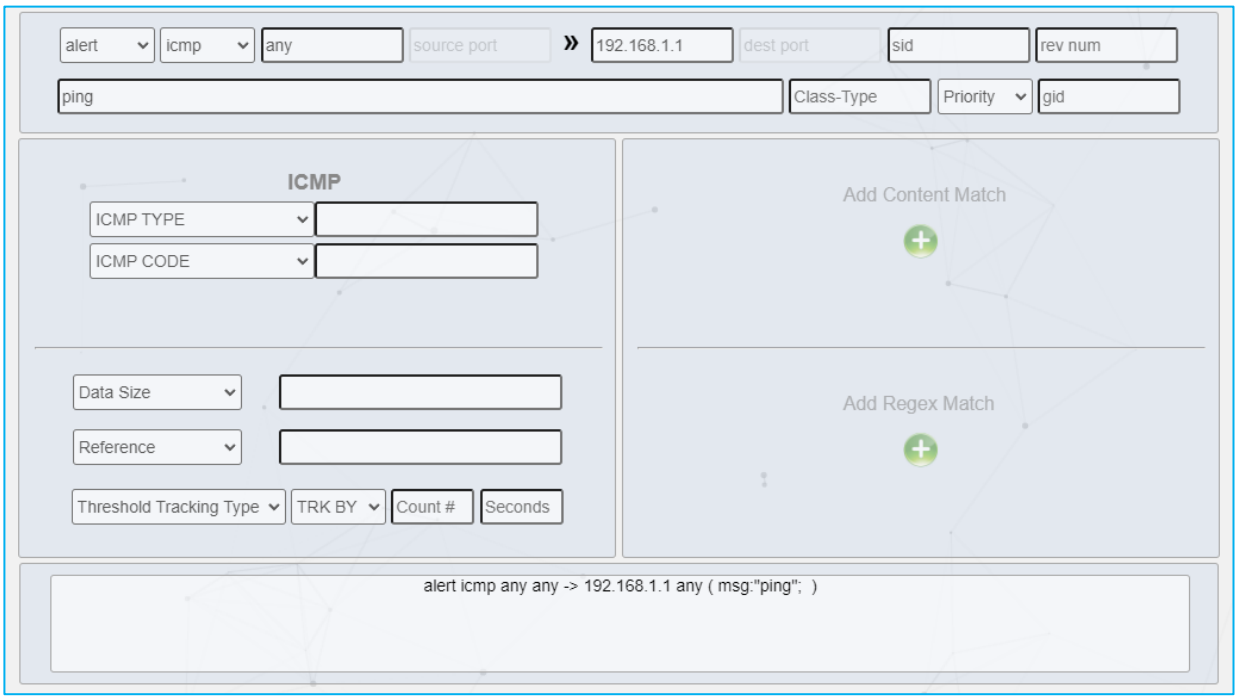

Рисунок 29 – Веб-форма Snorpy

Ядром проекта выступает код, исполненный на языке программирования JavaScript, Веб-форма описана при помощи HTML.

3.3 Библиотека захвата пакетов Npcap

Библиотека захвата пакетов Npcap – это библиотека захвата и отправки пакетов проекта Nmap, для семейства операционных систем Windows, она реализует открытый API «Pcap» с использованием пользовательского драйвера ядра Windows. Это позволяет программному обеспечению Windows захватывать необработанный сетевой трафик (включая беспроводные сети, проводную сеть Ethernet, локальный трафик и многие VPN), а также позволяет отправлять необработанные пакеты. Nmap – сокращение от «Network Mapper», наиболее корректным переводом этого выражения будет словосочетание «Сетевой картограф», что наиболее правильно отразит суть работы данной программы. Функционал Npcap обширен, примерами действий выполняемых про помощи Npcap могут быть:

– захват необработанных пакетов;

– фильтрация пакетов перед отправкой согласно заданным правилам;

– передача исходных данных в сети;

– сбор информации о сетевом трафике.

В данной работе библиотека захвата пакетов Npcap позволяет программному обеспечению Snort взаимодействовать с сетевыми пакетами в том виде, в котором они попадают к обработчику операционной системы. Эта необходимость обусловлена тем, что обработчик пакетов операционной системы после обработки сетевого пакета отбрасывает большую часть служебной информации заключенной в пакете. Программному обеспечению Snort необходима полная информация о пакете, в том виде, в котором он поступил в распоряжение системы. Это необходимо для снижения вероятности ошибочной индикации сигнатуры атаки, или же наоборот, срабатывания ложной тревоги.

4 Тестирование системы

4.1 Обработка эталонных файлов

Для проверки работы правил программного обеспечения Snort необходимо получить набор эталонных файлов. Эталонные файлы представляют дамп подозрительного трафика в формате pcap. Pcap – библиотека, предшествующая библиотеке Npcap. Несмотря на переход на новую версию библиотеки формат данных в ней сохранился без изменений, это сделано для возможности сохранения кроссплатформенной обработки. После загрузки пакета эталонных файлов их необходимо просканировать программным обеспечением Snort. Воспроизведем в командной строке обращение к программному обеспечению Snort в следующем виде: «Snort -pcap-dir=c:\test -c c:\snort\etc\snort.conf –K ascii -l c:\snort\log»:

– аргумент «-pcap-dir=c:\test» указывает, какую папку необходимо просканировать, с использованием инструкций библиотеки pcap;

– аргумент «-c c:\snort\etc\snort.conf» указывает программному обеспечению Snort на файл конфигурации, с которым будет происходить взаимодействие;

– аргумент «–K ascii» позволяет кодировать log-файл в формате ascii, удобным для прочтения пользователем;

– аргумент «-l c:\snort\log» указывает путь к директории, в которой будут созданы log-файлы.

Результат исполнения этой команды изображен на рисунках 30, 31.

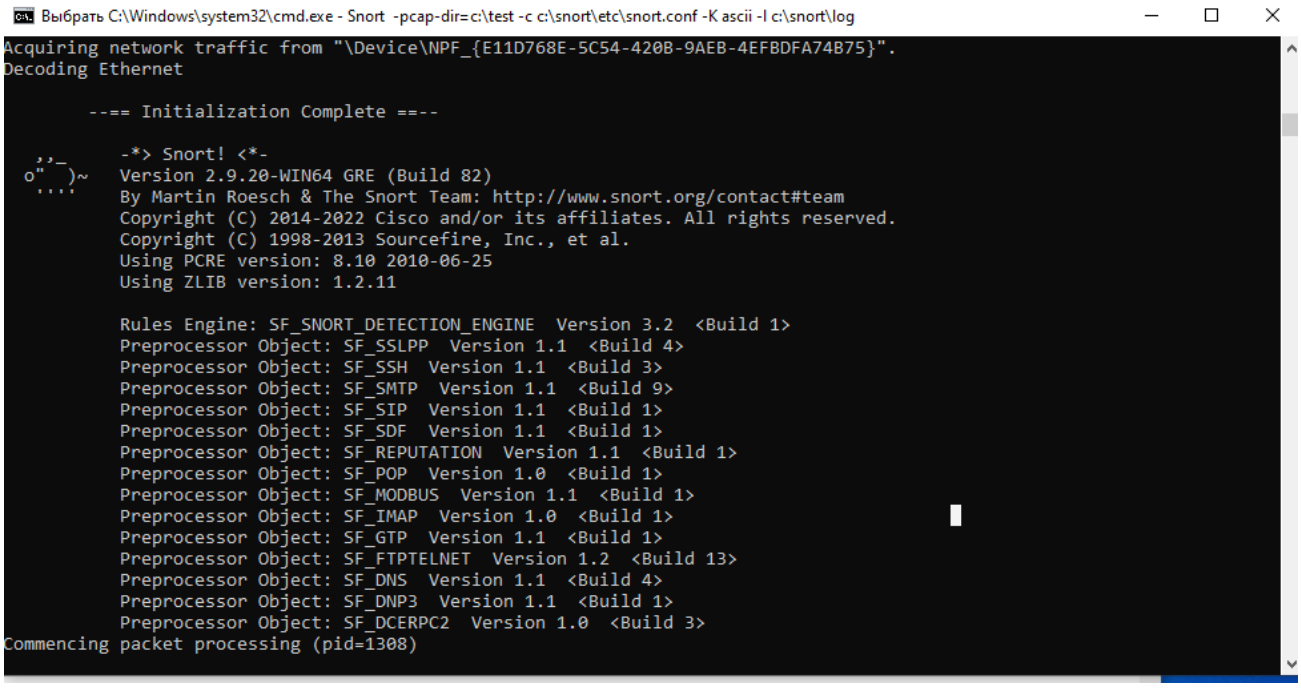

Рисунок 30 – Процесс обработки эталонных файлов

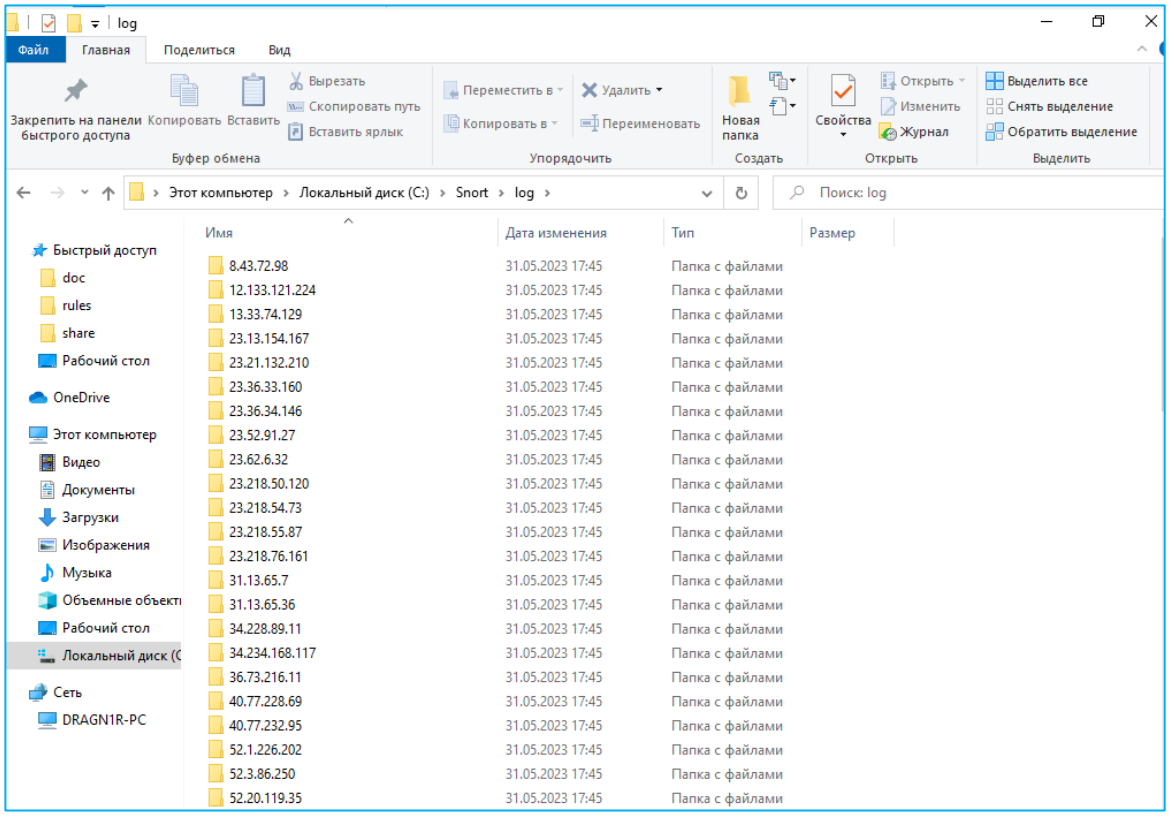

Рисунок 31 – Результат обработки эталонных файлов

Результатом обработки эталонного файла является группа директорий, сгруппированных по IP-адресу, с которого были зарегистрированы обращения к

системе – источнику pcap файла. Каждая директория содержит в себе один или более log-файл с расширением «.ids». Дополнительно система сгенерировала общий отчет содержащий в себе общую информацию о произведенном анализе. Общий объем файла с указанием тревог (alerts) составил 36172 строк. Внутренняя информация содержащаяся в log-файле закодирована в формате ascii, что позволяет извлечь информацию в удобном пользователю виде. Ознакомимся с содержимым log-файла, используя программу Notepad++ (Рисунок 32).

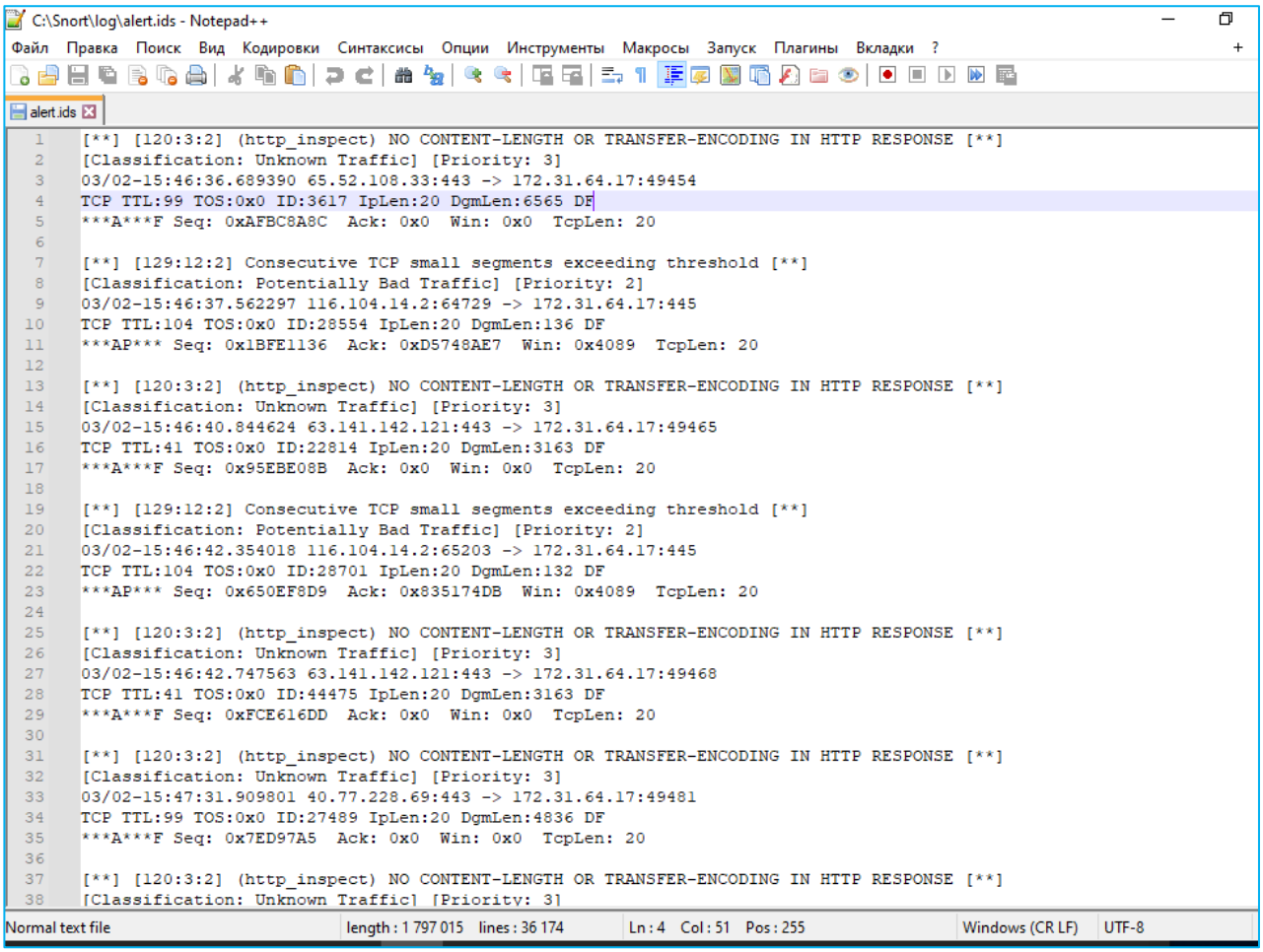

Рисунок 32 – Содержимое log-файла

Разберем структуру оповещения, содержащегося в log-файле, сгенерированного при помощи программного обеспечения Snort. Оповещение описывает пользователю время и дату события, записывает IP-адрес отправителя пакета, IP-адрес целевого узла, тип атаки, служебную информацию, содержащую в себе параметры сети, и указывает приоритет события, который обозначен внутри

файла правил. Дополнительно, если это указать в файле правила, будет указан sid события, для облегчения поиска определенных событий внутри log-файла.

4.2 Обработка эталонного файла расположенного на удаленном хосте

Программное обеспечение Snort позволяет производить сканирование и анализ файлов pcap расположенных в удаленных сетевых директориях. Сканирование pcap – файлов расположенных в удаленных сетевых директориях производится аналогично сканированию файлов расположенных непосредственно на жестком диске сервера. Для произведения сканирования необходимо заменить путь к директории на жестком диске, на абсолютный путь расположенный в сетевом окружении (Рисунок 33).

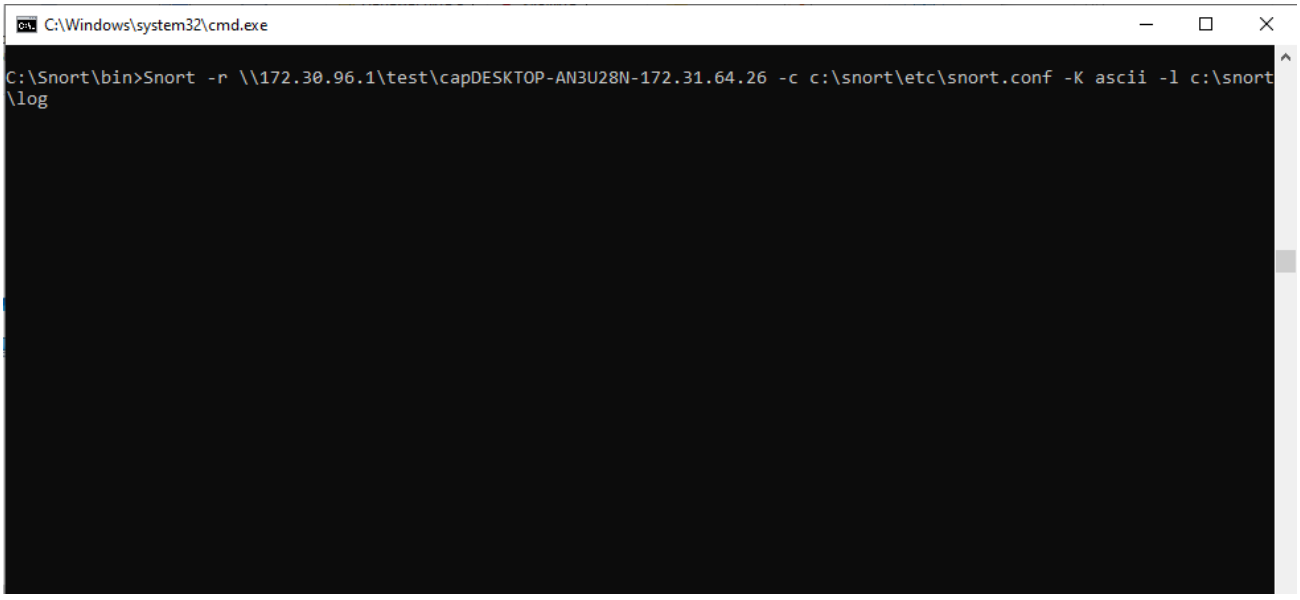

Рисунок 33 – Команда сканирования pcap файла, расположенного в удаленной директории

Произведя сканирование pcap файла расположенного в удаленной директории и произведя сопоставительную оценку длительности анализа эталонного файла сделаем вывод о том, что в случае анализа эталонного файла расположенного в удаленной директории лимитирующим фактором быстродействия алгоритма анализа является пропускная способность сетевого канала, которая

значительно ниже скорости взаимодействия протоколов передачи данных использующихся для обмена информацией внутри одного виртуального сервера. Таким образом из произведенных тестовых операций мы можем сделать вывод о том, что хранение и анализ файлов pcap на удаленных станциях возможен, но приводит к замедлению общего времени работы системы. Данный подход возможно применять в случаях работы с ограниченным дисковым пространством.

4.3 Тестирование системы в режиме мониторинга

Запустим программное обеспечение snort с использованием выражения «snort -i 4 -c c:\snort\etc\snort.conf -A console». Данное выражение позволит нам наблюдать трафик проходящий через сетевую карту виртуального сервера в режиме мониторинга. Для наглядности воспользуемся приложением Snorpy (рисунок 34) и сгенерируем несколько правил, которые будут создавать уведомления типа «alert» обнаружив пакеты использующие протоколы TCP, UDP и ICMP.

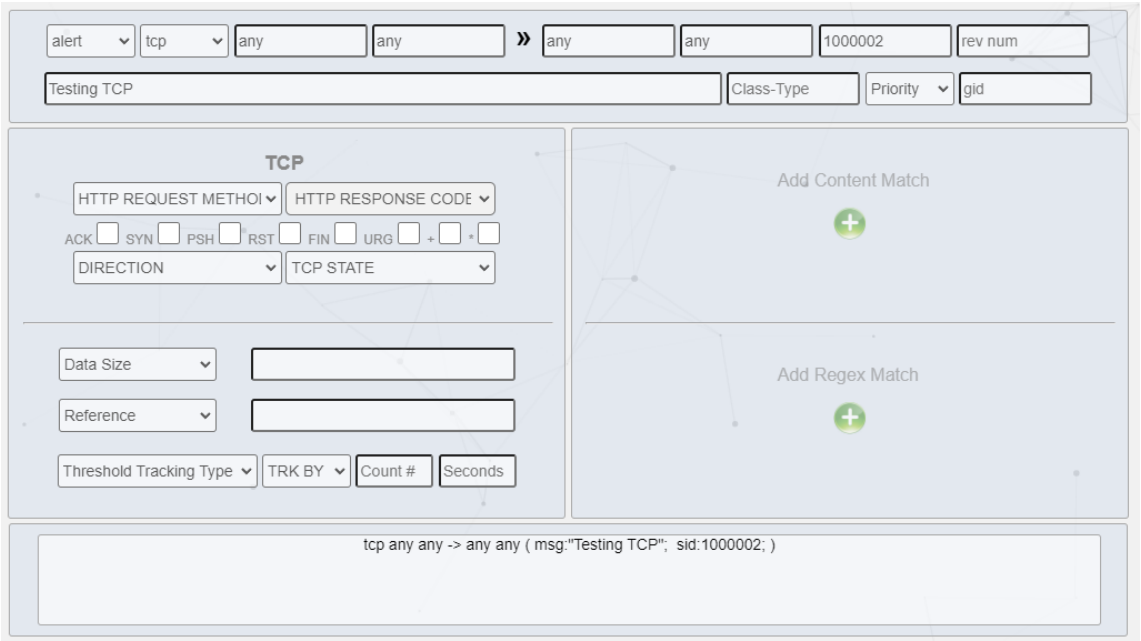

Рисунок 34 – Генерация правила для Snort при помощи приложения Snorpy

Создав необходимые правила при помощи приложения Snorpy и добавим их в файл local.rules (Рисунок 35), сохраним его и запустим программное обеспечение Snort.

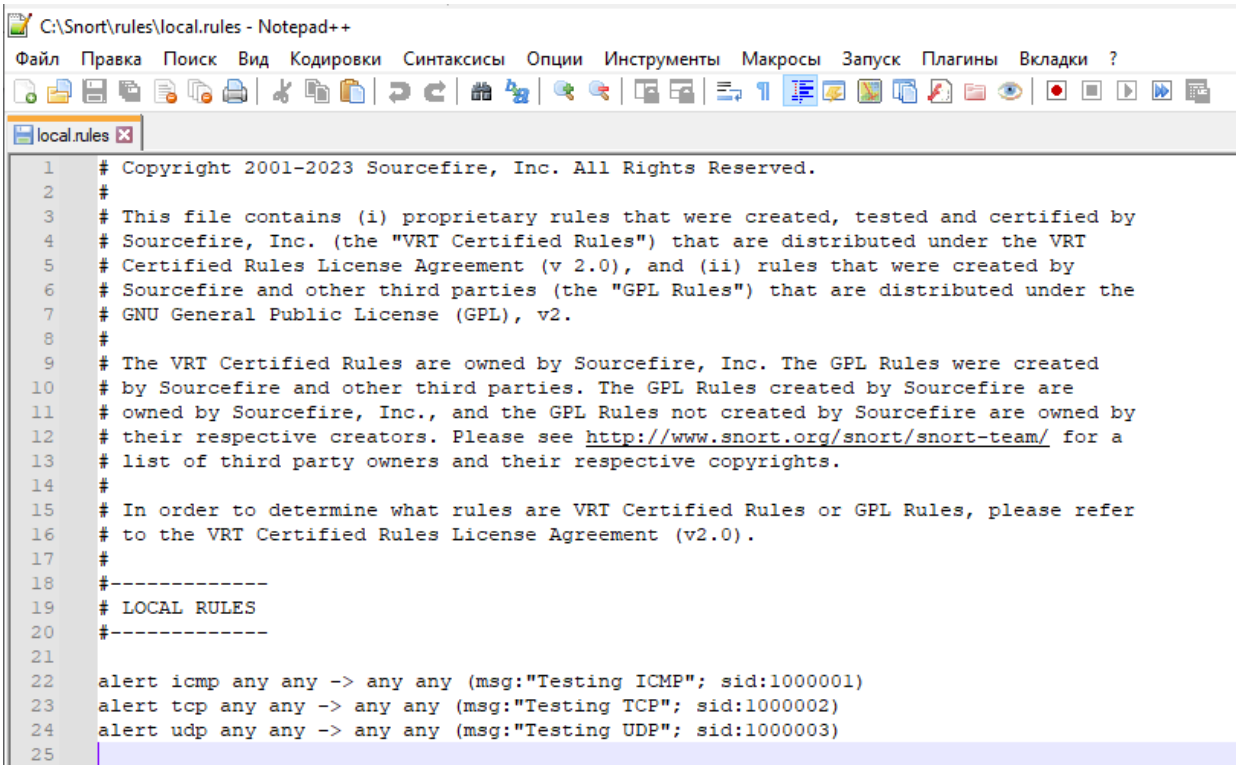

Рисунок 35 – Содержимое файла local.rules во время тестирования системы в режиме мониторинга

Во время работы программного обеспечения Snort в командной строке будут возникать событий типа «alert» (Рисунок 36). Произведем анализ событий:

– обмен TCP и UDP пакетами с ip-адресом 172.30.96.1 – обмен сервисными пакетами с хост-машиной виртуального сервера через внутреннюю сеть гипервизора Hyper-v;

– TCP-сессия не верифицированная методом трехстороннего рукопожатия поступающая с ip-адреса 13.69.109.131. Данный адрес зарегистрирован как адрес корпорации Microsoft и используется для обмена служебными сообщениями с операционной системой Windows.

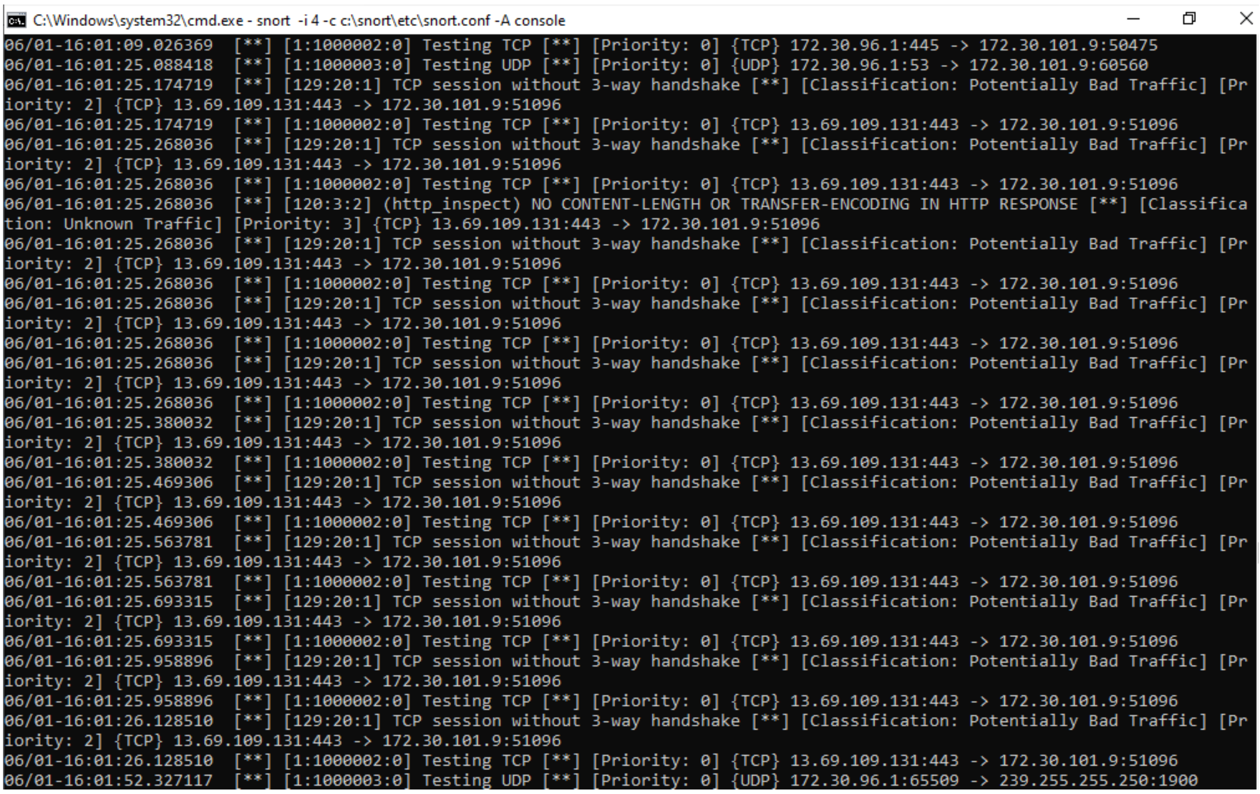

Рисунок 36 – Командная строка в процессе мониторинга сетевого соединения

ЗАКЛЮЧЕНИЕ

В первой части выпускной квалификационной работы был произведен обзор технологии DPI. Для более глубокого понимания темы была рассмотрена модель OSI, на которую ссылается описание технологии DPI. Описан принцип действия и рассмотрены основные недостатки в имеющихся решениях. Рассмотрены основные поставщики DPI-систем, оценены их решения, выявлены сильные и слабые стороны решений, а также выбран пример наилучшей реализации.

Приведены примеры использования DPI-систем в различных сферах. Проведен краткий экскурс во взаимодействие систем DPI и DLP, приведены примеры классификаций DLP-систем и их взаимодействие с DPI. Составлена краткая схема сопряжения двух систем.

Во второй части выпускной квалификационной работы был произведен обзор использующихся технологий. Были рассмотрены технологии виртуализации на базе гипервизора Hyper-V. Кратко рассмотрен процесс создания виртуального сервера. Обзор приложения Snort и приложения Npcap, необходимого для его работы.

После этого была произведена установка и настройка приложения Snort, приложения Npcap, а также произведена настройка приложения Snort путем изменения конфигурационного файла и установка файлов правил для приложения Snort.

Используя тестовое правило для оценки состояния и работоспособности программы, была произведена проверка и оценка состояния системы, а также оценка работоспособности отдельных элементов системы, и всех элементов системы в совокупности.

Третья часть выпускной квалификационной работы описывает функционирование системы и приводит краткий разбор основных алгоритмов составных элементов системы, включающие в себя описание функционирования про-

граммного обеспечения Snort, описание работы генератора правил Snorpy и описание функционала библиотеки захвата пакетов Npcap.

В четвертой части выпускной квалификационной работы произведены тестовые запуски системы, система протестирована эталонным файлом, протестирована возможность сканирования эталонного файла расположенного на удаленном хосте, а также тестирование системы в режиме мониторинга, выявившего, недоступную без использования системы, информацию о сетевых взаимодействиях происходящих на виртуальном сервере.

По итогу выполненной работы мы получили систему анализа пограничного трафика на основе глубокого анализа пакетов. Система базируется на виртуальном сервере под руководством операционной системы Windows 10. Система способна производить анализ предварительно записанных pcap-файлов, осуществлять мониторинг сетевого трафика в реальном времени и защиту от известных видов получения несанкционированного доступа к сетевым ресурсам.

К плюсам системы можно отнести:

– малый объем необходимого дискового пространства для работы системы;

– низкие требования к вычислительным ресурсам базовой станции;

– портативность;

– возможность встраивания в более крупные системы в качестве резервного или вспомогательного звена.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Бил, Д. Snort 2.1. Обнаружение вторжений / Д. Бил. – Москва : Издательство ООО «Бином-Пресс», 2006. – 656 с. – ISBN 5-9518-0136-2.

2. Васенин, В. А. Экспресс-анализ потоковых текстовых данных на предмет вхождения в них ключевых слов и фраз / В. А. Васенин, В. А. Роганов, М. Д. Дзабраев // Программная инженерия. – 2016 – Т. 7, № 1. – С. 3–12.

3. Конахович, Г.Ф. Сети передачи пакетных данных / Г. Ф. Конахович, В. М. Чуприн. – Киев : МК-Пресс, 2006. – 272 с. – ISBN: 978-966-8806-26-1.

4. Куроуз, Д. Компьютерные сети. Нисходящий подход. / Д. Куроуз, К. Росс. – Москва : Издательство «Эксмо», 2016. – 912 с. – ISBN 978-5-699-78090-7.

5. Моримото, Р. Windows Server 2008 Hyper-V Unleashed / Р. Моримото, М. Ноэл, Э. Аббат, К. Амарис, Г. Ярдени. – Москва : Издательский дом «Вильямс», 2014. – 800 с. – ISBN: 978-5-8459-1863-5.

6. Олифер, В. Г. Компьютерные сети. Принципы, технологии, протоколы. Юбилейное издание / В. Г. Олифер, Н. А. Олифер. – Санкт-Петербург : Питер, 2021. – 1008 с. – ISBN: 978-5-4461-1426-9.

7. Уилсон, Э. Мониторинг и анализ сетей: Методы выявления неисправностей / Э. Уилсон ; пер. с англ. О. Труфанов. – Москва : Лорри, 2012. – 350 с.

8. Уэнделл, О. Руководство по технологиям объединённых сетей. / О. Уэенделл. – 4-е изд. – Сан-Хосе (Калифорния) : Cisco press, 2005. – 1040 с. – ISBN 5-8459-0787-X.

9. Филимонов, А. Ю. Построение мультисервисных сетей Ethernet / А. Ю. Филимонов. – Москва : BHV, 2007. – 592 с. – ISBN 978-5-9775-0007-4.

10. Balakrishnan, R. Advanced QoS for Multi–Service IP/MPLS Networks / R. Balakrishnan. – New Jersey : Wiley, 2008. – 464 p. – ISBN 9780470293690.

11. Heckmann, O. The Competitive Internet Service Provider: Network Architecture, Interconnection, Traffic Engineering and Network Design / O. Heckmann. – London, 2007. – 400 p.

12. Michael W. Network Flow Analysis 1st edition by Lucas / W. Michael. – San Francisco, 2010. – 209 p.

13. Russell, J. Deep packet inspection / J. Russell, 2013. – 136 p. – ISBN: 978-5-5091-6394-4.

ПРИЛОЖЕНИЕ А

Конфигурационный файл «Snort»

```
#--------------------------------------------------
#   VRT Rule Packages Snort.conf
#
#   For more information visit us at:
#     http://www.snort.org                  Snort Website
#     http://vrt-blog.snort.org/     Sourcefire VRT Blog
#
#     Mailing list Contact:      snort-sigs@lists.sourceforge.net
#     False Positive reports:    fp@sourcefire.com
#     Snort bugs:                bugs@snort.org
#
#     Compatible with Snort Versions:
#     VERSIONS : 2.9.20.0
#
#     Snort build options:
#       OPTIONS : --enable-gre --enable-mpls --enable-targetbased --enable-ppm --enable-perfprofiling --
enable-zlib --enable-active-response --enable-normalizer --enable-reload --enable-react --enable-flexresp3
#
#     Additional information:
#     This configuration file enables active response, to run snort in
#     test mode -T you are required to supply an interface -i <interface>
#     or test mode will fail to fully validate the configuration and
#     exit with a FATAL error
#--------------------------------------------------

######################################################
# This file contains a sample snort configuration.
# You should take the following steps to create your own custom configuration:
#
#  1) Set the network variables.
#  2) Configure the decoder
#  3) Configure the base detection engine
#  4) Configure dynamic loaded libraries
#  5) Configure preprocessors
#  6) Configure output plugins
#  7) Customize your rule set
#  8) Customize preprocessor and decoder rule set
#  9) Customize shared object rule set
```

```
##################################################

##################################################
# Step #1: Set the network variables.  For more information, see README.variables
##################################################

# Setup the network addresses you are protecting
ipvar HOME_NET 172.23.79.207/20

# Set up the external network addresses. Leave as "any" in most situations
ipvar EXTERNAL_NET !$HOME_NET

# List of DNS servers on your network
ipvar DNS_SERVERS $HOME_NET

# List of SMTP servers on your network
ipvar SMTP_SERVERS $HOME_NET

# List of web servers on your network
ipvar HTTP_SERVERS $HOME_NET

# List of sql servers on your network
ipvar SQL_SERVERS $HOME_NET

# List of telnet servers on your network
ipvar TELNET_SERVERS $HOME_NET

# List of ssh servers on your network
ipvar SSH_SERVERS $HOME_NET

# List of ftp servers on your network
ipvar FTP_SERVERS $HOME_NET

# List of sip servers on your network
ipvar SIP_SERVERS $HOME_NET

# List of ports you run web servers on
portvar HTTP_PORTS
[36,80,81,82,83,84,85,86,87,88,89,90,311,323,383,443,444,555,591,593,623,631,664,801,808,818,901,972,1158,12
20,1270,1414,1533,1581,1719,1720,1741,1801,1812,1830,1942,2231,2301,2375,2381,2578,2809,2869,2980,3000,3
029,3037,3057,3128,3323,3443,3702,4000,4343,4444,4848,5000,5054,5060,5061,5117,5222,5250,5416,5443,5450,
5480,5555,5600,5814,5894,5984,5985,5986,6060,6080,6173,6988,7000,7001,7005,7070,7071,7080,7144,7145,718
```

0,7181,7510,7770,7777,7778,7779,8000,8001,8008,8014,8015,8020,8028,8040,8080,8081,8082,8085,8088,8090,8095,8118,8123,8161,8180,8181,8182,8222,8243,8280,8300,8333,8344,8393,8400,8443,8484,8500,8509,8511,8694,8787,8800,8848,8852,8880,8888,8899,8983,9000,9001,9002,9050,9060,9080,9090,9091,9111,9200,9201,9290,9443,9447,9502,9700,9710,9788,9830,9850,9990,9999,10000,10080,10100,10250,10255,10297,10443,11371,12601,13014,15489,16000,16992,16993,16994,16995,17000,18081,19980,20000,29991,30007,30018,30888,33300,34412,34443,34444,36099,37215,40007,41080,44449,49152,49153,50000,50002,50452,51423,53331,54444,55252,55555,56712]

```
# List of ports you want to look for SHELLCODE on.
portvar SHELLCODE_PORTS !80

# List of ports you might see oracle attacks on
portvar ORACLE_PORTS 1024:

# List of ports you want to look for SSH connections on:
portvar SSH_PORTS 22

# List of ports you run ftp servers on
portvar FTP_PORTS [21,2100,3535]

# List of ports you run SIP servers on
portvar SIP_PORTS [5060,5061,5600]

# List of file data ports for file inspection
portvar FILE_DATA_PORTS [$HTTP_PORTS,110,143]

# List of GTP ports for GTP preprocessor
portvar GTP_PORTS [2123,2152,3386]

# other variables, these should not be modified
ipvar AIM_SERVERS
```

[64.12.24.0/23,64.12.28.0/23,64.12.161.0/24,64.12.163.0/24,64.12.200.0/24,205.188.3.0/24,205.188.5.0/24,205.188.7.0/24,205.188.9.0/24,205.188.153.0/24,205.188.179.0/24,205.188.248.0/24]

```
# Path to your rules files (this can be a relative path)
# Note for Windows users:  You are advised to make this an absolute path,
# such as:  c:\snort\rules
var RULE_PATH c:\snort\rules
var SO_RULE_PATH c:\snort\so_rules
var PREPROC_RULE_PATH c:\snort\preproc_rules

# If you are using reputation preprocessor set these
```

```
var WHITE_LIST_PATH c:\snort\rules
var BLACK_LIST_PATH c:\snort\rules

##################################################
# Step #2: Configure the decoder.  For more information, see README.decode
##################################################

# Stop generic decode events:
config disable_decode_alerts

# Stop Alerts on experimental TCP options
config disable_tcpopt_experimental_alerts

# Stop Alerts on obsolete TCP options
config disable_tcpopt_obsolete_alerts

# Stop Alerts on T/TCP alerts
config disable_tcpopt_ttcp_alerts

# Stop Alerts on all other TCPOption type events:
config disable_tcpopt_alerts

# Stop Alerts on invalid ip options
config disable_ipopt_alerts

# Alert if value in length field (IP, TCP, UDP) is greater th elength of the packet
# config enable_decode_oversized_alerts

# Same as above, but drop packet if in Inline mode (requires enable_decode_oversized_alerts)
# config enable_decode_oversized_drops

# Configure IP / TCP checksum mode
config checksum_mode: all

config logdir: c:\snort\log


##################################################
# Step #3: Configure the base detection engine.  For more information, see  README.decode
##################################################

# Configure PCRE match limitations
```

```
config pcre_match_limit: 3500
config pcre_match_limit_recursion: 1500

# Configure the detection engine  See the Snort Manual, Configuring Snort - Includes - Config
config detection: search-method ac-split search-optimize max-pattern-len 20

# Configure the event queue.  For more information, see README.event_queue
config event_queue: max_queue 15 log 15 order_events content_length

config paf_max: 16000

##################################################
# Step #4: Configure dynamic loaded libraries.
# For more information, see Snort Manual, Configuring Snort - Dynamic Modules
##################################################

# path to dynamic preprocessor libraries
dynamicpreprocessor directory C:\Snort\lib\snort_dynamicpreprocessor

# path to base preprocessor engine
dynamicengine C:\Snort\lib\snort_dynamicengine\sf_engine.dll

# path to dynamic rules libraries (Shared Object (SO) Rules)
# Set this path to where the compiled *.so binaries are installed
# dynamicdetection directory C:\Snort\lib\snort_dynamicrules

##################################################
# Step #5: Configure preprocessors
# For more information, see the Snort Manual, Configuring Snort - Preprocessors
##################################################

# GTP Control Channle Preprocessor. For more information, see README.GTP
# preprocessor gtp: ports { 2123 3386 2152 }

# Inline packet normalization. For more information, see README.normalize
# Does nothing in IDS mode
#preprocessor normalize_ip4
#preprocessor normalize_tcp: block, rsv, pad, urp, req_urg, req_pay, req_urp, ips, ecn stream
##preprocessor normalize_icmp4
#preprocessor normalize_ip6
#preprocessor normalize_icmp6
```

Target-based IP defragmentation. For more inforation, see README.frag3

preprocessor frag3_global: max_frags 65536

preprocessor frag3_engine: policy windows detect_anomalies overlap_limit 10 min_fragment_length 100 timeout 180

Target-Based stateful inspection/stream reassembly. For more inforation, see README.stream5

preprocessor stream5_global: track_tcp yes, \

track_udp yes, \

track_icmp no, \

max_tcp 262144, \

max_udp 131072, \

max_active_responses 2, \

min_response_seconds 5

preprocessor stream5_tcp: policy windows, detect_anomalies, require_3whs 180, \

overlap_limit 10, small_segments 3 bytes 150, timeout 180, \

ports client 21 22 23 25 42 53 70 79 109 110 111 113 119 135 136 137 139 143 161 445 513 514 587 593 691 1433 1521 1741 2100 3306 6070 6665 6666 6667 6668 6669 7000 8181 32770 32771 32772 32773 32774 32775 32776 32777 32778 32779, \

ports both 36 80 81 82 83 84 85 86 87 88 89 90 110 311 323 383 443 444 465 555 563 591 593 623 631 636 664 801 808 818 901 972 989 992 993 994 995 1158 1220 1270 1414 1533 1581 1719 1720 1741 1801 1812 1830 1942 2231 2301 2375 2381 2578 2809 2869 2980 3000 3001 3029 3037 3057 3128 3300 3323 3443 3702 3901 4000 4343 4444 4848 5000 5054 5060 5061 5117 5222 5250 5416 5443 5450 5480 5555 5600 5814 5894 5984 5985 5986 6060 6080 6173 6988 7000 7001 7005 7070 7071 7080 7144 7145 7180 7181 7510 7770 7777 7778 7779 7801 7802 7900 7901 7902 7903 7904 7905 7906 7907 7908 7909 7910 7911 7912 7913 7914 7915 7916 7917 7918 7919 7920 8000 8001 8008 8014 8015 8020 8028 8040 8080 8081 8082 8085 8088 8090 8095 8118 8123 8161 8180 8181 8182 8222 8243 8280 8300 8333 8344 8393 8400 8443 8484 8500 8509 8511 8694 8787 8800 8848 8852 8880 8888 8899 8983 9000 9001 9002 9050 9060 9080 9090 9091 9111 9200 9201 9290 9443 9447 9502 9700 9710 9788 9830 9850 9990 9999 10000 10080 10100 10250 10255 10297 10443 11371 12601 13014 15489 15672 16000 16992 16993 16994 16995 17000 18081 19980 20000 29991 30007 30018 30888 33300 34412 34443 34444 36099 37215 40007 41080 44449 49152 49153 50000 50002 50452 51423 53331 54444 55252 55555 56712

preprocessor stream5_udp: timeout 180

performance statistics. For more information, see the Snort Manual, Configuring Snort - Preprocessors - Performance Monitor

preprocessor perfmonitor: time 300 file /var/snort/snort.stats pktcnt 10000

HTTP normalization and anomaly detection. For more information, see README.http_inspect

preprocessor http_inspect: global iis_unicode_map unicode.map 1252 compress_depth 65535 decompress_depth 65535

preprocessor http_inspect_server: server default \

http_methods { GET POST PUT SEARCH MKCOL COPY MOVE LOCK UNLOCK NOTIFY POLL BCOPY BDELETE BMOVE LINK UNLINK OPTIONS HEAD DELETE TRACE TRACK CONNECT SOURCE SUBSCRIBE UNSUBSCRIBE PROPFIND PROPPATCH BPROPFIND BPROPPATCH RPC_CONNECT PROXY_SUCCESS BITS_POST CCM_POST SMS_POST RPC_IN_DATA RPC_OUT_DATA RPC_ECHO_DATA } \

chunk_length 500000 \

server_flow_depth 0 \

client_flow_depth 0 \

post_depth 65495 \

oversize_dir_length 500 \

max_header_length 750 \

max_headers 100 \

max_spaces 200 \

small_chunk_length { 10 5 } \

ports { 36 80 81 82 83 84 85 86 87 88 89 90 311 323 383 443 444 555 591 593 623 631 664 801 808 818 901 972 1158 1220 1270 1414 1533 1581 1719 1720 1741 1801 1812 1830 1942 2231 2301 2375 2381 2578 2809 2869 2980 3000 3029 3037 3057 3128 3323 3443 3702 4000 4343 4444 4848 5000 5054 5060 5061 5117 5222 5250 5416 5443 5450 5480 5555 5600 5814 5894 5984 5985 5986 6060 6080 6173 6988 7000 7001 7005 7070 7071 7080 7144 7145 7180 7181 7510 7770 7777 7778 7779 8000 8001 8008 8014 8015 8020 8028 8040 8080 8081 8082 8085 8088 8090 8095 8118 8123 8161 8180 8181 8182 8222 8243 8280 8300 8333 8344 8393 8400 8443 8484 8500 8509 8511 8694 8787 8800 8848 8852 8880 8888 8899 8983 9000 9001 9002 9050 9060 9080 9090 9091 9111 9200 9201 9290 9443 9447 9502 9700 9710 9788 9830 9850 9990 9999 10000 10080 10100 10250 10255 10297 10443 11371 12601 13014 15489 15672 16000 16992 16993 16994 16995 17000 18081 19980 20000 29991 30007 30018 30888 33300 34412 34443 34444 36099 37215 40007 41080 44449 49152 49153 50000 50002 50452 51423 53331 54444 55252 55555 56712 } \

non_rfc_char { 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 } \

enable_cookie \

extended_response_inspection \

inspect_gzip \

normalize_utf \

unlimited_decompress \

normalize_javascript \

apache_whitespace no \

ascii no \

bare_byte no \

directory no \

double_decode no \

iis_backslash no \

iis_delimiter no \

iis_unicode no \

multi_slash no \

utf_8 no \

u_encode yes \
webroot no \
decompress_swf { deflate } \
decompress_pdf { deflate }

ONC-RPC normalization and anomaly detection. For more information, see the Snort Manual, Configuring Snort - Preprocessors - RPC Decode

preprocessor rpc_decode: 111 32770 32771 32772 32773 32774 32775 32776 32777 32778 32779 no_alert_multiple_requests no_alert_large_fragments no_alert_incomplete

Back Orifice detection.

preprocessor bo

FTP / Telnet normalization and anomaly detection. For more information, see README.ftptelnet

preprocessor ftp_telnet: global inspection_type stateful encrypted_traffic no check_encrypted

preprocessor ftp_telnet_protocol: telnet \
ayt_attack_thresh 20 \
normalize ports { 23 } \
detect_anomalies

preprocessor ftp_telnet_protocol: ftp server default \
def_max_param_len 100 \
ports { 21 2100 3535 } \
telnet_cmds yes \
ignore_telnet_erase_cmds yes \
ftp_cmds { ABOR ACCT ADAT ALLO APPE AUTH CCC CDUP } \
ftp_cmds { CEL CLNT CMD CONF CWD DELE ENC EPRT } \
ftp_cmds { EPSV ESTA ESTP FEAT HELP LANG LIST LPRT } \
ftp_cmds { LPSV MACB MAIL MDTM MIC MKD MLSD MLST } \
ftp_cmds { MODE NLST NOOP OPTS PASS PASV PBSZ PORT } \
ftp_cmds { PROT PWD QUIT REIN REST RETR RMD RNFR } \
ftp_cmds { RNTO SDUP SITE SIZE SMNT STAT STOR STOU } \
ftp_cmds { STRU SYST TEST TYPE USER XCUP XCRC XCWD } \
ftp_cmds { XMAS XMD5 XMKD XPWD XRCP XRMD XRSQ XSEM } \
ftp_cmds { XSEN XSHA1 XSHA256 } \
alt_max_param_len 0 { ABOR CCC CDUP ESTA FEAT LPSV NOOP PASV PWD QUIT REIN STOU SYST XCUP XPWD } \
alt_max_param_len 200 { ALLO APPE CMD HELP NLST RETR RNFR STOR STOU XMKD } \
alt_max_param_len 256 { CWD RNTO } \
alt_max_param_len 400 { PORT } \
alt_max_param_len 512 { SIZE } \
chk_str_fmt { ACCT ADAT ALLO APPE AUTH CEL CLNT CMD } \
chk_str_fmt { CONF CWD DELE ENC EPRT EPSV ESTP HELP } \
chk_str_fmt { LANG LIST LPRT MACB MAIL MDTM MIC MKD } \

```
    chk_str_fmt { MLSD MLST MODE NLST OPTS PASS PBSZ PORT } \
    chk_str_fmt { PROT REST RETR RMD RNFR RNTO SDUP SITE } \
    chk_str_fmt { SIZE SMNT STAT STOR STRU TEST TYPE USER } \
    chk_str_fmt { XCRC XCWD XMAS XMD5 XMKD XRCP XRMD XRSQ } \
    chk_str_fmt { XSEM XSEN XSHA1 XSHA256 } \
    cmd_validity ALLO < int [ char R int ] > \
    cmd_validity EPSV < [ { char 12 | char A char L char L } ] > \
    cmd_validity MACB < string > \
    cmd_validity MDTM < [ date nnnnnnnnnnnnnn[.n[n[n]]] ] string > \
    cmd_validity MODE < char ASBCZ > \
    cmd_validity PORT < host_port > \
    cmd_validity PROT < char CSEP > \
    cmd_validity STRU < char FRPO [ string ] > \
    cmd_validity TYPE < { char AE [ char NTC ] | char I | char L [ number ] } >
  preprocessor ftp_telnet_protocol: ftp client default \
    max_resp_len 256 \
    bounce yes \
    ignore_telnet_erase_cmds yes \
    telnet_cmds yes


  # SMTP normalization and anomaly detection.  For more information, see README.SMTP
  preprocessor smtp: ports { 25 465 587 691 } \
    inspection_type stateful \
    b64_decode_depth 0 \
    qp_decode_depth 0 \
    bitenc_decode_depth 0 \
    uu_decode_depth 0 \
    log_mailfrom \
    log_rcptto \
    log_filename \
    log_email_hdrs \
    normalize cmds \
    normalize_cmds { ATRN AUTH BDAT CHUNKING DATA DEBUG EHLO EMAL ESAM ESND
ESOM ETRN EVFY } \
    normalize_cmds { EXPN HELO HELP IDENT MAIL NOOP ONEX QUEU QUIT RCPT RSET SAML
SEND SOML } \
    normalize_cmds { STARTTLS TICK TIME TURN TURNME VERB VRFY X-ADAT X-DRCP X-
ERCP X-EXCH50 } \
    normalize_cmds { X-EXPS X-LINK2STATE XADR XAUTH XCIR XEXCH50 XGEN XLICENSE
XQUE XSTA XTRN XUSR } \
    max_command_line_len 512 \
```

```
    max_header_line_len 1000 \
    max_response_line_len 512 \
    alt_max_command_line_len 260 { MAIL } \
    alt_max_command_line_len 300 { RCPT } \
    alt_max_command_line_len 500 { HELP HELO ETRN EHLO } \
    alt_max_command_line_len 255 { EXPN VRFY ATRN SIZE BDAT DEBUG EMAL ESAM ESND ESOM EVFY IDENT NOOP RSET } \
    alt_max_command_line_len 246 { SEND SAML SOML AUTH TURN ETRN DATA RSET QUIT ONEX QUEU STARTTLS TICK TIME TURNME VERB X-EXPS X-LINK2STATE XADR XAUTH XCIR XEXCH50 XGEN XLICENSE XQUE XSTA XTRN XUSR } \
    valid_cmds { ATRN AUTH BDAT CHUNKING DATA DEBUG EHLO EMAL ESAM ESND ESOM ETRN EVFY } \
    valid_cmds { EXPN HELO HELP IDENT MAIL NOOP ONEX QUEU QUIT RCPT RSET SAML SEND SOML } \
    valid_cmds { STARTTLS TICK TIME TURN TURNME VERB VRFY X-ADAT X-DRCP X-ERCP X-EXCH50 } \
    valid_cmds { X-EXPS X-LINK2STATE XADR XAUTH XCIR XEXCH50 XGEN XLICENSE XQUE XSTA XTRN XUSR } \
    xlink2state { enabled }

  # Portscan detection.  For more information, see README.sfportscan
  preprocessor sfportscan: proto  { all } memcap { 10000000 } sense_level { low }

  # ARP spoof detection.  For more information, see the Snort Manual - Configuring Snort - Preprocessors - ARP Spoof Preprocessor
  # preprocessor arpspoof
  # preprocessor arpspoof_detect_host: 192.168.40.1 f0:0f:00:f0:0f:00

  # SSH anomaly detection.  For more information, see README.ssh
  preprocessor ssh: server_ports { 22 } \
          autodetect \
          max_client_bytes 19600 \
          max_encrypted_packets 20 \
          max_server_version_len 100 \
          enable_respoverflow enable_ssh1crc32 \
          enable_srvoverflow enable_protomismatch

  # SMB / DCE-RPC normalization and anomaly detection.  For more information, see README.dcerpc2
  preprocessor dcerpc2: memcap 102400, events [co ]
  preprocessor dcerpc2_server: default, policy WinXP, \
    detect [smb [139,445], tcp 135, udp 135, rpc-over-http-server 593], \
    autodetect [tcp 1025:, udp 1025:, rpc-over-http-server 1025:], \
```

```
    smb_max_chain 3, smb_invalid_shares ["C$", "D$", "ADMIN$"]

    # DNS anomaly detection.  For more information, see README.dns
    preprocessor dns: ports { 53 } enable_rdata_overflow

    # SSL anomaly detection and traffic bypass.  For more information, see README.ssl
    preprocessor ssl: ports { 443 465 563 636 989 992 993 994 995 5061 7801 7802 7900 7901 7902 7903
7904 7905 7906 7907 7908 7909 7910 7911 7912 7913 7914 7915 7916 7917 7918 7919 7920 8443 }, trustservers,
noinspect_encrypted

    # SDF sensitive data preprocessor.  For more information see README.sensitive_data
    preprocessor sensitive_data: alert_threshold 25

    # SIP Session Initiation Protocol preprocessor.  For more information see README.sip
    preprocessor sip: max_sessions 40000, \
      ports { 5060 5061 5600 }, \
      methods { invite \
            cancel \
            ack \
            bye \
            register \
            options \
            refer \
            subscribe \
            update \
            join \
            info \
            message \
            notify \
            benotify \
            do \
            qauth \
            sprack \
            publish \
            service \
            unsubscribe \
            prack }, \
      max_uri_len 512, \
      max_call_id_len 80, \
      max_requestName_len 20, \
      max_from_len 256, \
      max_to_len 256, \
```

```
  max_via_len 1024, \
  max_contact_len 512, \
  max_content_len 2048

# IMAP preprocessor.  For more information see README.imap
preprocessor imap: \
  ports { 143 } \
  b64_decode_depth 0 \
  qp_decode_depth 0 \
  bitenc_decode_depth 0 \
  uu_decode_depth 0

# POP preprocessor. For more information see README.pop
preprocessor pop: \
  ports { 110 } \
  b64_decode_depth 0 \
  qp_decode_depth 0 \
  bitenc_decode_depth 0 \
  uu_decode_depth 0

# Modbus preprocessor. For more information see README.modbus
preprocessor modbus: ports { 502 }

# DNP3 preprocessor. For more information see README.dnp3
preprocessor dnp3: ports { 20000 } \
  memcap 262144 \
  check_crc

# Reputation preprocessor. For more information see README.reputation
#preprocessor reputation: \
#   memcap 500, \
#   priority whitelist, \
#   nested_ip inner, \
#   whitelist $WHITE_LIST_PATH\whitelist.rules, \
#   blacklist $BLACK_LIST_PATH\blacklist.rules

##################################################
# Step #6: Configure output plugins
# For more information, see Snort Manual, Configuring Snort - Output Modules
##################################################

# unified2
```

```
# Recommended for most installs
# output unified2: filename merged.log, limit 128, nostamp, mpls_event_types, vlan_event_types

# Additional configuration for specific types of installs
# output alert_unified2: filename snort.alert, limit 128, nostamp
# output log_unified2: filename snort.log, limit 128, nostamp

# syslog
# output alert_syslog: LOG_AUTH LOG_ALERT

# pcap
# output log_tcpdump: tcpdump.log

# metadata reference data.  do not modify these lines
include classification.config
include reference.config


##################################################
# Step #7: Customize your rule set
# For more information, see Snort Manual, Writing Snort Rules
#
# NOTE: All categories are enabled in this conf file
##################################################

# site specific rules
include $RULE_PATH\local.rules

include $RULE_PATH\app-detect.rules
include $RULE_PATH\attack-responses.rules
include $RULE_PATH\backdoor.rules
include $RULE_PATH\bad-traffic.rules
include $RULE_PATH\blacklist.rules
include $RULE_PATH\botnet-cnc.rules
include $RULE_PATH\browser-chrome.rules
include $RULE_PATH\browser-firefox.rules
include $RULE_PATH\browser-ie.rules
include $RULE_PATH\browser-other.rules
include $RULE_PATH\browser-plugins.rules
include $RULE_PATH\browser-webkit.rules
include $RULE_PATH\chat.rules
include $RULE_PATH\content-replace.rules
```

include $RULE_PATH\ddos.rules
include $RULE_PATH\dns.rules
include $RULE_PATH\dos.rules
include $RULE_PATH\experimental.rules
include $RULE_PATH\exploit-kit.rules
include $RULE_PATH\exploit.rules
include $RULE_PATH\file-executable.rules
include $RULE_PATH\file-flash.rules
include $RULE_PATH\file-identify.rules
include $RULE_PATH\file-image.rules
include $RULE_PATH\file-java.rules
include $RULE_PATH\file-multimedia.rules
include $RULE_PATH\file-office.rules
include $RULE_PATH\file-other.rules
include $RULE_PATH\file-pdf.rules
include $RULE_PATH\finger.rules
include $RULE_PATH\ftp.rules
include $RULE_PATH\icmp-info.rules
include $RULE_PATH\icmp.rules
include $RULE_PATH\imap.rules
include $RULE_PATH\indicator-compromise.rules
include $RULE_PATH\indicator-obfuscation.rules
include $RULE_PATH\indicator-scan.rules
include $RULE_PATH\indicator-shellcode.rules
include $RULE_PATH\info.rules
include $RULE_PATH\malware-backdoor.rules
include $RULE_PATH\malware-cnc.rules
include $RULE_PATH\malware-other.rules
include $RULE_PATH\malware-tools.rules
include $RULE_PATH\misc.rules
include $RULE_PATH\multimedia.rules
include $RULE_PATH\mysql.rules
include $RULE_PATH\netbios.rules
include $RULE_PATH\nntp.rules
include $RULE_PATH\oracle.rules
include $RULE_PATH\os-linux.rules
include $RULE_PATH\os-mobile.rules
include $RULE_PATH\os-other.rules
include $RULE_PATH\os-solaris.rules
include $RULE_PATH\os-windows.rules
include $RULE_PATH\other-ids.rules
include $RULE_PATH\p2p.rules

include $RULE_PATH\phishing-spam.rules
include $RULE_PATH\policy-multimedia.rules
include $RULE_PATH\policy-other.rules
include $RULE_PATH\policy.rules
include $RULE_PATH\policy-social.rules
include $RULE_PATH\policy-spam.rules
include $RULE_PATH\pop2.rules
include $RULE_PATH\pop3.rules
include $RULE_PATH\protocol-dns.rules
include $RULE_PATH\protocol-finger.rules
include $RULE_PATH\protocol-ftp.rules
include $RULE_PATH\protocol-icmp.rules
include $RULE_PATH\protocol-imap.rules
include $RULE_PATH\protocol-nntp.rules
include $RULE_PATH\protocol-other.rules
include $RULE_PATH\protocol-pop.rules
include $RULE_PATH\protocol-rpc.rules
include $RULE_PATH\protocol-scada.rules
include $RULE_PATH\protocol-services.rules
include $RULE_PATH\protocol-snmp.rules
include $RULE_PATH\protocol-telnet.rules
include $RULE_PATH\protocol-tftp.rules
include $RULE_PATH\protocol-voip.rules
include $RULE_PATH\pua-adware.rules
include $RULE_PATH\pua-other.rules
include $RULE_PATH\pua-p2p.rules
include $RULE_PATH\pua-toolbars.rules
include $RULE_PATH\rpc.rules
include $RULE_PATH\rservices.rules
include $RULE_PATH\scada.rules
include $RULE_PATH\scan.rules
include $RULE_PATH\server-apache.rules
include $RULE_PATH\server-iis.rules
include $RULE_PATH\server-mail.rules
include $RULE_PATH\server-mssql.rules
include $RULE_PATH\server-mysql.rules
include $RULE_PATH\server-oracle.rules
include $RULE_PATH\server-other.rules
include $RULE_PATH\server-samba.rules
include $RULE_PATH\server-webapp.rules
include $RULE_PATH\shellcode.rules
include $RULE_PATH\smtp.rules

```
include $RULE_PATH\snmp.rules
include $RULE_PATH\specific-threats.rules
include $RULE_PATH\spyware-put.rules
include $RULE_PATH\sql.rules
include $RULE_PATH\telnet.rules
include $RULE_PATH\tftp.rules
include $RULE_PATH\virus.rules
include $RULE_PATH\voip.rules
include $RULE_PATH\web-activex.rules
include $RULE_PATH\web-attacks.rules
include $RULE_PATH\web-cgi.rules
include $RULE_PATH\web-client.rules
include $RULE_PATH\web-coldfusion.rules
include $RULE_PATH\web-frontpage.rules
include $RULE_PATH\web-iis.rules
include $RULE_PATH\web-misc.rules
include $RULE_PATH\web-php.rules
include $RULE_PATH\x11.rules

##################################################
# Step #8: Customize your preprocessor and decoder alerts
# For more information, see README.decoder_preproc_rules
##################################################

# decoder and preprocessor event rules
include $PREPROC_RULE_PATH\preprocessor.rules
include $PREPROC_RULE_PATH\decoder.rules
include $PREPROC_RULE_PATH\sensitive-data.rules
```

Министерство науки и высшего образования РФ

Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой
_____ О.В. Непомнящий
«23» 06 2023 г.

БАКАЛАВРСКАЯ РАБОТА

090301 Информатика и вычислительная техника

Система анализа пограничного трафика на основе глубокого анализа пакетов

Руководитель	_____ подпись	22.06.23 дата	доцент, канд. техн. наук должность, ученая степень	Ф.А. Казаков
Выпускник	_____ подпись	22.06.23 дата		Ю.А. Коптев
Нормоконтролёр	_____ подпись	22.06.23 дата		Ф.А. Казаков

Красноярск 2023