Разработка и анализ эффективности средств отражения распределенных атак

СОДЕРЖАНИЕ

СОКРАЩЕНИЯ И УСЛОВНЫЕ ОБОЗНАЧЕНИЯ

ВВЕДЕНИЕ

1. ОПИСАНИЕ ИС

1.1 Описание ИС

1.2 Модель нарушителя

1.3 Модель угроз

1.3.1. Классификация угроз в соответствии с IT-Baseline Protection Manual

1.3.1.1 Угрозы, связанные с форс-мажорными обстоятельствами

## 1.3.1.2 Угрозы, связанные с недостатками организации и управления

1.3.1.3 Угрозы, связанные с человеческим фактором

1.3.1.4 Угрозы, связанные с техническими неисправностями

1.3.1.5 Угрозы, связанные со спланированными действиями нарушителей

1.3.2 Классификация угроз по нарушаемым базовым услугам ИС

1.3.2.1 Угрозы нарушения конфиденциальности информации

1.3.2.2 Угрозы нарушения целостности информации

1.3.2.3 Угрозы нарушения аутентичности

1.3.2.4 Угрозы нарушения наблюдаемости

1.3.2.5 Угрозы нарушения доступности ресурсов

1.4 Особенности реализации DoS/DDos атак. TCP SYN атака

1.5 Постановка задач по защите от угроз

2. ИЗВЕСТНЫЕ МЕТОДЫ ПРОТИВОДЕЙСТВИЯ TCP SYN АТАКЕ

2.1 TCP SYN cookies

2.2 TCP RST cookies

2.3 Floodgate

2.4 Предмаршрутизационная фильтрация

2.5 Random/Old Drop

2.6 Syn-Proxy

2.7 Stack tweaking

2.8 BlackListing

3. МАТЕМАТИЧЕСКАЯ МОДЕЛЬ TCP SYN АТАКИ

3.1 Краткие сведения из теории систем массового обслуживания

3.2 Поток требования СМО

3.3. Сервер TCP соединения как СМО

3.4 СМО с бесконечным количеством обслуживающих приборов

3.5 Модель, учитывающая потерю пакетов в сети

4. МЕТОДИКИ СБОРА ДАННЫХ

4.1 Определение времени прохождения IP пакета по сети Internet

4.2 Определение вероятности потери пакетов в сети

4.3 Определение интенсивности входящего потока требований

5. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

5.1 Особенности установки Snort

5.2 Внутренняя структура Snort

5.2.1 Препроцессоры

5.2.2 Модули обнаружения

5.2.3 Модули вывода

5.3 Разработка модуля обнаружения

5.3.1 Структура модуля TcpConnEstTimeChecker

5.3.2 Структура модуля TcpSynFloodPreventionModule

5.3.3 Взаимодействие TcpConnEstTimeChecker и TcpSynFloodPreventionModule в реализации tcp\_syn\_flood

ВЫВОДЫ

ПЕРЕЧЕНЬ ССЫЛОК

ПРИЛОЖЕНИЯ

СОКРАЩЕНИЯ И УСЛОВНЫЕ ОБОЗНАЧЕНИЯ

БД – базы данных

ДСТСЗИ СБУ – Департамент специальных телекоммуникационных систем и защиты информации Службы Безопасности Украины (Державна служба спеціального зв'язку та захисту інформації України)

# ИС – информационная система

ИТС – информационно-телекомуникационная система

КЗИ – комплексная защита информации

КС – компьютерная система

ОС – операционная система

ОЗУ – оперативное запоминающее устройство

ПО – программное обеспечение

СВ – случайная величина

СМО – система массового обслуживания

ACK – Acknowledgement Flag

ARP – Address Resolution Protocol

ASN.1 – Abstract Syntax Notation One

BO - Back Orifice

CSV – Coma Separated Values

DDoS – Distributed Denial of Service

DNS – Domain Name Service

DoS – Denial of Service

DSL – Digital Subscriber Line

FIN – Finalization Flag

ICMP – Internet Control Message Protocol

IDS - Intrusion Detection System

IPS – Intrusion Prevention System

IP - Internet Protocol

HTTP – Hyper Text Transfer Protocol

MSS – Maximum Segment Size

OSI – Open System Interconnection

PSH – Push Flag

RPC – Remote Procedure Call

RST – Reset Flag

SMS – Simple Message Service

SYN – Synchronize Flag

TCP – Transmission Control Protocol

TTL – Time to Live

UDP – User Data Protocol

URG – Urgent Flag

WWW – World Wide Web

ВВЕДЕНИЕ

В настоящее время, трудно себе представить успешную компанию, не использующую для организации делопроизводства достижения науки и техники в сфере информационных технологий. Также интеграции современных технологий способствует развитие в Украине соответствующей нормативной базы. Одним из перспективных направлений является развитие электронного документооборота, инфраструктуры открытых ключей, использование средств современной криптографии органами государственной власти, органами местного самоуправления, предприятиями, учреждениями, организациями и т.д. [1,2]. Одним из основных преимуществ использования таких технологий является значительное повышение эффективности делопроизводства за счет ускорения поиска необходимой информации, увеличения скорости обмена информацией, уменьшение процента утерянной информации и т.д. Такие средства криптографии как электронная цифровая подпись способны обеспечить обеспечение таких базовых услуг информационных систем как конфиденциальность, целостность информации и т.д [3]. Однако использование этих средств не дает гарантии обеспечения такой важной услуги, как доступность ресурсов [3]. Для того чтобы инфраструктура открытых ключей могла полноценно функционировать, необходимо, чтобы пользователи системы всегда могли иметь доступ к центрам сертификации различных уровней.

К сожалению, в последнее время все большее распространение получил целый класс атак (DoS/DDoS), направленных на отказ в обслуживании. Успешная реализация таких атак позволяет блокировать доступ пользователей информационных систем к ресурсам различных серверов, что может вывести из рабочего состояния всю систему.

Одним из способов реализации атаки типа отказа в обслуживании является загрузка всех ресурсов сервера обработкой огромного количества ложных запросов. В большинстве случаев, для организации таких атак используются компьютеры "мирных" пользователей без их ведома и согласия. Это осуществляется путем установки на недостаточно защищенные машины вредоносного программного обеспечения, такого как "троянские кони", "черви", компьютерные вирусы и т.д. После того, как злоумышленник смог инфицировать достаточно большое количество узлов сети, реализация распределенной атаки сводится к тому, чтобы отправить одновременно всем зараженным машинам команду, активирующую вредоносное ПО, превращая тем самым "мирные" компьютеры в источник распределенной атаки.

В настоящее время, организация атак типа отказа в обслуживании является довольно прибыльным занятием. Существуют два способа получить финансовое вознаграждение. В первом случае за организацию атак платят нечистоплотные конкуренты, целью которых является нанесение удара по репутации других компаний и организаций. А в другом случае хакеры занимаются вымогательством – выбирают жертву, атакуют ее и требуют выкуп за прекращение атаки.

Для эффективного противодействия DDoS атакам существуют несколько серьезных препятствий. В частности, многие из них не требуют установления сеанса связи с источником атаки, что значительно затрудняет поиск злоумышленника. Но даже в случае поимки хакера, он может не получить заслуженного наказания в виду несовершенства законодательства в различных странах [4].

На данный момент в мире существуют решения, снижающие негативное воздействие DDoS атак (может отфильтровываться до 99% вредоносного трафика [5]), но такие средства обладают несколькими недостатками, которые ограничивают возможности их использования. Во-первых, использующиеся алгоритмы являются коммерческой тайной разработчиков, что не дает возможности соответствующим организациям сертифицировать эти продукты. Вторым недостатком является высокая цена, недоступная для многих компаний и организаций. Например, услуги компаний AT&T и MCI в этой области стоят около 12 тысяч долларов в месяц[5].

Из всего вышесказанного видно, что проблема обнаружения и противодействия DDoS атакам является актуальной и требует недорогих и эффективных решений. В магистерской работе предлагается методика раннего обнаружения одной из самых распространенных DDoS атак – TCP SYN атаки. В основе этой методики лежит математическая модель, описывающая взаимодействие сервера с клиентами. Разработанная модель учитывает индивидуальные значения различных параметров, характеризующих работу сети и сервера, что повышает эффективность обнаружения атаки. В работе так же предлагается программная реализация разработанной методики, представленная в виде модуля расширения функциональности для системы предотвращения вторжений Snort\_inline. Такое решение ориентировано на защиту критичных ресурсов корпоративной сети от указанной выше атаки.

1. ОПИСАНИЕ ИС

# 1.1 Описание ИС

Как известно, XXI век принято считать веком высоких технологий. Трудно себе представить современное общество, в котором не использовались бы последние достижения стремительно развивающейся науки и техники. Одним из направлений развития современной науки являются информационные технологии, с каждым днем открывающие перед человечеством все новые и новые возможности. Одним из замечательных продуктов этих технологий являются информационные системы (ИС), повсеместно внедряемые во всяческие сферы человеческой деятельности.

Далее под информационной системой будем понимать объединенную совокупность аппаратных, программно-аппаратных и программных средств, осуществляющих создание, хранение, обработку и уничтожение разнообразной информации, а так же обмен ею путем взаимодействия между собой [6].

Наиболее известным примером информационной системы является глобальная сеть Internet. Она охватывает своей паутиной практически всю поверхность земного шара, открывая для человечества колоссальные возможности. Фактически Internet состоит из множества локальных и глобальных сетей, принадлежащих различным компаниям и предприятиям, связанных между собой различными линиями связи. Internet можно представить себе в виде мозаики сложенной из небольших сетей разной величины, которые активно взаимодействуют одна с другой, пересылая различную информацию [7].

При низкой стоимости услуг (часто это только фиксированная ежемесячная плата за используемые линии или телефон) пользователи могут получить доступ к коммерческим и некоммерческим информационным службам практически всех мировых стран. В архивах свободного доступа сети Internet можно найти информацию практически по всем сферам человеческой деятельности, начиная с новых научных открытий до прогноза погоды на ближайший месяц.

Internet предоставляет уникальные возможности дешевой и надежной связи по всему миру. Это оказывается очень удобным для фирм имеющих свои филиалы по всему миру, транснациональных корпораций и структур управления. Обычно, использование инфраструктуры Internet для международной связи обходится значительно дешевле прямой компьютерной связи через спутниковый канал или через телефон.

Электронная почта – одна из самых распространенных услуг сети Internet. Посылка письма по электронной почте обходится значительно дешевле посылки обычного письма. Кроме того, сообщение, посланное по электронной почте, дойдет до адресата за короткий промежуток времени (от нескольких десятков секунд), в то время как обычное письмо может добираться до адресата несколько дней и недель.

Еще одной распространенной услугой Internet является World Wide Web (WWW) - система для работы с гипертекстом. Потенциально она является наиболее мощным средством поиска. Гипертекст соединяет различные документы на основе заранее заданного набора слов. Например, когда в тексте встречается новое слово или понятие, система, работающая с гипертекстом, дает возможность перейти к другому документу, в котором это слово или понятие рассматривается более подробно. WWW часто используется в качестве дружественного интерфейса к базам данных [6].

С каждым днем растет зависимость успешной работы компаний от использования современных информационных технологий, которые предоставляют грандиозные возможности. На рис. 1.1 приведен пример типичной корпоративной сети компании, имеющей разветвленную структуру географически удаленных друг от друга филиалов, [8].

Рис.1.1 Пример ИС

Такая ИС позволяет, например, проводить конференции с участием сотрудников, работающих в разных городах и странах, с такой же легкостью как если бы они находились в одном помещении. В таких системах обрабатывается информация различного характера и содержания. Это может быть жизненно важная для компании информация, например о коммерческой деятельности, результаты научных исследований, на которые было затрачено много ресурсов и времени и т.д. Очевидно, что, например, ознакомление с такими сведениями конкурентов, может привести к непоправимым последствиям, вплоть до банкротства. В связи с этим к информационной системе должны выдвигаться требования по обеспечению некоторых базовых услуг. Такие услуги будут рассмотрены ниже. Здесь стоит отметить, что причинами нарушения базовых услуг могут быть как обстоятельства случайного характера, так и специально спланированные действия нарушителей.

Эти обстоятельства могут иметь как случайный характер, так и являться следствием спланированных действий нарушителей. Классификация возможных нарушителей в ИС приведена в пункте 1.2.

1.2 Модель нарушителя

В соответствии с [3], нарушитель – это пользователь, который осуществляет несанкционированный доступ к информации. Здесь стоит отметить, что под несанкционированным доступом к информации может быть как ознакомление с ней таки и ее редактирование и удаление. В контексте данной работы это определение может быть расширено с учетом того, что злоумышленник может не быть пользователем атакуемой системы. В соответствии с нормативными документами, действующими в Украине, определены четыре уровня возможностей нарушителя в ИС системе [9]:

1. Нулевой уровень – случайное неспециальное ознакомление с содержанием информации.
2. Первый уровень – нарушитель, имеющий ограниченные средства и самостоятельно создающий средства и методы атак на средства КЗИ и ИТС, с применением широко распространенного ПО и вычислительной техники.
3. Второй уровень – нарушитель корпоративного типа. Имеет возможность создания специальных технических средств, стоимость которых соотносится с возможными финансовыми убытками при потере, искажении и уничтожении защищаемой информации. Для проведения вычислений могут использоваться локальные вычислительные сети.
4. Третий уровень – нарушитель имеет научно-технический ресурс, приравниваемый к научно-техническому ресурсу экономически развитого государства.

Методика, предлагаемая в данной магистерской работе, ориентирована на защиту от злоумышленников нулевого, первого и второго уровней.

В связи с тем, что злоумышленники третьего уровня обладают практически неограниченными возможностями, защита от реализуемых ими угроз выходит за рамки данной работы.

В пункте 1.3 приведена модель возможных в рассматриваемой ИС угроз.

1.3 Модель угроз

В соответствии с [3], угроза – это любые обстоятельства или события, которые могут быть причиной нарушения политики безопасности и (или) нанесения ущерба. Ущерб может заключаться в нарушении свойств информации путем ее разрушения, искажения или несанкционированного ознакомления, либо в разрушении, искажении или несанкционированном использовании ресурсов системы. Источниками угроз могут быть различные объекты и явления, что значительно затрудняет их учет при построении комплексной системы защиты информации. В связи с этим в мировой практике принято строить модель угроз, в которой приводится классификация возможных угроз, их описание и средства возможной реализации.

1.3.1 Классификация угроз в соответствии с IT-Baseline Protection Manual

Одним из лучших документов в этой области классификации угроз является [10]. В этом стандарте приводится перечень возможных угроз, а так же рекомендуются организационные и технические меры для защиты от них. В приведенной в [10] классификации все угрозы разделены на 5 основных групп:

1. Угрозы, связанные с форс-мажорными обстоятельствами
2. Угрозы, связанные с недостатками организации и управления
3. Угрозы, связанные с человеческим фактором
4. Угрозы, связанные с техническими неисправностями.
5. Угрозы, связанные со спланированными действиями злоумышленников.

Каждая из этих групп содержит большой перечень угроз, подробное рассмотрение которых выходит за пределы этой работы, поэтому ниже приведены только некоторые примеры, дающие представление о разнообразности угроз.

1.3.1.1 Угрозы, связанные с форс-мажорными обстоятельствами

Рассматриваемые в этой группе угрозы характеризуются тем, что их источники трудно заранее предсказать и их проявление носит случайных характер. Одной из таких угроз являются проблемы с персоналом. Ущерб в этом случае проявляется в том, что болезнь, смерть или забастовки персонала могут привести к прерыванию выполнения критичных задач, или выходу из строя критичных ресурсов. Источниками целого ряда угроз являются ЧП как природного (молнии, пожары, наводнения, дожди, магнитные бури) так и техногенного (возгорание кабелей, аварии, нарушение систем тепло-, водо-, и электроснабжения) характера. Стоит отметить, что такого рода угрозы могут наносить как непосредственный, так и косвенный ущерб. Например, во время пожара оборудование может подвергаться деструктивному воздействию не только от контакта с открытым огнем, но и под действием газовых смесей, образующихся при горении.

Так же к угрозам этой группы относятся недопустимые температуры и влажность, пыль и грязь, который могут привести к выходу из строя некоторых ресурсов информационной системы.

Для систем, подсистемы которых тесно связаны между собой, серьезными угрозами являются отказы и сбои в системе. Отказ в работе одного из компонентов может повлечь за собой сбой работы всего механизма. Примером такой угрозы может быть резкий скачок напряжения в электросети, результатом которого может быть выход из строй блока питания одного из критичных ресурсов системы. В результате этот ресурс может стать недоступным для всей системы на длительное время. Еще одним примером являются распределенных ИТ систем, в которых время является критичным ресурсом. В этом случае сбои в работе WAN могут привести к сбоям работы всей системы или ее компонентов.

1.3.1.2 Угрозы, связанные с недостатками организации и управления

Угрозы этой группы характеризуются тем, что их источником являются недоработки организационного характера. К таким угрозам относятся отсутствие или несовершенство регламентирующих правил и документов, недостаточное ознакомление с ними персонала, низкокачественный мониторинг и аудит мер ИТ безопасности. Например, если в политике безопасности не запрещено использование неучтенных носителей данных, или служащий не ознакомлен с этим положением, то значительно повышается вероятность попадания в корпоративную сеть вредоносного ПО (ad-ware – вирусов, троянских коней и т.д.), принесенного сотрудником из дома на flash диске. Еще одним примером этого вида угроз является недостаточное финансирование, либо неэффективное использование ресурсов. Например, необходимость использования старых версий ОС (таких как Windows 95) приводит к невозможности аудита действий пользователей.

Так же к угрозам этой группы относятся ошибки в проектировании системы. Например, неправильный расчет пропускной способности канала связи приведет либо к необоснованным затратам на услуги связи, либо к замедлению работы других компонентов. Стоит упомянуть угрозы связанные с тестированием. К ним относятся некачественное тестирование и тестирование с использованием реальных данных. В первом случае в эксплуатацию может поступить нестабильная и (или) неправильно работающая система, а во втором может иметь место нарушение конфиденциальности данных.

Другими примерами таких угроз являются неавторизированный доступ в помещение, использование ресурсов, использование полномочий и т.д.

1.3.1.3 Угрозы, связанные с человеческим фактором

Источниками угроз этой группы являются некорректные действия пользователей системы. Вследствие таких действий возможна потеря конфиденциальности или целостности данных. Например, если пользователь распечатал информацию на принтере и забыл забрать распечатку, то ее содержание может стать доступно не имеющим на это прав пользователям. Потеря целостности возможна в случае неправильной настройки прав доступа к файлам. Еще одним примером угрозы этой группы является неполное следование мерам безопасности. Например, хранение носителей информации в закрытом ящике стола еще не гарантирует достаточной защиты от неавторизованного доступа. Если ключ от ящика находится в офисе в общедоступном месте – например на столе. Неаккуратные действия пользователей и не целевое использование системы могут привести к потере данных, порче оборудования и т.д.

Большая часть угроз этой группы связана с некорректным администрированием и настройкой системы. Например, неправильное соединение кабелей, настройка сетевых устройств может привести к тому, что данные будут дополнительно передаваться другим адресатам. В случае неправильной настройки прикладных сервисов? таких как Web, SMTP, POP3, RAS и т.д. так же возможно нарушение конфиденциальности и целостности информации.

1.3.1.4 Угрозы, связанные с техническими неисправностями.

Источниками угроз этой группы являются различного рода технические неисправности. Это могут быть сбои в сетях электро-, тепло-, водоснабжения, в противопожарных системах и системах кондиционирования и вентиляции, в сетях связи и т.д. В результате реализации таких угроз может выйти из строя оборудование, что в свою очередь может привести к остановке функционирования всей системы, или ее отдельных компонентов. Так же примерами такого рода угроз могут быть неисправности оборудования, носителей информации, ошибки при передаче данных.

Так же к этой группе относятся ошибки и уязвимости ПО. Например, в некоторых широко распространенных приложениях есть функциональность позволяющая применять криптопреобразования к документам и т.д. Однако эта функциональность реализует нестойкие алгоритмы, защиту которых можно легко обойти с помощью легкодоступных в Интернете утилит. Так же широко известной разновидностью уязвимостей ПО является переполнение буфера. Эти уязвимости используют тот факт, что разработчики довольно часто не ограничивают размер вводимых пользователями данных. Это может привести к тому, что введенные пользователем данные могут быть интерпретированы как исполняемый код.

Еще одна интересная угроза связана с настройками CD привода. В случае, когда включена опция автоматического определения диска, автоматически выполняется файл AUTORUN.INF, содержание которого заранее неизвестно.

1.3.1.5 Угрозы, связанные со спланированными действиями злоумышленников

Эта группа содержит наибольшее количество угроз. В ней представлены угрозы, которые могут быть реализованы как при непосредственном физическом воздействии (кражи, акты вандализма и т.д.) так и удаленно.

Одним из примеров таких угроз является действие так называемого adware ПО, к которому относятся компьютерные вирусы, черви, троянские кони, макро вирусы, rootkit’ы и т.д.

Так же существуют угрозы, связанные с анализом сетевого трафика с помощью специальных программ (sniffers). При реализации этих угроз злоумышленник может получить несанкционированный доступ к конфиденциальной информации: e-mail, паролям (которые во многих протоколах передаются по сети в открытом виде). Так же с помощью сетевого трафика злоумышленник может получить представление о внутренней структуре сети (используемые IP адреса, топологию и т.д.), ПО используемое на машинах в сети и т.д. Если злоумышленник контролирует сетевое устройство (маршрутизатор, мост и т.д.) через которое проходит трафик, то кроме нарушения конфиденциальности информации может присутствовать нарушение ее целостности.

Стоит упомянуть угрозы связанные со спуфингом (spoofing). Примерами таких угроз являются ARP и DNS spoofing. Реализация этих угроз позволяет изменить маршрутизацию пакетов в локальных (ARP, DNS) и глобальных (DNS) сетях, в результате чего злоумышленник получает возможность контролировать трафик атакуемых систем. Так же существует угроза Web spoofing. Она заключается в том, что злоумышленник может имитировать какой либо Web сайт путем регистрации доменного имени со схожим названием, при этом многие пользователи об этом даже не будут догадываться. Например, у компании "Рога и Копыта Ukraine" есть web сайт www.rogakopita.ua. Если пользователи не знают точного доменного имени? то многие из них начнут поиск с [www.rogakopita.com](http://www.rogakopita.com), по которому злоумышленники могут разместить похожий сайт.

1.3.2 Классификация угроз по нарушаемым базовым услугам ИС

Как упоминалось ранее, к ИС должны выдвигаться требования по обеспечению базовых услуг. Такими базовыми услугами являются [3]:

* обеспечение конфиденциальности информации. Конфиденциальность – это свойство информации, заключающееся в том, что она не может быть получена неавторизованным пользователем и (или) процессом [3].
* обеспечение целостности информации. Целостность – это свойство информации, заключающееся в том, что она не может быть модифицирована неавторизованным пользователем и (или) процессом [3].
* обеспечение доступности ресурсов. Доступность – это свойство ресурса системы, заключающееся в том, что пользователь и (или) процесс, который владеет соответствующими полномочиями, может использовать ресурс в соответствии с правилами, установленными политикой безопасности, не ожидая дольше заданного (маленького) промежутка времени, т.е. когда они находятся в виде нужном пользователю, в месте нужном пользователю и в нужное пользователю время [3]
* обеспечение аутентичности. Обеспечивается с помощью аутентификации. Аутентификация – это процедура проверки соответствия предъявленного идентификатора объекта КС на предмет принадлежности его этому объекту [3].
* обеспечение наблюдаемости. Наблюдаемость – это свойство КС, которое позволяет фиксировать деятельность пользователей и процессов, использование пассивных объектов, а так же однозначно устанавливать идентификаторы причастных к конкретным событиям пользователей и процессов с целью предотвращения нарушения политики безопасности и (или) обеспечения ответственности за конкретные действия [3].

Так же угрозы могут быть классифицированы по принципу, к нарушению какой из базовых услуг ИС они приводят. Здесь можно выделить угрозы нарушающие конфиденциальность, целостность и доступность информации и ресурсов, а так же аутентичности и наблюдаемости.

1.3.2.1 Угрозы нарушения конфиденциальности информации

Эта угрозы заключается том, что информация становится известной тем сторонам информационного взаимодействия, у которых нет прав на ознакомление с этой информацией. Примерами реализации такой угрозы могут быть:

* Перехват и анализ сетевого трафика с помощью специализированного ПО. Такое ПО называется снифферами.
* Криптоанализ зашифрованных данных
* Несанкционированный доступ к данным, находящимся в различных запоминающих устройствах (на жестком диске, в ОЗУ, flash и т.д).

1.3.2.2 Угрозы нарушения целостности информации

Угрозы нарушения целостности информации включают в себя несанкционированное изменение или удаление данных, обрабатываемых в информационной системе. Примерами реализации такой угрозы могут быть:

* Модификация трафика, проходящего через хост атакующего, либо с помощью специализированного ПО (троянские кони, rootkit)
* Модификация файлов других пользователей, хранящихся на разделяемом дисковом пространстве

1.3.2.3 Угрозы нарушения аутентичности

Угрозы нарушения аутентичности заключаются в том, что в результате проведения некоторых действий пользователь и (или) процесс выдает себя за другого пользователя и имеет возможность воспользоваться чужими правами и привелегиями. Примерами реализации такой угрозы являются:

* Навязывание ложных сетевых адресов (ARP-spoofing) и доменных имен (DNS-spoofing), соответственно, может осуществляться на сетевом и транспортном уровнях модели OSI.
* Классическая атака типа человек посередине (Man in the middle). Заключается в том, что злоумышленник незаметно внедряется в канал связи между двумя абонентами и получает полный контроль над информацией (модификация, удаления, создание дезинформации), которой обмениваются участвующие стороны. При этом он остается абсолютно невидимым для абонентов.

1.3.2.4 Угрозы нарушения наблюдаемости

Угрозы нарушения наблюдаемости заключаются в том, что в результате некоторых действий злоумышленника становится невозможным фиксирование деятельности пользователей и процессов, использования пассивных объектов, а так же однозначно устанавливать идентификаторы причастных к конкретным событиям пользователей и процессов. Примерами реализации таких атак может быть:

* Очистка журналов аудита
* Отключение системы аудита
* Переполнение журнала аудита, в результате чего, записи о некоторых событиях пропадают
* Заражение системы rootkit’ом

1.3.2.5 Угрозы нарушения доступности ресурсов

Угрозы нарушения доступности ресурсов заключаются в проведении некоторых действий, в результате которых блокируется доступ к некоторому ресурсу информационной системы со стороны легальных пользователей. Примерами реализации таких угроз могут быть:

* загрузка ресурсов сервера фиктивными запросами злоумышленника, в результате чего запросы от легальных пользователей не могут быть обработаны.
* обрыв канала связи, между взаимодействующими сторонами

Атаки, реализующие такие угрозы, называются DoS (Denial of Service) атаками. Целью данной работы является разработка методики обнаружения и противодействия TCP SYN атаке – одной из самых распространенных атак этого класса. В связи с этим ниже рассматриваются особенности реализации атак типа отказа в обслуживании и, в частности, TCP SYN атаки.

1.4 Особенности реализации DoS/DDos атак. TCP SYN атака

DoS/DDoS-атаки направлены на нарушение базовой услуги доступности. Основная цель DoS/DDoS-атак вывести атакуемый объект из рабочего состояния и сделать его ресурсы недоступными для легальных пользователей. Атаку, направленную на отказ в обслуживании, можно провести двумя способами: используя уязвимости в программном обеспечении атакуемой системы и при помощи отсылки большого количества определенно составленных сетевых пакетов (flood).

Первый способ сложнее и требует более высокой квалификации атакующего. Второй способ основан на применении "грубой силы". Идея заключается в том, чтобы загрузить вычислительные ресурсы сервера обработкой огромного количества посланных злоумышленником пакетов. Такая загрузка сервера в лучшем случае может привести к тому, что сервер будет неспособен принять на обработку запросы от легальных пользователей, а в худшем случае может привести к зависанию и отключению сервера.

Здесь стоит отметить, что можно выделить два типа атак, направленных на загрузку ресурсов системы: в первом случае загружаются вычислительные ресурсы сервера, а в другом – пропускная способность канала связи. Разрабатываемая методика ориентирована на защиту от атак первого типа, поэтому дальше будем считать, что пропускной способности достаточно, чтобы сервер получил весь адресованный ему трафик.

Для многих DoS/DDoS атак результаты обработки сервером пакетов, отправленных злоумышленником, последнего не интересуют. Это значит, что атакующий может отправлять поток ложных заявок с ложных IP адресов (это понятие называется spoofing), что препятствует его обнаружению и эффективному противодействию такого рода атакам.

Для проведения успешной DoS-атаки необходима довольно высокая пропускная способность канала. Поэтому атака на отказ в обслуживании в большинстве случаев проводится сразу с нескольких машин. Атака, в проведении которой участвует большое количество машин, получила название DDoS. Стоит отметить, что для распределенной атаки могут использоваться инфицированные специальным ПО машины не принадлежащие атакующему. Такие зараженные машины называются "зомби". Одним из способов получения "зомби" является массовое внедрение "трояна" на компьютеры мирных пользователей. Получив определенную извне команду такой "троян" превращает "мирный" компьютер с доступом в Internet в источник ложных запросов, направленных на перегрузку ресурсов сервера.

Наиболее распространенными DoS атаками являются:

* TCP SYN Flood или просто TCP SYN
* TCP flood
* Ping of Death
* ICMP flood
* UDP flood

Рассмотрим подробнее TCP SYN (tcp syn flood) атаку, которая направлена на прикладные сервисы, использующие протокол транспортного уровня TCP. Этот протокол получил широкое распространение в информационных системах за счет того, что он гарантирует 100% доставку всех передаваемых данных. Взаимодействующие узлы сети, использующие в качестве транспорта этот протокол, устанавливают между собой TCP соединения, в рамках которых ведется контроль над тем, что получатель получит все посланные отправителем пакеты. Это достигается за счет того, что получатель извещает отправителя о том, какие пакеты он получил. Если до получателя дошли не все предназначенные ему пакеты, то отправитель повторно их отправит. Как видно достоинством этого протокола является возможность установления соединения. Для хранения информации о текущем состоянии соединения в частности используется поле битовых флагов в пакетах, используемых протоколом. Под это поле отведено 8 бит, однако 2 из них являются зарезервированными и в настоящее время используются только 6 флагов: URG (флаг срочности), ACK (флаг подтверждения), PSH (флаг push функции), RST (флаг сброса), SYN (флаг синхронизации) и FIN (флаг окончания). К сожалению, установленный стандартом [11] механизм установления соединения не является совершенным, и рассматриваемая атака, как раз использует его недостатки.

Основная цель этой TCP SYN атаки – превысить ограничение на количество TCP соединений, которые находятся в состоянии установки. Рассмотрим процедуру установки TCP соединения. Сначала клиент, инициализирующий соединение отправляет серверу TCP-SYN запрос. Получив такой запрос, сервер выделяет память для параметров соединения в специально предназначенном для этого буфере. Затем отправляет клиенту TCP пакет с флагами SYN+ACK. Получив пакет SYN+ACK, клиент должен отправить серверу пакет с подтверждением, т.е. пакет с установленным флагом ACK. Когда сервер получит и обработает этот пакет, соединение является установленным. Описанная выше

процедура изображена на рис. 1.1

Рис. 1.1 Установка TCP соединения

TCP SYN атака производится следующим образом: злоумышленник генерирует большое количество пакетов с установленными SYN флагами протокола TCP. Получая пакеты, атакуемая машина выделяет память для хранения параметров соединения и отправляет ответ – пакет с флагами SYN + ACK и ожидает пакета с флагом ACK. Очевидно, что ожидаемый ответ она не получит, и память будет освобождена только после истечения установленного таймаута. Через некоторое время буфер, выделенный для хранения параметров TCP, соединений будет полностью занят, в результате чего, система не сможет устанавливать новые соединения. После этого каждый дополнительный запрос еще сильнее увеличивает нагрузку. Такие атаки не нуждаются в обратной связи с атакующим, и поэтому злоумышленник может генерировать пакет с произвольными IP адресами отправителя.

Отсутствие обратной связи с атакующим делает обнаружение и отражение TCP-SYN атаки довольно сложной задачей.

1.5 Постановка задач по защите от угроз

В настоящее время в открытой литературе не известны эффективные методы обнаружения TCP SYN атак. В современных ОС присутствуют механизмы защиты атакуемого сервера, например SYN cookies. Операционная система автоматически включает защиту, когда обнаруживает превышение значений некоторых параметров. Например, ОС Windows 2000 следит за значениями трех параметров: TcpMaxHalfOpen, TcpMaxHalfOpenRetried, TcpMaxPortsExhausted [12]. Пороговые значения для этих параметров имеют значения по умолчанию и могут меняться администратором. Если исходные значения не подходят для конкретного сервера, то перед администратором стоит непростая задача эффективно настроить защиту. Кроме того, этот метод требует внесения соответствующих изменений в реализацию стека TCP/IP, которые некоторые специалисты в области сетевых технологий считают "серьезным нарушением" протокола TCP[13].

Другим недостатком средств обнаружения TCP атаки интегрированных в ОС является то, что при перегрузке (имеется в виду нехватка ресурсов процессора и памяти) или зависании самой системы средства противодействия так же становятся неработоспособными.

Целью магистерской работы является создание математически обоснованной методики обнаружения TCP SYN атаки. Для этого необходимо построить математическую модель, описывающую взаимодействие TCP сервера с клиентами. Исходными параметрами для такой модели должны быть характеристики сервера и канала связи, а выходным параметром должно быть утверждение о наличии или отсутствии TCP-SYN атаки.

Для возможности использования предлагаемой методики на практике для защиты критичных ресурсов корпоративной сети необходимы так же средства, позволяющие определить фактические значения входных параметров модели для конкретного сервера и сети, к которой он подключен.

2. ИЗВЕСТНЫЕ МЕТОДЫ ПРОТИВОДЕЙСТВИЯ TCP SYN АТАКЕ

В этом разделе рассматриваются существующие методы обнаружения и противодействия TCP SYN атаке, описываются их достоинства и недостатки.

2.1 TCP SYN Cookies

Этот метод защиты от рассматриваемой атаки был предложен в 1996 году, в скором времени после первых TCP SYN атак [13], вызвавших большой резонанс. В основе этого метода лежит изменение идентификатора последовательности TCP TCP (TCP sequence number). Значение этого параметра определяется следующим образом [13]:

* 5 старших битов: значение t mod 32, где t – 32-разрядный счетчик временных интервалов, значение которого увеличивается на 1 каждые 64 секунды;
* следующие 3 бита: кодированное значение MSS, выбранное сервером в ответ на MSS клиента;
* младшие 24 бита: выбранная сервером на основе IP-адресов и номеров портов отправителя и получателя, а также величины t значение секретной функции.

Такой алгоритм выбора начального порядкового номера соответствует основным требованиям протокола TCP, в соответствии с которыми номера должны увеличиваться достаточно медленно и начальные порядковые номера для серверов растут несколько быстрее, нежели порядковые номера для клиентов.

Серверы, использующие функции SYN cookie, не отвергают соединения при заполнении очереди SYN. Взамен они передают инициатору соединения пакет SYN+ACK, в точности соответствующий пакету, который был бы передан при большем размере очереди SYN (исключения: сервер должен отвергать (reject) опции TCP такие, как большое окно, и должен использовать 1/8 значения MSS, которое он может кодировать). При получении пакета ACK, сервер убеждается в работе секретной функции для последнего (recent) значения t и перестраивает запись очереди SYN в соответствии со значением MSS.

Стоит отметить, что этот метод реализован в ОС семейства Linux и FreeBSD. К достоинствам этого метода можно отнести его достаточную эффективность. Недостатком метода является необходимость модификации реализации стека TCP/IP, которая, по мнению некоторых специалистов в области сетевых технологий [13], противоречит спецификации протокола TCP.

2.2 TCP RST Cookies

Этот метод заключается в том, что клиенту, приславшему запрос на соединение, отправляется SYN+ACK пакет, с неверными параметрами. В соответствии со спецификацией протокола TCP [10] клиент должен прислать RST пакет. Если такой пакет приходит к серверу, то сервер добавляет клиента в список "благополучных" клиентов. Достоинством этого метода является проверка клиента, при этом к недостаткам можно отнести несовершенство механизма такой проверки. Во-первых, эта проверка требует дополнительного времени на отправку некорректного SYN + ACK и получение RST пакетов. В открытой литературе недостаточно информации об этом методе, что затрудняет анализ его достоинств и недостатков. В частности, возникает вопрос, по какому критерию сервер идентифицирует клиентов. Если это только IP адрес, то злоумышленник может при организации атаки устанавливать в качестве адреса отправителя адрес уже проверенного сервером клиента [14].

2.3 Floodgate

Метод заключается в том, что в условиях атаки сервер обрабатывает не все SYN пакеты, а только их часть. Обычно эта часть выбирается случайным образом. Достоинством этого метода является простота реализации, однако это выливается в его же основной недостаток – низкую эффективность, т.к. заявки легальных пользователей будут отфильтровываться системой наравне с вредоносным трафиком [15].

2.4 Предмаршрутизационная фильтрация

Этот метод заключается в том, что каждый маршрутизатор в сети контролирует IP адреса отправителей в маршрутизируемом им трафике, выявляет пакеты с ложными адресами и уничтожает их. Главным достоинством этого метода является то, что он полностью исключает возможность атаки с подложных адресов, что является главным препятствием для ее эффективного обнаружения и блокирования. Недостатком метода является его дороговизна, обусловленная необходимостью замены огромного числа действующих в настоящее время в сети Internet маршрутизаторов [14].

2.5 Random/Old Drop

Метод основан на том, что некоторые из полуоткрытых соединений закрываются сервером. При этом в случае Random модификации закрываются случайно выбранные соединения, а в случае Old – те полуоткрытые соединения, которые существуют дольше остальных. Достоинством такого метода является простота реализации, а главным недостатком – низкая эффективность фильтрации трафика, при которой с высокой вероятностью будут закрыты соединения с легальными клиентами [14].

2.6. SYN-Proxy

Этот метод требует дополнительный прокси-сервер, назначением которого является обработка SYN пакетов. Он служит посредником между клиентом и сервером. Если прокси-серверу удалось установить соединение с клиентом, то клиент допускается к ресурсам главного сервера. Достоинством данного метода является то, что ресурсы основного сервера используются с большей эффективностью, а недостатки заключаются в незащищенности прокси-сервера от атаки и сложность реализации [14].

2.7 Stack Tweaking

Заключается в изменениях настройки параметров протокола TCP на сервере. Как правило, этими параметрами являются: таймаут перед закрытием полуоткрытого соединения, максимально возможное количество полуоткрытых соединений и время ожидания ответного ACK-пакета. Достоинством метода является возможность повышения эффективности работы сервера за счет учета параметров сервера и сети. Недостатки заключаются в том, что это метод не работает против интенсивной атаки и требует высокой квалификации администратора [12].

2.8 Blacklisting

Заключается в том, что сервер не обслуживает заявки, поступающие от клиентов, внесенных в "черный список" [15]. Этот метод малоэффективен в виду того, что обычно атаки производятся с подставных адресов. Этот метод был бы эффективен при использовании совместно с методом предмаршрутизационной фильтрации, рассмотренном в пункте 2.4.

Таким образом, в настоящее время существуют различные механизмы обнаружения и противодействия TCP SYN атаке. Наиболее эффективным из них является метод SYN Cookies, однако он, как и все другие, имеет недостатки, такие как необходимость внесения соответствующих изменений в реализацию стека протоколов TCP/IP на защищаемом сервере, недостаточная эффективность обнаружения атаки из-за отсутствия методики выбора конкретных значений для параметров защиты. В связи с этим, проблема TCP SYN атак требует новых эффективных решений, и тематика данной работы является чрезвычайно актуальной.

3. МАТЕМАТИЧЕСКАЯ МОДЕЛЬ TCP-SYN АТАКИ

Целью магистерской работы является создание методики обнаружения TCP SYN атаки, в основе которой находится математическая модель, учитывающая особенности среды, для которой разрабатывается методика. В связи с этим на начальном этапе исследований стал вопрос о том, какой математический аппарат целесообразно использовать для построения наиболее эффективной модели. После анализа существующих направлений современной науки было принято решение использовать теорию систем массового обслуживания, специфика которой идеально подходит для решения поставленной задачи. Ниже приведено описание некоторых понятий, используемых в работе. Подробнее с этим разделом теории вероятностей можно ознакомиться здесь [16].

3.1 Краткие сведения из теории систем массового обслуживания

Под системой массового обслуживания [16] обычно понимается совокупность обслуживающих приборов и обслуживаемых требований (заявок) из некоторого входящего потока требований.

Число приборов в системе массового обслуживания может быть любым. Основной характеристикой прибора является время обслуживания одного требования этим прибором. Этот показатель характеризует не качество обслуживания, а пропускную способность прибора. Время обслуживания обычно непостоянно. Оно зависит от различных факторов. Поэтому в общем случае эта величина является случайной [16]. При этом в теории массового обслуживания считают, что продолжительность обслуживания различных требований одним прибором есть независимые случайные величины с одним и тем же законом распределения. Наиболее часто предполагают, что это закон является показательным. Его применяют в тех случаях, когда время обслуживания подавляющего большинства требований мало и только для сравнительно небольшой части требований оно велико. При показательном распределении времени обслуживания требований теоретические рассуждения существенно упрощаются, а многие окончательные результаты оказываются справедливыми и для произвольного закона распределения, но с тем же средним временем обслуживания [16].

Так же в теории массового обслуживания принято считать, что входящий поток требований распределен по пуассоновскому закону распределения. По определению пуассоновский поток должен удовлетворять трем следующим требованиям: стационарности, отсутствия последствия и ординарности.

Поток называется стационарным, если вероятность поступления k требований в течение промежутка времени t не зависит от момента начала этого промежутка.

Под отсутствием последствия понимается то, что вероятность поступлений k требований в систему после произвольного момента времени t0 не зависит от того, когда и сколько поступило требований до этого момента времени. Из этого следует взаимная независимость поступления того или иного числа требований на обслуживание в непересекающиеся промежутки времени.

Свойство ординарности означает, что вероятность поступления более одного требования за малый промежуток времени dT есть величина более высокого порядка малости, чем dT. Оно выражает практическую невозможность одновременного поступления двух или более требований.

Стоит отметить, что многие реальные потоки являются приближенно пуассоновскими.

Пуассоновский поток полностью определяется одним параметром – интенсивностью потока λ.. На практике величину λ находят статистически. При этом одновременно, например, при помощи критерия согласия , проверяют, действительно ли рассматриваемый поток требований с заданной вероятностью можно считать пуассоновским.

Математический аппарат теории массового обслуживания позволяет определить основные параметры системы: среднее число занятых приборов, вероятность отказа в обслуживании требования, среднюю длину очереди, среднее время простоя требования в очереди и т.д.

Для нас наибольший интерес будет представлять среднее число занятых приборов [16]:

(3.1)

Где n – количество приборов в системе,

,

 - интенсивность потока требований,

 - математическое ожидание времени обслуживания одного требования.

 - вероятность нахождения в системе ровно k требований

(3.2)

Приведенные соотношения позволяют определить среднее количество заявок, находящихся в системе массового обслуживания, что будет использовано ниже при построении методики обнаружения рассматриваемой атаки.

3.2 Поток требований СМО

Будем рассматривать множество TCP SYN пакетов, поступающих к серверу, в качестве входящего потока заявок. Покажем, что в определённых условиях этот поток можно считать пуассоновским.

Интенсивность этого потока может зависеть от времени, если рассматривать его в течение достаточно больших промежутков времени. Например, в течение суток в дневное время его интенсивность может быть больше, чем ночью. Тем не менее, при уменьшении продолжительности рассматриваемого промежутка интенсивность поступающих TCP SYN заявок стабилизируется и может рассматриваться как некоторая постоянная величина. Для различных сетей продолжительность такого промежутка может быть разной (как правило, от нескольких минут до нескольких часов) и может быть установлена экспериментально.

В этом случае вероятность поступления k требований в интервале времени (0, t) равна вероятности поступления k требований в любом другом интервале той же длительности (a, a + t) в пределах заданного промежутка. Таким образом, рассматриваемый поток обладает свойством стационарности.

Далее будем считать, что пользователи обращаются к ресурсам сервера независимо друг от друга. Если при одном обращении пользователя к серверу устанавливается одно TCP соединение, то поток требований обладает свойством отсутствия последствия (в соответствии с определением этого свойства [16]). Однако некоторые приложения прикладного уровня взаимодействуют друг с другом посредством параллельно установленных TCP соединений. Покажем, что и в этом случае входящий поток обладает указанным свойством.

Рассмотрим влияние процесса обращения браузера к web-странице на поток TCP SYN пакетов, поступающих к серверу. Как правило, большинство возвращаемых сервером страниц содержат гиперссылки на другие ресурсы, такие как изображения, элементы ActiveX, flash-анимации и другие элементы, выводимые на html странице в окне браузера. В соответствии со спецификацией протокола HTTP [17], для получения каждого из ресурсов браузер должен сделать отдельный запрос к web серверу, и, следовательно, установить TCP-соединение. Если web-страница содержит i элементов, требующих немедленной загрузки, то при выполнении N обращений количество TCP соединений будет равно (i+1)N. В этом случае можно рассматривать в качестве одного требования отправление (i+1) SYN пакетов. Очевидно, что каждое обращение к web-странице можно рассматривать как одну заявку, и интенсивность потока таких требований будет в (i+1) раз меньше. В предлагаемой далее модели возможно введение дополнительного коэффициента для учёта объёдинения таких взаимосвязанных SYN пакетов в одну заявку. Объединенные заявки являются независимыми, т.к. пользователи обращаются к ресурсам сервера независимо друг от друга. Из этого следует, что входящий поток требований обладает свойством отсутствия последствия.

Покажем, что поток требований является ординарным. Рассмотрим сервер с одним сетевым интерфейсом. По такому подключению одновременно не могут прийти сразу несколько IP пакетов, т.к. в блоке данных протоколов канального уровня (Ethernet, DSL-соединение, модемное подключение и др.) может быть максимум один IP пакет [18]. Соответственно, существует некоторый малый промежуток времени, в течение которого может поступить не более одной заявки. Следовательно, для сервера с одним сетевым интерфейсом входной поток TCP SYN пакетов является ординарным.

Таким образом, поток заявок, содержащих TCP SYN пакеты, поступающие на сервер с одним сетевым интерфейсом, обладает свойствами стационарности, ординарности и отсутствия последствия, и в соответствии с определением, такой поток является пуассоновским.

3.3 Сервер TCP соединения как СМО

Как было показано в п.3.2, поток поступающих на сервер TCP SYN пакетов в заданных условиях является пуассоновским. Это значит, что его можно рассматривать как поток требований, поступающих в СМО. Однако для построения модели удобнее в качестве множества заявок рассматривать эквивалентный ему поток. В нормальном режиме работы в ответ на каждый полученный TCP SYN пакет сервер должен отправить TCP SYN+ACK пакет[10]. Из того, что существует взаимнооднозначное соответствие между входящими и исходящими пакетами следует эквивалентность потоков. Далее в качестве требований СМО будем рассматривать отправляемые сервером SYN+ACK пакеты. Множеством обслуживающих приборов будем считать ресурсы сервера, предназначенные для хранения параметров TCP соединений. В такой интерпретации обслуживание требования – это резервирование соответствующих ресурсов либо до успешного установления TCP соединения (получения ACK пакета, который должен быть получен в соответствии с [10]), либо до истечения отведенного на сервере таймаута.

Для такой модели признаком TCP SYN атаки является резкое увеличение количества заявок в СМО. Находясь под воздействием атаки, сервер выделяет соответствующие ресурсы, которые остаются занятыми в течение отведенного таймаута. Для современных операционных систем и сетевых технологий времени таймаута (от десятков секунд до нескольких минут [12]) достаточно чтобы занять все доступные ресурсы сервера, предназначенные для хранения параметров TCP соединений. Для рассматриваемой нами модели это означает резкое увеличение занятых обслуживающих приборов.

Рассмотрим более детально ресурсы сервера, выступающие в качестве обслуживающих приборов. Параметры TCP соединений хранятся в соответствующем буфере [18], который можно представить в виде массива размерности L, элементы которого хранят параметры TCP соединений. Их можно разделить на три типа: содержащие параметры установленных соединений, полуоткрытых соединений и свободные. Пусть B – количество открытых в данный момент TCP соединений. Тогда n = L - B – количество элементов второго и третьего типов, совокупность которых мы будем рассматривать в качестве множества обслуживающих приборов СМО. При этом занятые обслуживанием требований приборы – это элементы второго типа. На рис. 3.1 изображен описанный массив, а на рис. 3.2 показано представление ресурсов сервера в качестве множества обслуживающих приборов.

Рис. 3.2 Сервер TCP соединения как СМО

В зависимости от соотношения интенсивности входящего потока требований  и размерности массива L можно рассматривать два типа СМО. Если интенсивность входящего потока заявок значительно меньше возможностей сервера, что справедливо для большинства современных систем, то целесообразно рассматривать СМО с бесконечным числом обслуживающих приборов. В противном случае можно рассматривать СМО с отказами. Ввиду того, что на практике в нормальном режиме работы возможности сервера со значительным запасом покрывают входящие требования, то рассмотрение системы с отказами является неактуальным. В дальнейшем будем рассматривать систему первого типа.

3.4 СМО с бесконечным количеством обслуживающих приборов

## Как уже было показано, для описания модели взаимодействия клиентов и сервера TCP соединения в нормальном режиме работы целесообразно рассматривать СМО с бесконечным числом обслуживающих приборов. Обозначим отношение интенсивности входящего потока требований  к среднему времени обслуживания заявки  коэффициентом . Т.к. поток требований является пуассоновским, то вероятность того, что в системе находится ровно k требований, определяется как

(3.3)

Подставив это значение в соотношение (3.2), описывающее среднее число приборов, занятых обслуживанием (общее число полуоткрытых соединений) получим:

(3.4)

Соответственно,

(3.5)

Из соотношений (3.4) и (3.5) для СМО с бесконечным числом обслуживающих приборов имеем [16]:

(3.6)

Предложенная модель описывает работу сервера в нормальном режиме и позволяет учитывать такие параметры, как интенсивность обращений к серверу и среднее время обслуживания заявки. Однако такая СМО недостаточно полно описывает работу сервера, т.к. не учитывает возможность потери пакетов при передаче в современных сетях.

Для усовершенствования предложенной модели целесообразно разделить рассматриваемую СМО на две системы, обслуживающие заявки на нормальное установление соединения (когда все пакеты доставлены) и полуоткрытые соединения, удаляемые по таймауту. Для разделения исходного потока требований на множества заявок для каждой из систем необходимо ввести критерий, позволяющий определить принадлежность заявок к вышеописанным типам. Для этого в дальнейшем будет использован тот факт, что в большинстве случаев время прохождения IP пакета между произвольными хостами в Internet не превосходит некоторого порогового значения [19]. Определение этого порога рассмотрено в п.4.1.

### 3.5 Модель, учитывающая потерю пакетов в сети

Как было отмечено выше, предложенная ранее модель требует усреднения среднего времени обслуживания по всем требованиям, что не в полной мере учитывает особенности процесса установления TCP соединений. Для устранения этого недостатка разделим описанную в п.3.3 СМО на две системы: СМО1 и СМО2. Будем считать, что первая система описывает обслуживание заявок, для которых полуоткрытые соединения будут успешно установлены после получения сервером ACK пакетов, а вторая – требования, для которых соединения не будут установлены и после истечения отведенного таймаута будут удалены.

Как будет показано в п.4.1, в большинстве случаев время обмена парой пакетов между произвольными хостами не превосходит порог . При условии того, что на клиенте корректно реализован протокол TCP, появление полуоткрытых соединений, которые не установленных в течение промежутка времени длительностью  объясняется потерей либо SYN+ACK, либо ACK пакета. Поэтому к требованиям второго типа будем относить заявки, для которых TCP соединение находится в полуоткрытом состоянии дольше чем . Обозначим через s и l – количества соединений первого и второго типов соответственно. Такое представление сервера изображено на рис.3.4

Рис 3.4. Сервер TCP соединения, как СМО

Определим соотношения, описывающие состояние такой системы. Аналогично (3.6) определим среднее количество полуоткрытых соединений:

 . (3.7)

Как следует из соотношения (3.7), среднее число полуоткрытых соединений является случайной величиной, равной сумме двух случайных величин, имеющих пуассоновский закон распределения. Первая из них описывает среднее количество полуоткрытых соединений, которые не представляют собой угрозу с точки зрения TCP SYN атаки. Вторая составляющая представляет собой полуоткрытые соединения, которые не будут установлены и через заданный промежуток времени (определяемый таймаутом) будут удалены, до этого занимая ресурсы сервера. Как отмечалось выше, увеличение количества таких соединений является признаком TCP SYN атаки. Поэтому в основу методики целесообразно положить СМО, учитывающую только требования второго типа.

Далее будем рассматривать в качестве заявки не все SYN+ACK пакеты, для которых сервер ожидает ответный ACK пакет, а только те, для которых время ожидания превышает пороговое значение , описанное в п.3.5. Очевидно, что при нормальной работе (отсутствии TCP SYN атаки) для каждой такой заявки был потерян либо SYN+ACK, либо ACK пакет. Интенсивность поступления таких заявок определяется следующим соотношением:

(3.8)

где:

 – интенсивность поступающих на вход сетевой карты сервера TCP SYN пакетов

 – вероятность появления полуоткрытого соединения, которое не будет установлено

Параметр  зависит от качества работы сети, которое характеризуется вероятностью потери пакета в сети (). Методика определения фактического значения этой вероятности описана в [20]. Найдем зависимость  от . Пусть событие A заключается в том, что был потерян SYN+ACK пакет, а событие B представляет собой потерю ACK пакета. Вероятность события A равна вероятности потери пакета в сети:

(3.9)

Т.к. событие B может наступить только тогда, когда не наступило событие A (ACK пакет может быть отправлен только после получения SYN+ACK пакета[10]), то его вероятность равна:

(3.10)

Рассмотрим событие C, заключающееся в появление полуоткрытого соединения второго типа. Оно равно сумме событий A и B. Отсюда с учетом (3.9) и (3.10) получим:

 (3.11)

Из соотношений (3.8) и (3.11) найдем интенсивность потока требований второго типа:

(3.12)

В современных ОС таких как Windows и Linux, ядро отсылает несколько копий SYN+ACK пакетов до тех пор, пока не будет получен ACK

пакет. Обозначим количество таких копий параметром . Тогда интересующее нас событие заключается в том, что ни для одной из копий SYN + ACK пакета не дойдет ответный ACK пакет, и соотношение (3.12) принимает следующий вид:

(3.13)

Т.к. интенсивность потока требований второго типа пропорциональна интенсивности первоначального потока, то он так же является пуассоновским.

Среднее число таких заявок, находящихся на обслуживании в СМО, определяется вторым слагаемым формулы(3.7):

(3.14)

где: – таймаут отведенный на сервере на установление TCP соединения

 – вероятность потери пакета в сети

 – количество копий SYN + ACK пакетов отправляемых ОС

Как было показано в п.3.4, СВ характеризующая среднее количество занятых приборов в СМО рассматриваемого нами типа пуассоновский закон распределения. В нашем случае параметр этого распределения равен l. Известно, что для пуассоновского распределения математическое ожидание и дисперсия равны параметру распределения [21] и в нашем случае так же равны l.

На рисунках 3.4 и 3.5 приведены графики плотности распределения и закона распределения случайной величины, распределенной по пуассоновскому закону, с параметром .

Для определения наличия или отсутствия TCP SYN атаки нам понадобится значение функции распределения, которое, как известно, определяется следующим образом:

(3.15)

При использовании данной модели, признаком TCP SYN атаки является превышение значения функции распределения от текущего количества полуоткрытых соединений некоторого порогового значения Fпор (рис. 3.5), которое будет соответствовать вероятности верного обнаружения TCP SYN атаки (вероятности ошибки первого рода).

Достоинствами предлагаемой методики являются возможность своевременного (раннего) обнаружения атаки, ее способность адаптироваться к реальным параметрам сети. При значительном увеличении интенсивности обращений к серверу со стороны легальных пользователей количество потерянных пакетов увеличится пропорционально вероятности потери пакета в сети. Т.к. для современных сетей эта величина имеет небольшое значение, то эффективность обнаружения снизится незначительно. Недостатком является то, что неисправности сетевого оборудования, в результате которых увеличивается вероятность потери пакета в сети, будут интерпретированы как TCP SYN атака.

Для того, чтобы иметь возможность эффективно обнаруживать атаку на практике, необходимы средства, позволяющие определять значения исходных параметров модели для конкретного сервера и сети, к которой он подключен. Возможные методики сбора таких данных предлагаются ниже.

4. МЕТОДИКИ СБОРА ДАННЫХ

Для эффективного использования предложенной выше методики на практике необходимо иметь возможность определить фактические значения исходных параметров модели для системы находящейся в нормальном состоянии (при условии отсутствия атаки). Как было показано выше, такими параметрами являются: интенсивность потока заявок (TCP пакетов с установленным SYN флагом), вероятность потери пакетов в сети, к которой подключен сервер и среднее время обслуживания заявки (успешного установления TCP соединения). В этом разделе приведены возможные подходы к решению этой задачи. Более подробно этот материал изложен в [22].

4.1 Определение времени прохождения IP пакета по сети Internet

Значение порогового значения  времени прохождения пакетов между двумя хостами в сети работающей в нормальном режиме можно определить двумя способами. Первый способ – это, накопив довольно большую статистику, найти максимальное значение. Более сложный подход основан на проверке гипотезы о законе распределения СВ, описывающей время прохождения пакетов.

В обоих случаях нам необходима большая статистика времени прохождения пакетов, наиболее полно представляющая генеральную совокупность. Для ее получения была использована утилита ping.

Известно, что эта утилита предназначена для проверки качества соединения с удаленным компьютером. Утилита ping работает поверх протокола ICMP (Internet Control Message Protocol). Для проверки соединения с удаленным хостом утилита ping посылает ему ICMP-request, в ответ на который, удаленный хост должен ответить сообщением ICMP-reply.

Утилита ping позволяет получить статистику для заданного хоста. Среди прочих данных этой статистики можно определить время прохождения каждой пары пакетов с сообщениями ICMP-request и ICMP-reply между двумя хостами. Именно этот параметр нас и интересует – время за которое два удаленных хоста обмениваются парой пакетов (в нашем случае это TCP SYN+ACK и TCP ACK пакеты).

Для накопления ping статистики были использованы различные удаленные хосты, физически расположенные в разных странах и на разных континентах. Исходя из этого, можно утверждать, что полученные результаты достаточно обобщены.

Полученные результаты приведены на рисунках 4.1 – 4.3.

Рис. 4.1. Время возврата пакетов ICMP-reply

На рис. 4.1 и 4.2 приведена упорядоченная по возрастанию статистика времени возврата ICMP-reply. На рис. 4.2 статистика приведена без учета трех пакетов, время возврата которых в 40 раз превышает время возврата большинства пакетов. Рис. 4.2 приведен здесь для представления результатов в более крупном масштабе.

Рис. 4.2. Время возврата пакетов ICMP-reply

Рис. 4.3 Эмпирическое распределение времени возврата ICMP ответа.

На рис. 4.3 изображено эмпирическое распределение времени возврата ICMP ответов. По оси абсцисс отложены временные интервалы, полученные разбиением всего диапазона значений с шагом 10 мс. По оси ординат отложено количество значений, попавших в интервал.

Глядя на рис. 4.3, можно выдвинуть гипотезу о том, что распределение интересующей нас СВ представляет композицию нескольких распределений с различными параметрами. Определение этих законов и их параметров является перспективой развития полученной методики.

На данном этапе для определения порогового значения воспользуемся более простым методом. Проанализировав полученные эмпирические данные можно убедиться в том, что более чем в 98% случаев время прохождения пары пакетов от одного хоста к другому и обратно через сеть Internet не превышает 200 мс.

4.2 Определение вероятности потери пакетов

Для определения вероятности  потери пакетов в сети, так же можно воспользоваться утилитой ping, описанной в предыдущем разделе. Ее значение равно отношению количества пакетов с истекшим таймаутом ожидания к общему количеству отправленных ICMP запросов.

При анализе экспериментальных данных, полученных в предыдущем разделе было получено следующее значение:



Список пингуемых хостов был получен путем обработки html страницы, сгенерированной proxy сервером. Эта страница представляет собой месячный отчет о доступе автора в internet. В связи с этим ответы на ICMP запросы отправленные на некоторые хосты получены не были. Это объясняется тем, что эти хосты могут находится за межсетевым экраном, фильтрующим пакеты протокола ICMP. Поэтому при определении количества потерянных пакетов учитывались только те хосты, от которых приходил хоть один ICMP ответ.

4.3 Определение интенсивности входящего потока требований

В качестве предложений для определения этого параметра можно выделить:

* Анализ логов какого либо сниффера. Например Ethereal, tcpdump, IDS Snort, работающей в режиме сниффера.
* Написание собственного ПО

Соответствующее ПО разработано моим коллегой [22].

В этом разделе приведены возможные подходы к определению фактических значений исходных параметров рассмотренной выше модели для защищаемой системы находящейся в нормальном состоянии (при условии отсутствия атаки). Как отмечалось выше, такими параметрами являются интенсивность потока заявок (TCP пакетов с установленным SYN флагом), вероятность потери пакетов в сети, к которой подключен сервер и среднее время обслуживания заявки (успешного установления TCP соединения). Имея фактические значения этих параметров с помощью соотношений приведенных в разделе 3 можно определить допустимое пороговое значение для количества "просроченных" полуоткрытых соединений, которое будет использовано ниже.

Одним из важных этапов исследований является программная реализация предлагаемой методики противодействия TCP SYN атаке, особенности которой рассмотрены ниже.

5. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

Было принято решение реализовать предложенную методику обнаружения TCP SYN атаки в качестве модуля расширения функциональности (plug-in) для системы обнаружения вторжений IDS Snort. В качестве достоинств Snort, которые определили этот выбор можно отметить легко расширяемую архитектуру, открытость исходного кода. Так же стоит упомянуть, что эта система способна работать на большом количестве аппаратных платформ и ОС.

Т.к. в реальности простого обнаружения атаки недостаточно для бесперебойной работы системы, то помимо обнаружения система должна обеспечивать отражение атаки. Поэтому модуль расширения функциональности разрабатывался для модификации системы Snort – IPS Snort\_inline. Это система предотвращения вторжений, которая способна модифицировать пакеты в реальном режиме времени, по мере того как они поступают в сеть и/или покидают ее. Такие системы так же называются системами "с активным ответом". Для того, чтобы IPS могла контролировать весь трафик, поступающий и исходящий из сети, она должна располагаться на inline устройстве. Поэтому в предложенной реализации Snort\_inline работает на Linux-системе, функционирующей в режиме маршрутизатора.

5.1 Особенности установки Snort

Как было сказано выше, целевая система для разрабатываемого модуля – это Snort\_inline. Однако ее установка и настройка занятие, отнимающее много сил и времени. Поэтому на начальных этапах разработка велась на системе Snort, в то время как мой коллега занимался исследованием возможностей snort\_inline [22].

Для того чтобы поставить Snort на Linux машину необходимо, предварительно установить некоторые библиотеки:

* libpcap – библиотека работающая на канальном уровне. Используется системой Snort. С помощью нее snort получает доступ ко всем пакетам поступающим на сетевой интерфейс.
* libpcre – библиотека для работы с регулярными выражениями (regular expressions)
* libipq – используется системой Snort\_inline вместо библиотеки libpcap. Это библиотека организации очередей пакетов, которую предоставляет IPtables

Для непосредственной установки системы необходимо ввести стандартные команды:

/configure

make

make install

Стоит отметить, что на этапе разработки следует вызвать скрипт ./configure с опцией –enable-debug.

Настройка системы Snort (как и большинства других) дело тонкое, занимающее много времени. Поэтому на начальных этапах разработки было принято решение отлаживать модуль расширения функциональности на системе работающей со значениями практически всех параметров установленными по умолчанию. Единственным переопределяемым параметром был путь к файлу с правилами. Для того чтобы, запустить snort в таком режиме, надо выполнить следующую команду:

/snort –c rules.txt

Это значит, что snort будет проверять все пакеты на соответствие правилам, указанным в файле rules.txt.

5.2 Внутренняя структура Snort

Система Snort имеет гибкую архитектуру, представленную в виде множества подключаемых модулей. Подключаемые модули бывают трех типов:

* препроцессоры
* модули обнаружения
* модули вывода

5.2.1 Препроцессоры

Препроцессоры Snort бывают двух типов. Первый тип предназначен для обнаружения подозрительной активности, а второй тип предназначен для модификации пакетов протоколов высоких уровней (чем канальный) для последующей их обработки процессором обнаружения. Этот процесс называется нормализацией трафика. Он позволяет обнаруживать атаки, которые манипулируют внешним видом трафика для большей скрытности. Существует много препроцессоров snort, которые можно подключить или отключить, внеся соответствующие изменения в файл конфигурации. Исходные коды препроцессоров находятся в директории ./src/preprocessors. Здесь приведены только некоторые из них:

* portscan (2) – предназначен для обнаружения сканирования портов
* http\_inspect – предназначен для контроля http трафика
* stream4 – предназначен для контроля за TCP сессиями
* arpspoof – предназначен для обнаружения атак arp-spoofing
* bo – предназначен для обнаружения активности BackOrifice
* frag2 – предназначен для сборки фрагментированных пакетов
* RPC-decode - декодирование RPC-трафика;
* Telnet\_decode - декодирование трафика telnet-сессий;
* ASN1\_decode - выявление аномалий в строках формата ASN1;
* и т.д.

Открытость архитектуры snort дает возможность разработчикам написать свои препроцессоры, ориентированные на решения специфических задач.

5.2.2 Модули обнаружения

Модули обнаружения используются непосредственно для анализа обработанного препроцессорами трафика. Если этот модуль определяет, что пакет удовлетворяет указанному правилу, то он генерирует событие, которое дальше передается модулям вывода Snort. Именно в виде модуля обнаружения реализована методика обнаружения TCP SYN атаки, поэтому его структура будет подробно описана ниже.

5.2.3 Модули вывода

Модули вывода используются Snort для записи событий безопасности, ведения логов и т.д. в различные устройства и хранилища данных. Возможно настроить систему на ведение логов в отдельную базу данных, двоичные и текстовые файлы различных форматов. Возможны даже такие экзотические варианты, как уведомления администратора по e-mail или SMS. Исходные коды этих модулей находятся в директории ./src/output-plugins Вот некоторые модули вывода:

* alert\_syslog – вывод в формате syslog
* log\_tcpdump – вывод в формате tcpdump
* output\_database – вывод в БД. Возможны mysql, postgresql, oracle, odbc, mssql(только для Snort под Windows)
* csv – вывод в формате CSV ( coma separated values)
* и т.д.

У разработчиков так же есть возможность написания своих модулей вывода, которые благодаря открытости архитектуры легко интегрировать.

5.3 Разработка модуля обнаружения

В корневой директории Snort есть директория templates, в которой находятся шаблоны модулей вывода и обнаружения. Рассмотрим шаблон модуля обнаружения, представленный двумя файлами:

* sp\_template.h – заголовочный файл модуля
* sp\_template.c – файл реализации модуля

В заголовочном файле должно присутствовать объявление setup функции, которая отвечает за инициализацию модуля. Для того чтобы Snort знал, что необходимо проинициализировать данный модуль необходимо добавить вызов этой функции в функцию InitPlugIns(), реализация которой находится в файле ./src/plugbase.c. При этом, конечно, надо не забыть указать компилятору с помощью директивы #include в каком именно файле она определена.

Применительно к нашему модулю, который называется tcp\_syn\_flood необходимо сделать следующее:

1. создать в директории ./src/detection-plugins/ два файла:
* sp\_tcp\_syn\_flood.h
* sp\_tcp\_syn\_flood.с
1. в файл sp\_tcp\_syn\_flood.h добавить объявление функции SetupTcpSynFlood(), а в файл sp\_tcp\_syn\_flood.с ее реализацию:

void SetupTcpSynFlood()

{

/\* регистрируем модуль \*/

RegisterPlugin("tcp\_syn\_flood", TcpSynFloodInit);

DEBUG\_WRAP(DebugMessage(DEBUG\_PLUGIN,"Plugin: TcpSynFlood Setup\n"););

}

Здесь вызывается функция RegisterPlugin, предоставляемая snort, которая говорит snort о том, что если в правиле встречаются опции, относящиеся к модулю tcp\_syn\_flood, то необходимо вызвать функцию TcpSynFloodInit, находящуюся в файле sp\_tcp\_syn\_floodы.с. Реализация функции инициализации будет описана ниже.

1. добавить вызов функции SetupTcpSynFlood() в plugbase.c:
* В секции /\* custom detection plugin \*/ добавить

#include "detection-plugins/sp\_tcp\_syn\_flood.h"

* В теле функции InitPlugIns() добавить вызов функции SetupTcpSynFlood();
1. добавить элемент PLUGIN\_TCP\_SYN\_FLOOD к перечислению (enum) доступных модулей Snort

Для того чтобы наш модуль был успешно интегрирован в snort осталось реализовать функцию TcpSynFloodInit(). Это статическая функция, в которой необходимо выделить память для используемых модулем структур данных, проинициализировать эти структуры данными, указанными в правиле. Здесь стоит отметить, что для каждого правила, в котором заданы параметры для модуля tcp\_syn\_flood будет вызвана эта функция.

Здесь так же стоит отметить, что модуль tcp\_syn\_flood должен генерировать события Snort связанные с началом и окончанием атаки. Для этого его надо зарегистрировать в качестве генератора событий Snort. Сделать это проще простого. Надо добавить соответствующие объявления в файл ./src/generatos.h:

#define GENERATOR\_TCP\_SYN\_FLOOD 224

#define SYNFLOOD\_STARTED "(tcp syn flood: PREVED)"

#define SYNFLOOD\_FINISHED "(tcp syn flood: PAKA)"

В первой строчке мы регистрируем модуль в качестве генератора событий, а дальше идет объявление сообщений, которые могут записываться в логи. Как уже было сказано, нам необходимо генерировать два события.

Перед тем как продолжить описание реализации модуля на языке C, следует описать внутреннее устройство модуля и используемых в нем структур данных. Модуль tcp\_syn\_flood предназначен для обнаружения и возможного отражения TCP SYN атаки. Обе эти функциональности представлены отдельными подмодулями: TcpConnEstTimeChecker и TcpSynFloodPreventionModule. Первый из них предназначен именно для обнаружения атаки с помощью описанной разделе 3 методики. Он является решающей системой, которая генерирует события snort говорящие о том, что TCP атака началась или закончилась. Второй подмодуль предназначен для накопления "положительной" статистики работы защищаемого сервера при условии атаки. Под положительно статистикой здесь понимается учет количества успешно установленных TCP соединений сервера с клиентами. Дифференциация клиентов производится по признаку их IP адресов и значения поля TTL в приходящих от них пакетах. Если атака начинается, то накопление статистики приостанавливается, а на основании уже накопленных данных модуль принимает решения от каких клиентов пропускать SYN пакеты к серверу, а от каких нет.

5.3.1 Структура модуля TcpConnEstTimeChecker

Полное название этого модуля Tcp Connection Estimate Time Checker. Он предназначен для определения количества "просроченных" tcp соединений и сравнения их количества с допустимым порогом.

Стоит напомнить, что в соответствии с изложенной в разделе 3 методикой необходимо вести учет количества полуоткрытых TCP соединений, которые находятся в этом состоянии дольше определенного промежутка времени. Значения указанных параметров этот модуль берет из правила Snort:

* server\_timeout\_sec – таймаут отведенный на сервере для установки TCP соединений. Значение этого параметра задается в секундах
* max\_overdue\_count – максимально допустимое количество полуоткрытых на сервере соединений
* max\_overdue\_count\_diviation – разброс максимально допустимого количества полуоткрытых на сервере соединений. Это значит, что модуль будет генерировать событие "TCP SYN атака началась" после того, как на сервере будет (max\_overdue\_count + max\_overdue\_count\_diviation) полуоткрытых соединений и, соответственно "TCP SYN атака закончилась", когда количество таких соединений станет меньше чем (max\_overdue\_count -max\_overdue\_count\_diviation)
* overdue\_time\_sec – время после которого полуоткрытое соединение считается "просроченным". Значение этого параметра задается в секундах
* check\_period\_sec – период с которым модуль должен проверять текущее количество полуоткрытых соединений. Как будет показано дальше, операция проверки количества просроченных соединений требует больше вычислительных ресурсов, чем просто обработка пакетов. Если этому параметру установить довольно большое значение, то атака будет обнаружена позже, а если маленькое значение, то больше ресурсов системы будет расходоваться на более частые проверки.

Для того чтобы минимизировать затраты памяти и увеличить быстродействие было принято решение не хранить время прихода пакетов в явном виде. Для определения "возраста" полуоткрытых соединений используется довольно хитрая структура данных: массив бинарных деревьев. Такой массив показан на рис.5.1.

Рисунок 5.1 Массив бинарных деревьев, используемый TcpConnEstTimeChecker.

В узлах каждого из деревьев хранится информация об отдельном полуоткрытом TCP соединении. Эта информация представлена следующей структурой:

typedef struct \_TimeCheckerTreeNodeData

{

ubi\_trNode Node;

// Номер acknowledgement последовательности (для Syn+Ack и Ack пакетов)

u\_int32\_t SeqAckNumber;

}

TChTreeNodeData;

Для более тесной интеграции со Snort была использована готовая реализация бинарных деревьев, поставляемая с системой. Эта реализация так же используется другими модулями. Это позволило значительно сократить время разработки модуля, повысить эффективность его реализации и уменьшить количество возможных багов. Более того, в случае если в этой реализации деревьев имеются баги, то использующий их модуль автоматически подхватит исправления, если будет установлен на более поздние версии Snort. В Snort есть несколько реализаций бинарных деревьев: ubi\_BeenTree и ubi\_SplayTree. Они приведены к единому интерфейсу, который позволяет работать с ними независимо от текущей реализации. То какие деревья используются указывается лишь включением соответствующего заголовочного файла. В данный момент использованы Splay деревья. В дальнейшем возможно сравнение производительности системы, основанной на другой реализации.

Как видно из приведенного фрагмента кода в структуре объявлены два поля:

* Node – узел бинарного дерева. Это поле должно идти первым в объявлении структуры. Это обусловлено оптимизацией во внутренней реализацией бинарных деревьев Snort
* SeqAckNumber – значение последовательности Acknowledgement в SYN+ACK пакетах исходящих от защищаемого сервера.

При получении данным модулем SYN+ACK пакета исходящего от защищаемого сервера в дерево, имеющее нулевое смещение в массиве помещается соответствующий элемент. Этот элемент соответствует полуоткрытому на сервере соединению.

Как уже отмечалось выше, в целях минимизации используемой памяти и увеличения быстродействия, для каждого узла не хранится время его создания. "Возраст" узла определяется индексом дерева в массиве деревьев, в котором он находится. С периодичностью задаваемой вышеописанным параметром правила check\_period\_sec массив деревьев сдвигается на 1 дерево вправо. При этом последнее дерево удаляется, освобождая выделенную под него и его элементы память, а элемент массива с нулевым смещением инициализируется новым деревом.

Размерность массива деревьев определяется при инициализации модуля в функции InitTcpConnEstTimeChecker как:

int rootNodesCount = ceil(serverTimeout / \_checkPeriod);

При такой организации внутренних структур данных "возраст" полуоткрытых соединений, которым соответствуют узлы в i-м дереве массива определяется как (\_checkPeriod \* i) . Узлы самого правого в массиве дерева, которые при сдвиге удаляются соответствуют полуоткрытым на защищаемом сервере соединениям для которых истек таймаут отведенный на установку соединений и они закрываются автоматически.

При получении модулем ACK пакета, предназначенного серверу, производится поиск узла дерева, соответствующего полуоткрытому соединению. Поиск производится по ключу (Acknowledge number ACK пакета – 1). После того как узел найден, он удаляется из дерева, что соответствует закрытию полуоткрытого соединения на сервере.

Как было показано в разделе 3, основным критерием по которому принимается решение о начале или окончании атаки, это количество "просроченных" полуоткрытых соединений. В терминах вышеизложенных структур данных это количество элементов в деревьях, индексы которых больше определенного граничного значения, определяемого так же в функции инициализации модуля InitTcpConnEstTimeChecker:

// the index of the first node with overdued connections

checker->firstOverduedNodeIndex = checker->overdueTime / checker->checkPeriod;

Так же стоит отметить, что модуле реализована обработка RST пакетов приходящих как от защищаемого сервера, так и от клиента.

Описанный выше модуль в программной реализации представлен в виде следующей структуры и функций работы с ним:

typedef struct \_TcpConnEstTimeChecker

{

/\*\*\* Опции правила \*\*\*/

// time in seconds after which the half-open connection is overdue

int overdueTime;

// period in seconds to check the number of overdue half-open connections

int checkPeriod;

// the max allowed number of half-open connections

int overdueUpperBound;

// the diviation of overdueUpperBound

int overdueUpperBoundDiviation;

/\*\*\* Внутренние данные \*\*\*/

// the number of root nodes in the array

int rootNodesCount;

// the array of root nodes

ubi\_btRoot\* rootNodes;

// the index of the first node, which contains overdued connections

int firstOverduedNodeIndex;

// time when the last shift was made

time\_t lastShiftTime;

// Indicates if Syn Flood atack presents

int atackState;

}

TcpConnEstTimeChecker;

/\*\*\*Интерфейс\*\*\*/

void InitTcpConnEstTimeChecker(TcpConnEstTimeChecker\* checker, int \_overdueTime,

int \_checkPeriod, int \_overdueUpperBound, int \_overdueUpperBoundDiviation,

int \_serverTimeout);

void DeInitTcpConnEstTimeChecker(TcpConnEstTimeChecker\* checker);

int TcpConnEstTimeChecker\_ProcessPacket(TcpConnEstTimeChecker\* checker, Packet\* p, int packetType);

int ShiftRootNodes(TcpConnEstTimeChecker\* checker, int GenerationCount);

5.3.2 Структура модуля TcpSynFloodPreventionModule

Как упоминалось в разделе 2, на данный момент общедоступные методы противодействия TCP SYN атаке не отличаются эффективностью и обоснованностью выбора значений конкретных параметров при настройке системы. Т.к. на данный момент авторами не разработана модель, позволяющая построить математически обоснованную методику противодействия атаке, то было принято решение реализовать эту функциональность без соответствующей базы. При этом одним из требований к разработке является возможность работы с модулем посредством определенного интерфейса, не зависящего от конкретной реализации. Такое требование в дальнейшем позволит легко подключать различные (более эффективные) реализации. В объектно-ориентированных языках такой подход обычно реализуется соответствующими механизмами (наследование, интерфейсы и т.д.), однако язык С, на котором написан сам Snort таких возможностей не предоставляет. Поэтому приведение к единому интерфейсу реализовано по аналогии с бинарными деревьями Snort, в виде макроопределений, которые используются для работы с модулем. Такой подход затрудняет разработку и отладку приложения, но приводит к большему быстродействию конечного продукта.

Для реализации этого интерфейса модули предотвращения должны реализовывать три функции:

* Функция инициализации модуля. Предназначена для выделения памяти под внутренние структуры данных и их инициализацию
* Функция деинициализации модуля. Предназначена для освобождения памяти, выделенной под внутренние структуры данных
* Функция обработки пакета. В конкретной реализации эта функция при условии отсутствия TCP SYN атаки занимается ведением положительной статистики. В условиях присутствия атаки обновление статистики приостанавливается, а данная функция принимает решение о том пропускать ли этот пакет дальше или нет.

В текущей реализации статистика "положительных клиентов" ведется по учету количества успешно установленных TCP соединений с конкретного IP адреса, от которого пакеты приходят с определенным значением TTL. Учет значения TTL затрудняет возможному злоумышленнику имитацию "положительного" клиента. Например, злоумышленник знает, что у администратора системы определенный IP. В этом случае для введения системы предотвращения атаки в заблуждения он может генерировать множество TCP SYN пакетов, указывая в качестве адреса отправителя известный ему IP адрес. Если же система так же учитывает значение TTL, то злоумышленнику для успешной имитации так же необходимо знать количество маршрутизаторов находящихся между защищаемым сервером и машиной администратора.

Реализация этого модуля так же основана на использовании бинарных деревьев Snort. Внутренняя структура данных модуля имеет вид:

интернет атака программный snort

typedef struct \_TcpSynFloodPreventionModule

{

// корень дерева со статистикой

ubi\_btRootPtr rootStat;

long totalPacketsCount;

} TcpSynFloodPreventionModule;

Как видно из объявления структуры вся статистика хранится в одном дереве. Кроме того хранится общее количество обработанных модулем ACK пакетов. Оно используется при определении того, пропускать ли пакет или нет.

Следующая структура представляет собой данные, которые хранятся в узлах дерева:

typedef struct \_TcpSynFloodPreventionStatTreeNodeData

{

// узел дерева

ubi\_trNode Node;

// Поля идентифицирующие клиента

u\_int8\_t ttl;

struct in\_addr ipSrc;

// Количество пакетов удовлетворяющих условию. TTL=ttl and IPSrc=ipSrc

long counter;

} TcpSynFloodPreventionStatTreeNodeData;

Эта структура содержит следующие поля:

* Node – структура представляющая узел бинарного дерева Snort. Это поле должно быть первым в объявлении, т.к. это обусловлено оптимизацией во внутренней реализации деревьев.
* ttl – значение TTL для узла
* ipSrc – значение IP адреса клиента
* counter – количество обработанных ACK пакетов, пришедших о клиента с IP адресом ipSrc и значением TTL=ttl.

При такой организации внутренних структур данных решение о том, стоит ли пропускать пакет в случае присутствия атаки, принимается исходя из следующего соотношения:

currNodeData->counter > (module->totalPacketsCount / ubi\_trCount(module->rootStat));

5.3.3 Взаимодействие TcpConnEstTimeChecker и TcpSynFloodPreventionModule в реализации tcp\_syn\_flood

Для дальнейшего описания реализации следует привести структуру, в которой хранится внутреннее состояние tcp\_syn\_flood:

typedef struct \_TcpSynFloodData

{

// текущий режим работы

int workingMode;

// IP адрес защищаемого сервера

struct in\_addr serverIP;

// указатель на модуль TcpConnEstTimeChecker

TcpConnEstTimeChecker\* timeChecker;

// указатель на модуль TcpSynFloodPreventionModulePtr

TcpSynFloodPreventionModulePtr preventionModule;

} TcpSynFloodData;

Как видно из объявления кроме знакомых нам TcpConnEstTimeChecker и TcpSynFloodPreventionModulePtr здесь присутствует переменная workingMode, которая хранит текущее состояние. Она может принимать следующие значения, определенные в файле sp\_tcp\_syn\_flood.h:

/\*\*\* Режимы работы модуля \*\*\*/

#define TCP\_SYN\_FLOOD\_DETECTION1

#define TCP\_SYN\_FLOOD\_PREVENTION2

Очевидно, что работа каждого из модулей начинается с инициализации, которая происходит при инициализации tcp\_syn\_flood. Как было показано выше, за инициализацию модуля отвечает функция TcpSynFloodInit. Внутри нее происходит вызов функции TcpSynFloodRuleParseFunction, которая производит анализ параметров, указанных в правиле. Считается, что во время запуска Snort TCP SYN атака отсутствует и переменной workingMode присваивается значение TCP\_SYN\_FLOOD\_DETECTION.

Самая важная функция это TcpSynFloodCheckFunction, которая вызывается для обработки всех пакетов. В ней первым делом определяется тип пришедшего пакета. Возможные типы представлены соответствующими макроопределениями:

/\*\*\* Поддерживаемые типы пакетов \*\*\*/

#define PACKET\_TYPE\_UNSUPPORTED 0

#define PACKET\_TYPE\_SYN\_ACK 1

#define PACKET\_TYPE\_ACK 2

#define PACKET\_TYPE\_RST\_FROM\_SERVER 3

#define PACKET\_TYPE\_RST\_FROM\_CLIENT 4

#define PACKET\_TYPE\_SYN 5

Как видно из приведенного выше фрагмента на внутреннее состояние модуля влияют пришедшие от клиента пакеты с установленными комбинациями флагов SYN, ACK и RST, а от защищаемого сервера – SYN+ACK и RST. Различия между RST пакетами введены для того, чтобы корректно уменьшать количество полуоткрытых соединений. Следующий фрагмент кода демонстрирует это:

switch(packetType){

case PACKET\_TYPE\_ACK:

findNodeData->SeqAckNumber = p->tcph->th\_ack-1;

break;

case PACKET\_TYPE\_RST\_FROM\_SERVER:

findNodeData->SeqAckNumber = p->tcph->th\_ack-1;

break;

case PACKET\_TYPE\_SYN\_ACK:

findNodeData->SeqAckNumber = p->tcph->th\_seq-1;

break;

}

После того, как определен тип пришедшего пакета, он передается в рассмотренный выше модуль TcpConnEstTimeChecker, функция TcpConnEstTimeChecker\_ProcessPacket которого сообщает основному модулю о том, есть ли в данный момент атака или нет. Она может вернуть одно из двух следующих значений:

/\*\*\* Состояния атаки \*\*\*/

#define SYN\_ATACK\_IS\_NOT\_PRESENT 1

#define SYN\_ATACK\_IS\_PRESENT 2

Далее идет проверка того, нужно ли сгенерировать какое-нибудь сообщение и изменить текущий режим работы. Это иллюстрируется следующим фрагментом кода:

if(checkerResult == SYN\_ATACK\_IS\_PRESENT){

// Если атака только началась

if(tcpSynFloodData->workingMode == TCP\_SYN\_FLOOD\_DETECTION){

// Генерируем сообщение 'Atack Started'

GenerateSnortEvent(NULL, GENERATOR\_TCP\_SYN\_FLOOD, 0,0,0,3,SYNFLOOD\_STARTED);

//изменяем режим

tcpSynFloodData->workingMode = TCP\_SYN\_FLOOD\_PREVENTION;

}

}

else{

// Если атака только закончилась

if(tcpSynFloodData->workingMode == TCP\_SYN\_FLOOD\_PREVENTION){

// генерируем сообщение "ATACK FINISHED"

GenerateSnortEvent(NULL, GENERATOR\_TCP\_SYN\_FLOOD, 0,0,0,3,SYNFLOOD\_FINISHED);

//изменяем режим

tcpSynFloodData->workingMode = TCP\_SYN\_FLOOD\_DETECTION;

}

}

После этого идет проверка того, должен ли этот пакет быть обработан модулем предотвращения. Если это ACK пакет, пришедший от клиента, то он должен увеличить значение счетчика в статистике для данного клиента, а если это SYN пакет, то в зависимости от текущего режима работы модуля этот пакет либо будет пропущен, либо удален.

if(preventionResult == PREVENTION\_PACKET\_IS\_BAD){

if(InlineMode()){

InlineDrop();

}

}

Как видно из приведенного фрагмента в случае "плохого" пакета осуществляется проверка того, в каком режиме работает Snort. Если в режиме inline, то доступен метод InlineDrop, который говорит Snort о том, что этот пакет пропускать нельзя. В качестве дальнейшего развития этого модуля возможна реализация активного ответа для Snort работающего в режиме простого обнаружения. В качестве такого активного ответа может быть рассылка серверу и клиенту RST пакетов для закрытия этого соединения.

Как отмечалось выше, исходные значения параметров для модуля задаются администратором в виде правила, которое имеет следующий вид:

#tcp\_syn\_flood:

# [0] - ip\_server

# [1] - server\_timeout\_sec

# [2] - max\_overdue\_count

# [3] - max\_overdue\_count\_diviation

# [4] - overdue\_time\_sec

# [5] - check\_period\_sec

pass tcp any any <> any any (tcp\_syn\_flood:172.20.24.20, 60, 10, 2, 5, 1;)

Конкретное значение параметра max\_overdue\_count можно определить с помощью соотношения (3.14) . Для этого сначала с помощью специальных утилит [22] необходимо найти значения интенсивности входящего потока и вероятности потери пакетов в сети.

Рассмотрим пример, когда интенсивность обращений к серверу λ равна 1000 заявок в секунду, вероятность потери пакетов в сети Pпп 0.01, значение 1/μ (отведенный на сервере таймаут ) примем равным 60 секундам, и количество попыток повторной передачи SYN+ACK пакетов  равным 2.

Подставив такие значения в соотношение (3.14) получим:



Т.е. значение параметра max\_overdue\_count в правиле Snort следует принять равным 24.

Описанная выше программная реализация модуля расширения функциональности для системы предотвращения вторжений Snort\_inline может быть использована для защиты критичных ресурсов корпоративной сети. Для этого необходимо создать программно-аппаратный комплекс, представляющий из себя маршрутизатор на базе Linux системы с установленным IPS Snort\_inline. Во время разработки модуля акцент делался на независимость от операционной системы, но на данный момент тестирование проводилось только на ОС Linux.

К достоинствам такой реализации можно отнести возможность использовать один комплекс для защиты нескольких серверов, т.к. параметры защиты для конкретного сервера задаются в отдельном правиле Snort, допустимое количество которых ограничено лишь вычислительными возможностями оборудования.

ВЫВОДЫ

На современном этапе развития информационных технологий всё больше проявляется зависимость эффективного функционирования государственных и коммерческих предприятий и организаций от безопасности и надёжности применяемых корпоративных информационно-телекоммуникационных систем. Среди основных требований, предъявляемых к таким системам, можно выделить необходимость обеспечения услуги доступности. В виду того, что одним из самых распространенных видов атак в современных сетях являются атаки типа отказа в обслуживании, которые при успешной реализации способны парализовать работу как отдельных серверов, так и целых сетей, проблема обеспечения доступности ресурсов является чрезвычайно актуальной для общедоступных информационных систем, в частности, сети Internet.

В настоящее время существуют различные механизмы обнаружения и противодействия TCP SYN атаке. Наиболее эффективным из них является метод SYN Cookies, однако он, как и все другие, имеет недостатки, такие как необходимость внесения соответствующих изменений в реализацию стека протоколов TCP/IP на защищаемом сервере, недостаточная эффективность обнаружения атаки из-за отсутствия методики выбора конкретных значений для параметров защиты. В связи с этим, проблема TCP SYN атак требует новых эффективных решений.

В результате проделанной работы была разработана методика обнаружения TCP SYN атаки, позволяющая обнаруживать атаку на ранних стадиях. В основе предложенной методики лежит математическая модель, описывающая обслуживание сервером потока заявок на установление TCP соединения. С помощью математического аппарата теории систем массового обслуживания находятся допустимые интервалы значений для количества полуоткрытых TCP соединений на сервере, работающем в нормальном режиме (при условии отсутствия атаки). В соответствии с этим методом, решение о начале атаки принимается в том случае, когда реальное количество полуоткрытых на сервере соединений выходит за пределы допустимого интервала. Для определения границ этого интервала необходимо получить значения таких параметров как: интенсивность входного потока заявок, время, в течение которого с заданной вероятностью заявка будет обслужена (время прихода ACK пакета), вероятность потери IP пакетов при передаче по сети, количество попыток повторного отправления сервером SYN + ACK пакетов, таймаут отведенный на сервере на установку TCP соединения и вероятность верного обнаружения атаки. Часть параметров можно определить, используя предложенные методики сбора данных, часть определяется настройками стека протоколов TCP/IP на защищаемом сервере.

К достоинствам разработанной методики можно отнести то, что она позволяет обнаружить атаку на начальном этапе, устойчива к резкому возрастанию интенсивности входного потока запросов к серверу, учитывает характеристики сети и защищаемого сервера.

Недостатком данной методики является то, что неисправности сетевого оборудования, в результате которых повышается вероятность потери пакета в сети, будут интерпретированы как TCP SYN атака.

Предложенная методика реализована в виде программно-аппаратного комплекса, состоящего из маршрутизатора на базе Linux системы и системы предотвращения вторжений Snort\_inline c модулем расширения функциональности, которые представлен в виде двух подмодулей отвечающих за обнаружение и предотвращение атаки соответственно.

К достоинствам такой реализации можно отнести возможность использовать один программно-аппаратный комплекс для одновременной защиты нескольких серверов, т.к. параметры защиты для конкретного сервера задаются в отдельном правиле Snort, допустимое количество которых ограничено лишь вычислительными возможностями оборудования. Так же модульная архитектура позволяет довольно легко менять реализации модуля предотвращения.

Разработанное решение направлено на защиту критических ресурсов корпоративной сети, таких как сервера прикладных сервисов (Web, электронная почта, службы аутентификации и т.д) от TCP SYN атак.

Перспективой дальнейших исследований является разработка адаптивного метода оценки входных параметров в реальном масштабе времени. Так же перспективным является рассмотрение возможности использования разработанной методики для противодействия атакам, направленным на исчерпание всей пропускной способности канала связи, а так же адаптации ее для противодействия другим DoS/DDoS атакам.

ПЕРЕЧЕНЬ ССЫЛОК

1. Постанова Кабінету Міністрів України від 28.10.2004 р. №1452 "Про затвердження Порядку застосування електронного цифрового підпису органами державної влади, органами місцевого самоврядування, підприємствами, установами та організаціями державної форми власності"
2. Закон України "Про електронні документи та електронний документообіг" від 22.05.2003 р.
3. НД ТЗІ 1.1-003-99. ТЕРМІНОЛОГІЯ В ГАЛУЗІ ЗАХИСТУ ІНФОРМАЦІЇ В КОМП’ЮТЕРНИХ СИСТЕМАХ ВІД НЕСАНКЦІОНОВАНОГО ДОСТУПУ
4. http://bezpeka.com
5. <http://www.virulist.com>
6. http://www.void.ru
7. <http://www.webinform.ru>
8. http://bugtraq.ru
9. Приказ ДСТСЗИ СБУ 30.04.2004 N 31
10. BSI.IT Baseline Protection Manual. Standard Security Measures. Version: October 2000
11. RFC793 Transmission Control Protocol
12. <http://www.securityfocus.com/infocus/1729>
13. http://www.protocols.ru
14. <http://www.preferredtechnology.com>
15. http://www.iss.net
16. Г.И. Ивченко. Теория массового обслуживания, М:Высшая Школа, 1982.
17. RFC2616. Hypertext Transfer Protocol – HTTP/1.1.
18. Д. Камер. Сети TCP/IP том 1, изд. дом "Вильямс" М-Санкт-Петербург-Киев 2003
19. <http://www.internettrafficreport.com/main.htm>
20. http://securitylab.ru
21. Г.Корн, Т.Корн Справочник по математике для научных работников и инженеров М: "НАУКА",1968.
22. Д.Л. Ясницкий. "Разработка методики раннего обнаружения и отражения распределённых атак типа "отказ в обслуживании"". Магистерская аттестационная работа. Харьков: ХНУРЭ, 2006
23. Джей Бил и др. Snort 2.1. Обнаружение вторжений. 2-е изд. Пер. с англ. - М.: ООО "Бином-Пресс", 2006;
24. Медведовский И.Д., Семьянов П.В., Платонов В.В. Атака через Internet.
25. <http://www.securityfocus.com>
26. Д.Л. Ясницкий, В.Д. Литовский, Р.В. Олейников. Методика раннего обнаружения TCP SYN атаки. Прикладная радиоэлектроника. Т.5, Харьков: ХНУРЭ, 2006.
27. Snort-2.4.3 source code.
28. http://kiev-security.org.ua
29. <http://www.sophist.demon.co.uk/ping>
30. http://www.citforum.ru
31. [www.computer.org/](http://www.computer.org/)
32. http://wiki.hping.org/172
33. <http://www.snort.org>
34. http://www.bsi.de/
35. <http://www.linuxsecurity.com>

Приложение А

Исходный код модуля расширение функциональности для IPS Snort\_inline

// Файл tcp\_syn\_flood.h

#ifndef \_\_SP\_TCP\_SYN\_FLOOD\_H\_\_

#define \_\_SP\_TCP\_SYN\_FLOOD\_H\_\_

// TcpSynFloodDefinitions

#define TcpSynFloodPreventionModulePtr void\*

/\*\*\* Pluging working modes \*\*\*/

#define TCP\_SYN\_FLOOD\_DETECTION1

#define TCP\_SYN\_FLOOD\_PREVENTION2

/\*\*\* TcpSynFlood atack states \*\*\*/

#define SYN\_ATACK\_IS\_NOT\_PRESENT 1

#define SYN\_ATACK\_IS\_PRESENT 2

/\*\*\* Supported packet types \*\*\*/

#define PACKET\_TYPE\_UNSUPPORTED 0

#define PACKET\_TYPE\_SYN\_ACK 1

#define PACKET\_TYPE\_ACK 2

#define PACKET\_TYPE\_RST\_FROM\_SERVER 3

#define PACKET\_TYPE\_RST\_FROM\_CLIENT 4

#define PACKET\_TYPE\_SYN 5

/\*\*\* Prevention module checking packet results \*\*\*/

#define PREVENTION\_PACKET\_IS\_OK 1

#define PREVENTION\_PACKET\_IS\_BAD 2

void SetupTcpSynFlood();

#endif /\* \_\_SP\_TCP\_SYN\_FLOOD\_H\_\_ \*/

// Файл tcp\_syn\_flood.с

#include <sys/types.h>

#include <stdlib.h>

#include <ctype.h>

#include "rules.h"

#include "decode.h"

#include "plugbase.h"

#include "parser.h"

#include "debug.h"

#include "util.h"

#include "plugin\_enum.h"

#include "generators.h"

#include "event\_wrapper.h"

#ifdef HAVE\_STRINGS\_H

#include "mstring.h"

#endif

#include "sp\_tcp\_syn\_flood.h"

#include "tcp\_conn\_est\_time\_checker.h"

#include "tcp\_syn\_flood\_prevention\_stat.h"

/\*

\* setup any data structs here

\*/

typedef struct \_TcpSynFloodData

{

// the current mode of the plugin

int workingMode;

// the IP address of the server being protected

struct in\_addr serverIP;

// tcp connection estimate time checker

TcpConnEstTimeChecker\* timeChecker;

// prevention module

TcpSynFloodPreventionModulePtr preventionModule;

} TcpSynFloodData;

/\* function prototypes go here \*/

static void TcpSynFloodInit(char \*, OptTreeNode \*, int);

static void TcpSynFloodRuleParseFunction(char \*, OptTreeNode \*);

static int TcpSynFloodCheckFunction(Packet \*, struct \_OptTreeNode \*, OptFpList \*);

/\* internal functions prototypes \*/

int ParseIntElement(char\* token, char \*name);

void SetupTcpSynFlood()

{

/\* map the keyword to an initialization/processing function \*/

RegisterPlugin("tcp\_syn\_flood", TcpSynFloodInit);

DEBUG\_WRAP(DebugMessage(DEBUG\_PLUGIN,"Plugin: TcpSynFlood Setup\n"););

}

static void TcpSynFloodInit(char \*data, OptTreeNode \*otn, int protocol)

{

// multiple declaration check

if(otn->ds\_list[PLUGIN\_TCP\_SYN\_FLOOD])

{

FatalError("%s(%d): Multiple tcpsynflood options in rule\n", file\_name,

file\_line);

}

// allocate the data structure and attach it to the

// rule's data struct list

TcpSynFloodData\* tcpSynFloodData = (TcpSynFloodData\*) SnortAlloc(sizeof(TcpSynFloodData));tcpSynFloodData->workingMode = TCP\_SYN\_FLOOD\_DETECTION;

tcpSynFloodData->preventionModule = TcpSynFloodPreventionCreateModule();

otn->ds\_list[PLUGIN\_TCP\_SYN\_FLOOD] = tcpSynFloodData;

TcpSynFloodRuleParseFunction(data, otn);

// finally, attach the option's detection function to the rule's

// detect function pointer list

AddOptFuncToList(TcpSynFloodCheckFunction, otn);

}

/\*\* Expected Rule Structure

# tcp\_syn\_flood:

# [0] - ip\_server

# [1] - server\_timeout\_sec

# [2] - max\_overdue\_count

# [3] - max\_overdue\_count\_diviation

# [4] - overdue\_time\_sec

# [5] - check\_period\_sec

\*/

static void TcpSynFloodRuleParseFunction(

char \*data,

OptTreeNode \*otn)

{

int server\_timeout\_sec = 0;

int max\_overdue\_count = 0;

int max\_overdue\_count\_diviation = 0;

int overdue\_time\_sec = 0;

int check\_period\_sec = 0;

TcpSynFloodData \*tcpSynFloodData;

tcpSynFloodData = otn->ds\_list[PLUGIN\_TCP\_SYN\_FLOOD];

while(isspace((int)\*data))

data++;

int numTokens;

const int TokensCount = 6;

char \*\*tokens = mSplit(data, ",", TokensCount, &numTokens, 0);

printf("numtokens %d\n", numTokens );

if(numTokens != TokensCount)

{

FatalError("in TcpSynFlood rule: invalid number of init parameters\n");

}

if(inet\_aton(tokens[0], &tcpSynFloodData->serverIP) == 0)

{

FatalError("in TcpSynFlood rule: %s is invalid ip address\n", tokens[0]);

}

server\_timeout\_sec = ParseIntElement(tokens[1], "server timeout");

max\_overdue\_count = ParseIntElement(tokens[2], "overdue count");

max\_overdue\_count\_diviation = ParseIntElement(tokens[3], "overdue count diviation");

overdue\_time\_sec = ParseIntElement(tokens[4], "overdue time (in seconds)");

check\_period\_sec = ParseIntElement(tokens[5], "check period (in seconds)");

// init checker

TcpConnEstTimeChecker\* checker = (TcpConnEstTimeChecker\*)SnortAlloc(sizeof(TcpConnEstTimeChecker));

InitTcpConnEstTimeChecker(checker,

overdue\_time\_sec, // overdueTime

check\_period\_sec, // int \_checkPeriod

max\_overdue\_count, // overdueUpperBound

max\_overdue\_count\_diviation, // overdueUpperBoundDiviation

server\_timeout\_sec// serverTimeout

);

tcpSynFloodData->timeChecker = checker;

printf("TcpSynFlood module initialized with the following parameters:\n");

printf("\tserver timeout %d\n", server\_timeout\_sec);

printf("\tmax overdue count %d\n", max\_overdue\_count);

printf("\tmax overdue count diviation %d\n", max\_overdue\_count\_diviation);

printf("\toverdue time %d\n", overdue\_time\_sec);

printf("\theck period %d\n", check\_period\_sec);

// free tokens memory

mSplitFree(&tokens, numTokens);

}

static int TcpSynFloodCheckFunction(

Packet \*p,

struct \_OptTreeNode \*otn,

OptFpList \*fp\_list)

{

// Get Rule Data

TcpSynFloodData \*tcpSynFloodData;

tcpSynFloodData = otn->ds\_list[PLUGIN\_TCP\_SYN\_FLOOD];

int packetType = GetPacketType(tcpSynFloodData->timeChecker, p, tcpSynFloodData->serverIP);

if(packetType != PACKET\_TYPE\_UNSUPPORTED)

{

// process ACK packets with prevention module

if(packetType == PACKET\_TYPE\_ACK)

{

printf("Processing ACK\n");

// check if atack is absent

int changeStat = (tcpSynFloodData->workingMode == TCP\_SYN\_FLOOD\_DETECTION) ? CHANGE\_STAT\_YES : CHANGE\_STAT\_NO;

if(changeStat == CHANGE\_STAT\_YES)

{

// update statistics for "good" client

TcpSynFloodPreventionProcessPacket(tcpSynFloodData->preventionModule, p, changeStat);

}

} // process SYN packet with prevetntion module

else if(packetType == PACKET\_TYPE\_SYN)

{

printf("processing SYN\n");

// check if atack is present

if(tcpSynFloodData->workingMode == TCP\_SYN\_FLOOD\_PREVENTION)

{

printf("processing SYN while attack is present\n");

// get packet status

int packetStatus = TcpSynFloodPreventionProcessPacket(tcpSynFloodData->preventionModule, p, CHANGE\_STAT\_NO);

if(packetStatus == PREVENTION\_PACKET\_IS\_BAD)

{

printf("Processing bas SYN\n");

if(InlineMode())

{

InlineDrop();

}

// else {} // Another type of ActiveReply should be implemented

// For example sending RST packets to server and client

return 0;

}

}

}

// process packet with time checker

int checkerResult = TcpConnEstTimeChecker\_ProcessPacket(tcpSynFloodData->timeChecker, p, packetType);

if(checkerResult == SYN\_ATACK\_IS\_PRESENT)

{

// Check if atack has been started now

if(tcpSynFloodData->workingMode == TCP\_SYN\_FLOOD\_DETECTION)

{

// Generate log message 'Atack Started'

GenerateSnortEvent(NULL, GENERATOR\_TCP\_SYN\_FLOOD, 0,0,0,3,SYNFLOOD\_STARTED);

//change mode

tcpSynFloodData->workingMode = TCP\_SYN\_FLOOD\_PREVENTION;

}

}

else

{

// Check if atack has been finished now

if(tcpSynFloodData->workingMode == TCP\_SYN\_FLOOD\_PREVENTION)

{

// Generate event log "ATACK FINISHED"

GenerateSnortEvent(NULL, GENERATOR\_TCP\_SYN\_FLOOD, 0,0,0,3,SYNFLOOD\_FINISHED);

//change mode

tcpSynFloodData->workingMode = TCP\_SYN\_FLOOD\_DETECTION;

}

}

}// PACKET IS SUPPORTED

return fp\_list->next->OptTestFunc(p, otn, fp\_list->next);

}

int ParseIntElement(char\* token, char \*name)

{

char \* tail;

int value = (int) strtol(token, &tail, 10);

if(\*tail)

{

FatalError("in TcpSynFlood rule: %s is invalid %s.\n", token, name);

}

return value;

}

inline int GetPacketType(TcpConnEstTimeChecker\* checker, Packet\* p, struct in\_addr ipServer)

{

// check IP address

struct in\_addr ipSrc = p->iph->ip\_src;

struct in\_addr ipDst = p->iph->ip\_dst;

u\_int8\_t flags = p->tcph->th\_flags;

if((ipDst.s\_addr == ipServer.s\_addr) && ((flags ^ R\_SYN) == 0))

{

return PACKET\_TYPE\_SYN;

}

if((ipSrc.s\_addr == ipServer.s\_addr) && ((flags ^ R\_SYN ^ R\_ACK) == 0))

{

return PACKET\_TYPE\_SYN\_ACK;

}

else if((ipDst.s\_addr == ipServer.s\_addr) && ((flags ^ R\_ACK) == 0))

{

return PACKET\_TYPE\_ACK;

}

else if((ipSrc.s\_addr == ipServer.s\_addr) && ((flags ^ R\_RST) == 0))

{

return PACKET\_TYPE\_RST\_FROM\_SERVER;

}

else if((ipDst.s\_addr == ipServer.s\_addr) && ((flags ^ R\_RST) == 0))

{

return PACKET\_TYPE\_RST\_FROM\_CLIENT;

}

return PACKET\_TYPE\_UNSUPPORTED;

}

// файл tcp\_conn\_est\_time\_checker.h

#ifndef \_\_TCP\_CONN\_EST\_TIME\_CHECKER\_H\_\_

#define \_\_TCP\_CONN\_EST\_TIME\_CHECKER\_H\_\_

#include <time.h>

#include <sys/time.h>

#include "config.h"

#include "decode.h"

#include "ubi\_SplayTree.h"

typedef struct \_TcpConnEstTimeChecker

{

/\*\*\* Rule Options \*\*\*/

// time in seconds after which the half-open connection is overdue

long overdueTime;

// period in seconds to check the number of overdue half-open connections

long checkPeriod;

// the max allowed number of half-open connections

int overdueUpperBound;

// the diviation of overdueUpperBound

int overdueUpperBoundDiviation;

/\*\*\* Internal Data \*\*\*/

// the number of root nodes in the array

int rootNodesCount;

// the array of root nodes

ubi\_btRoot\* rootNodes;

// the index of the first node, which contains overdued connections

int firstOverduedNodeIndex;

// time when the last shift was made

struct timeval lastShiftTime;

// Indicates if Syn Flood atack presents

int atackState;

}

TcpConnEstTimeChecker;

/\*\*\* Inerface \*\*\*/

void InitTcpConnEstTimeChecker(TcpConnEstTimeChecker\* checker, long \_overdueTime,

long \_checkPeriod, int \_overdueUpperBound,

int \_overdueUpperBoundDiviation, long \_serverTimeout);

void DeInitTcpConnEstTimeChecker(TcpConnEstTimeChecker\* checker);

int TcpConnEstTimeChecker\_ProcessPacket(TcpConnEstTimeChecker\* checker, Packet\* p, int packetType);

int ShiftRootNodes(TcpConnEstTimeChecker\* checker, int GenerationCount);

#endif /\* \_\_SP\_TCP\_SYN\_FLOOD\_H\_\_ \*/

// файл tcp\_conn\_est\_time\_checker.c

#ifndef \_\_TCP\_CONN\_EST\_TIME\_CHECKER\_H\_\_

#include "tcp\_conn\_est\_time\_checker.h"

#endif

#include "sp\_tcp\_syn\_flood.h"

#include <memory.h>

#include <math.h>

#include <stdlib.h>

#include "rules.h"

#include "util.h"

/\*\*\*\*\*\*\*\*\* States of Timechecker Tree Node Data \*\*\*\*\*\*\*\*/

#define NODE\_STATE\_SYN\_RECEIVED 1

#define NODE\_STATE\_SYN\_ACK\_RECEIVED 2

typedef struct \_TimeCheckerTreeNodeData

{

ubi\_trNode Node;

// state of the node

int NodeState;

// Sequence number for client SYN packet

u\_int32\_t ClientNumber;

// Sequence number for server SYN+ACK packet

u\_int32\_t ServerNumber;

}

TChTreeNodeData;

/\*\*\* TChTreeNodeData manipulation functions \*\*\*/

static int TChTreeNodeDataCompareFunc(ubi\_trItemPtr ItemPtr, ubi\_trNodePtr NodePtr);

static void TChTreeNodeDataDeleteNode(ubi\_btNodePtr NodePtr);

/\*\*\* TcpConnEstTimeChecker manipulation functions \*\*\*/

void InitTcpConnEstTimeChecker(TcpConnEstTimeChecker\* checker, long \_overdueTime,

long \_checkPeriod, int \_overdueUpperBound,

int \_overdueUpperBoundDiviation, long \_serverTimeout)

{

CheckInitParams(\_overdueTime, \_checkPeriod, \_overdueUpperBound, \_overdueUpperBoundDiviation,\_serverTimeout);

checker->overdueTime = \_overdueTime;

checker->checkPeriod = \_checkPeriod;

checker->overdueUpperBound = \_overdueUpperBound;

checker->overdueUpperBoundDiviation = \_overdueUpperBoundDiviation;

// Get rootNodes count

double serverTimeout = \_serverTimeout;

int rootNodesCount = ceil(serverTimeout / \_checkPeriod);

checker->rootNodesCount = rootNodesCount;

printf("NODES COUNT %d\n", rootNodesCount);

// init the array of root nodes

checker->rootNodes = (ubi\_btRoot\*)SnortAlloc(sizeof(ubi\_btRoot) \* rootNodesCount);

// the index of the first node with overdued connections

checker->firstOverduedNodeIndex = checker->overdueTime / checker->checkPeriod;

int i;

for(i = 0; i < rootNodesCount; i++)

{

ubi\_trInitTree(checker->rootNodes + i,/\* ptr to the tree head \*/

TChTreeNodeDataCompareFunc, /\* comparison function \*/

ubi\_trDUPKEY);

//0); /\* do not allow nither OVERWRITE nor DUPLICATES \*/

}

// get current time

struct timezone tz;

gettimeofday(&checker->lastShiftTime, &tz);

//time(&checker->lastShiftTime);

checker->atackState = SYN\_ATACK\_IS\_NOT\_PRESENT;

}

void DeInitTcpConnEstTimeChecker(TcpConnEstTimeChecker\* checker)

{

int rootNodesCount = checker->rootNodesCount;

// delete trees

int i;

for(i = 0; i < rootNodesCount; i++)

{

ubi\_trKillTree(checker->rootNodes + i, TChTreeNodeDataDeleteNode);

}

// delete array

free(checker->rootNodes);

checker->rootNodes = NULL;

}

long GetTimeDifference(struct timeval\* time1, struct timeval\* time2)

{

long secDiff =time1->tv\_sec - time2->tv\_sec;

long micSecDiff = time1->tv\_usec - time2->tv\_usec;

return labs(secDiff)\*1000000 + labs(micSecDiff);

}

int TcpConnEstTimeChecker\_ProcessPacket(TcpConnEstTimeChecker\* checker, Packet\* p, int packetType)

{

int i;

/\*

// get current time

time\_t curTime;

time(&curTime);

//check the time

int diff = difftime(curTime, checker->lastShiftTime);

\*/

struct timeval currTime;

struct timezone zone;

gettimeofday(&currTime, &zone);

long diff = GetTimeDifference(&currTime, &checker->lastShiftTime);

if(diff >= checker->checkPeriod)

{

// shift trees

printf("shifting trees\n");

int generationsCount = ((float)diff) / checker->checkPeriod;

ShiftRootNodes(checker, generationsCount);

}

// is used as item to search

TChTreeNodeData\* findNodeData = NULL;

// flag which indicates if the node is inserted successfully

ubi\_trBool insertResult;

// if Syn add node to the tree

if(packetType == PACKET\_TYPE\_SYN)

{

printf("processing SYN packet in time checker \n");

TChTreeNodeData\* newNodeData = (TChTreeNodeData\*)SnortAlloc(sizeof(TChTreeNodeData));

ubi\_trNodePtr nodePtr = &newNodeData->Node;

ubi\_btInitNode(nodePtr);

// save sequence number and set NODE\_STATE\_SYN\_RECEIVED

newNodeData->ClientNumber = p->tcph->th\_seq;

newNodeData->NodeState = NODE\_STATE\_SYN\_RECEIVED;

// trying to insert the node to the 0-th tree

insertResult =

ubi\_trInsert(checker->rootNodes, nodePtr, (ubi\_trItemPtr)newNodeData, NULL);

if(insertResult != ubi\_trTRUE)

{

printf("failed to add SYN to the tree\n");

// there is already the node with the same key in the tree

free(newNodeData);

}

}

// if Syn + Ack

else if(packetType == PACKET\_TYPE\_SYN\_ACK)

{

printf("processing SYN + ACK in time checker\n" );

// find the node that is corresponded to received SYN

findNodeData = (TChTreeNodeData\*)SnortAlloc(sizeof(TChTreeNodeData));

findNodeData->NodeState = NODE\_STATE\_SYN\_RECEIVED;

findNodeData->ClientNumber = p->tcph->th\_ack - 1;

// run over all trees and try to find the node

for(i = 0; i < checker->rootNodesCount; i++)

{

TChTreeNodeData\* foundNodeData = (TChTreeNodeData\*)ubi\_trFind(checker->rootNodes + i, findNodeData);

if(foundNodeData != NULL)

{

// remove node from tree

ubi\_trRemove(checker->rootNodes + i, foundNodeData);

// set Acknowledgement number and update node state to NODE\_STATE\_SYN\_ACK\_RECEIVED

foundNodeData->NodeState = NODE\_STATE\_SYN\_ACK\_RECEIVED;

foundNodeData->ServerNumber = p->tcph->th\_seq;

// insert node again in the tree

insertResult =

ubi\_trInsert(checker->rootNodes + i, foundNodeData, (ubi\_trItemPtr)foundNodeData, NULL);

if(insertResult != ubi\_trTRUE)

{

// there is already the node with the same key in the tree

free(foundNodeData);

}

break;

}

}

free(findNodeData);

}

// if Ack or Resets

else if ((packetType == PACKET\_TYPE\_ACK) ||

(packetType == PACKET\_TYPE\_RST\_FROM\_SERVER) ||

(packetType == PACKET\_TYPE\_SYN\_ACK))

{

TChTreeNodeData\* findNodeData = (TChTreeNodeData\*)SnortAlloc(sizeof(TChTreeNodeData));

switch(packetType)

{

case PACKET\_TYPE\_ACK:

printf("processing ACK packet\n");

findNodeData->NodeState = NODE\_STATE\_SYN\_ACK\_RECEIVED;

findNodeData->ClientNumber = p->tcph->th\_seq - 1;

findNodeData->ServerNumber = p->tcph->th\_ack - 1;

break;

/\*

case PACKET\_TYPE\_RST\_FROM\_SERVER:

printf("processing RST from server\n");

findNodeData->SeqAckNumber = p->tcph->th\_ack-1;

break;

case PACKET\_TYPE\_SYN\_ACK:

printf("processing RST from client\n");

findNodeData->SeqAckNumber = p->tcph->th\_seq-1;

break;

\*/

}

// run over all trees and try to Find and Delete node with the given key

for(i = 0; i < checker->rootNodesCount; i++)

{

ubi\_trNodePtr nodePtr = ubi\_trFind(checker->rootNodes + i, findNodeData);

if(nodePtr != NULL)

{

// delete

ubi\_trRemove(checker->rootNodes + i, nodePtr);

free((TChTreeNodeData\*)nodePtr);

break;

}

}

free(findNodeData);

}

// check overdue connections count

printf("chekcing\n");

return CheckOverdueConnectionsCount(checker);

}

int ShiftRootNodes(TcpConnEstTimeChecker\* checker, int generationCount)

{

int i;

if(generationCount > checker->rootNodesCount)

{

generationCount = checker->rootNodesCount;

}

// free old trees

for( i = (checker->rootNodesCount - generationCount); i < checker->rootNodesCount; i++ )

{

ubi\_trKillTree(checker->rootNodes + i, TChTreeNodeDataDeleteNode);

}

// shift

memmove(checker->rootNodes + generationCount,

checker->rootNodes, (checker->rootNodesCount - generationCount) \* sizeof(ubi\_btRoot));

// init new trees

for(i = 0; i < generationCount; i++)

{

ubi\_trInitTree(&checker->rootNodes[i],/\* ptr to the tree head \*/

TChTreeNodeDataCompareFunc, /\* comparison function \*/

ubi\_trDUPKEY); /\* allow duplicates \*/

}

struct timezone zone;

gettimeofday(&checker->lastShiftTime, &zone);

return 0;

}

int CheckOverdueConnectionsCount(TcpConnEstTimeChecker\* checker)

{

int resCount = 0;

int i;

for(i = checker->firstOverduedNodeIndex; i < checker->rootNodesCount; i++ )

{

resCount += ubi\_trCount(checker->rootNodes + i);

}

int currentlyAllowedBound;

if(checker->atackState == SYN\_ATACK\_IS\_PRESENT)

{

// subtract diviation from the bound

currentlyAllowedBound = checker->overdueUpperBound - checker->overdueUpperBoundDiviation;

}

else

{

// add diviation to the bound

currentlyAllowedBound = checker->overdueUpperBound + checker->overdueUpperBoundDiviation;

}

// save current atack state

checker->atackState = (resCount > currentlyAllowedBound )? SYN\_ATACK\_IS\_PRESENT : SYN\_ATACK\_IS\_NOT\_PRESENT;

printf("check overdued: %d - %d\n", currentlyAllowedBound, resCount);

return checker->atackState;

}

void CheckInitParams(int \_overdueTime,int \_checkPeriod,

int \_overdueUpperBound, int \_overdueUpperBoundDiviation,

int \_serverTimeout)

{

if(\_overdueTime < 0)

{

FatalError("TcpConnectionEstimateTimeChecker:: \_overdueTime must be > 0\n");

}

if(\_checkPeriod < 0)

{

FatalError("TcpConnectionEstimateTimeChecker:: \_checkPeriod must be > 0\n");

}

if(\_overdueUpperBound < 0)

{

FatalError("TcpConnectionEstimateTimeChecker:: \_overdueUpperBound must be > 0\n");

}

if(\_overdueUpperBoundDiviation < 0)

{

FatalError("TcpConnectionEstimateTimeChecker:: \_overdueUpperBoundDiviation must be > 0\n");

}

if((\_overdueUpperBound - \_overdueUpperBoundDiviation) < 0)

{

FatalError("TcpConnectionEstimateTimeChecker:: (\_overdueUpperBound - \_overdueUpperBoundDiviation) must be > 0\n");

}

if((\_overdueUpperBound + \_overdueUpperBoundDiviation) > \_serverTimeout)

{

FatalError("TcpConnectionEstimateTimeChecker:: (\_overdueUpperBound + \_overdueUpperBoundDiviation) must be < \_serverTimeout\n");

}

if(\_serverTimeout < 0)

{

FatalError("TcpConnectionEstimateTimeChecker:: \_serverTimeout must be > 0\n");

}

if(\_overdueTime > \_serverTimeout)

{

FatalError("TcpConnectionEstimateTimeChecker:: overdue time can't be greater than server timeout\n");

}

if(\_serverTimeout <= \_checkPeriod)

{

FatalError("TcpConnectionEstimateTimeChecker:: \_serverTimeout must be greater than \_checkPeriod\n");

}

}

/\* Returns -1 if A < B

Returns 1 if A > B

Returns 0 if A = B

At first Client number is checked.

Than if neccessary Server nubmer is checked

\*/

static int TChTreeNodeDataCompareFunc(ubi\_trItemPtr ItemPtr, ubi\_trNodePtr NodePtr)

{

TChTreeNodeData \*A = (TChTreeNodeData \*) ItemPtr;

TChTreeNodeData \*B = (TChTreeNodeData \*) NodePtr;

if(A->ClientNumber < B->ClientNumber)return -1;

if(A->ClientNumber == B->ClientNumber)return 0;

else // check curr node state

if(B->NodeState == NODE\_STATE\_SYN\_ACK\_RECEIVED)

{

if(A->ServerNumber < B->ServerNumber) return -1;

if(A->ServerNumber == B->ServerNumber) return 0;

else return 1;

}

else // state is NODE\_STATE\_SYN\_RECEIVED

{

return 1;

}

}

static void TChTreeNodeDataDeleteNode(ubi\_btNodePtr NodePtr)

{

free(NodePtr);

}

// файл tcp\_syn\_flood\_prevention\_stat.h

#ifndef \_TCP\_SYN\_FLOOD\_PREVENTION\_STAT\_H\_

#define \_TCP\_SYN\_FLOOD\_PREVENTION\_STAT\_H\_

//#include "config.h"

#include "decode.h"

#include "sp\_tcp\_syn\_flood.h"

#include "ubi\_SplayTree.h"

#define CHANGE\_STAT\_YES 1

#define CHANGE\_STAT\_NO 2

typedef struct \_TcpSynFloodPreventionModule

{

// the root of the statistics tree

ubi\_btRootPtr rootStat;

long totalPacketsCount;

} TcpSynFloodPreventionModule;

// Creates and initializes the prevention module

void\* TcpSynFloodPreventionStatCreateModule();

void TcpSynFloodPreventionStatDeinitModule(TcpSynFloodPreventionModule\* preventionModule);

int TcpSynFloodPreventionStatProcessPacket(TcpSynFloodPreventionModule\* preventionModule, Packet\* packet, int changeStat);

// Unified Tcp Syn Flood prevention interface

#define TcpSynFloodPreventionProcessPacket( module, p, changeStat ) TcpSynFloodPreventionStatProcessPacket( (TcpSynFloodPreventionModule\*) (module) ,(Packet\*) (p), (int) (changeStat) )

#define TcpSynFloodPreventionCreateModule TcpSynFloodPreventionStatCreateModule

#define TcpSynFloodPreventionDeinitModule( module ) TcpSynFloodPreventionStatDeinitModule( (TcpSynFloodPreventionModule\*) (module) )

#endif

// файл tcp\_syn\_flood\_prevention\_stat.c

#ifndef \_TCP\_SYN\_FLOOD\_PREVENTION\_STAT\_H\_

#include "tcp\_syn\_flood\_prevention\_stat.h"

#endif

typedef struct \_TcpSynFloodPreventionStatTreeNodeData

{

// the node in which data is stored

ubi\_trNode Node;

// Fields to identify from what client the packet has came

u\_int8\_t ttl;

struct in\_addr ipSrc;

// the number of packets with TTL=ttl and IPSrc=ipSrc that've been processed

long counter;

} TcpSynFloodPreventionStatTreeNodeData;

/\*\*\* TcpSynFloodPreventionStatTreeNodeData manipulation functions \*\*\*/

static int TcpSynFloodPreventionStatTreeNodeDataCompareFunc(ubi\_trItemPtr ItemPtr, ubi\_trNodePtr NodePtr);

static void TcpSynFloodPreventionStatTreeNodeDataDeleteNode(ubi\_btNodePtr NodePtr);

void\* TcpSynFloodPreventionStatCreateModule()

{

TcpSynFloodPreventionModule\* newModule = (TcpSynFloodPreventionModule\* )SnortAlloc(sizeof(TcpSynFloodPreventionModule));

newModule->totalPacketsCount = 0l;

int\* a = (int\*)SnortAlloc(10);

newModule->rootStat = (ubi\_btRootPtr)SnortAlloc(sizeof(ubi\_btRoot));

ubi\_trInitTree(newModule->rootStat,/\* ptr to the tree head \*/

TcpSynFloodPreventionStatTreeNodeDataCompareFunc, /\* comparison function \*/

0); /\* do not allow nither OVERWRITE nor DUPLICATES \*/

return newModule;

}

void TcpSynFloodPreventionStatDeinitModule(TcpSynFloodPreventionModule\* preventionModule)

{

// kill tree

ubi\_trKillTree(preventionModule->rootStat, TcpSynFloodPreventionStatTreeNodeDataDeleteNode);

free(preventionModule->rootStat);

free(preventionModule);

}

int TcpSynFloodPreventionStatProcessPacket(TcpSynFloodPreventionModule\* module, Packet\* packet, int changeStat)

{

// try to find

TcpSynFloodPreventionStatTreeNodeData\* findNodeData = (TcpSynFloodPreventionStatTreeNodeData\*)SnortAlloc(sizeof(TcpSynFloodPreventionStatTreeNodeData));

findNodeData->ipSrc = packet->iph->ip\_src;

findNodeData->ttl = packet->iph->ip\_ttl;

TcpSynFloodPreventionStatTreeNodeData\* currNodeData = (TcpSynFloodPreventionStatTreeNodeData\* )ubi\_trFind(module->rootStat, findNodeData);

// update statistics

if(changeStat == CHANGE\_STAT\_YES)

{

if(currNodeData == NULL)

{

// add new node to the tree

TcpSynFloodPreventionStatTreeNodeData\* newNodeData = (TcpSynFloodPreventionStatTreeNodeData\*)SnortAlloc(sizeof(TcpSynFloodPreventionStatTreeNodeData));

newNodeData->ipSrc = findNodeData->ipSrc;

newNodeData->ttl = findNodeData->ttl;

ubi\_trNodePtr newNodePtr = &newNodeData->Node;

ubi\_trInsert(module->rootStat, newNodePtr, (ubi\_trItemPtr)newNodeData, NULL);

currNodeData = newNodeData;

}

module->totalPacketsCount++;

currNodeData->counter++;

printf("stats is updated %d \n", currNodeData->counter);

}

free(findNodeData);

// Make the decision if the packet is bad

if(currNodeData == NULL) return PREVENTION\_PACKET\_IS\_BAD;

double avg = 0;

double nodesCount = ubi\_trCount(module->rootStat);

if(nodesCount != 0)

{

avg = module->totalPacketsCount / nodesCount;

}

if(currNodeData->counter >= avg)

{

printf("packet is OK\n");

return PREVENTION\_PACKET\_IS\_OK;

}

else

{

printf("packet is BAD\n");

return PREVENTION\_PACKET\_IS\_BAD;

}

}

/\* Returns -1 if A < B

Returns 1 if A > B

Returns 0 if A = B \*/

static int TcpSynFloodPreventionStatTreeNodeDataCompareFunc(ubi\_trItemPtr ItemPtr, ubi\_trNodePtr NodePtr)

{

TcpSynFloodPreventionStatTreeNodeData \*A = (TcpSynFloodPreventionStatTreeNodeData \*) ItemPtr;

TcpSynFloodPreventionStatTreeNodeData \*B = (TcpSynFloodPreventionStatTreeNodeData \*) NodePtr;

if((A->ipSrc.s\_addr == B->ipSrc.s\_addr) && (A->ttl == B->ttl))

return 0;

else

{

if(A->ipSrc.s\_addr < B->ipSrc.s\_addr)

return -1;

else if(A->ipSrc.s\_addr > B->ipSrc.s\_addr)

return 1;

else

return (A->ttl < B->ttl ) ? -1 : 1;

}

}

static void TcpSynFloodPreventionStatTreeNodeDataDeleteNode(ubi\_btNodePtr NodePtr){

free(NodePtr);

}

Приложение Б

Исходный код вспомогательной утилиты

Утилита предназначена для:

* Извлечения из html страницы списка пингуемых хостов
* Извлечение из логов пингования времени отклика
* Анализ распределения полученных извлеченных значений времени

namespace pings{

class Class1{

public static void ExtractUrls(string FileName){

StreamReader sr = new StreamReader(FileName);

StreamWriter sw = new StreamWriter("run\_pings.cmd");

string content = sr.ReadToEnd();

string pattern = @"href=.\*""";

System.Text.RegularExpressions.MatchCollection matches = Regex.Matches(content, pattern );

foreach(Match match in matches){

string val = match.Value;

if(val.IndexOf("viacom.local") > -1) continue;

val = val.Replace("href=", "");

val = val.Replace("http://", "");

val = val.Replace(@"""", "");

val = val.Replace("/", "");

sw.WriteLine("ping " + Regex.Split(val, ":")[0]);

}

sr.Close();

sw.Close();

}

public static void ExtractPingTime(string FileName){

StreamReader sr = new StreamReader(FileName);

StreamWriter sw = new StreamWriter("extracted\_time.txt");

string str;

int count = 0;

while((str = sr.ReadLine()) != null){

if(str != string.Empty){

string [] tokens = str.Split();

foreach(string token in tokens){

if(token.IndexOf("time=") > -1){

count ++;

sw.WriteLine(token.Replace("time=", "").Replace("ms", ""));

}

}

}

}

sr.Close();

sw.Close();

}

public static void Usage(){

Console.WriteLine("pings <option> <filename>");

Console.WriteLine(" option={url, time}");

}

[STAThread]

static void Main(string[] args){

if(args.Length == 2){

switch(args[0]){

case "url":

ExtractUrls(args[1]);

break;

case "time":

ExtractPingTime(args[1]);

break;

}

}

else{

Usage();

}

}

}

}