МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
"САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ имени академика С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)"

ВЫЯВЛЕНИЕ АНОМАЛЬНЫХ СЕТЕВЫХ СОСТОЯНИЙ И ПРИЧИН ИХ ВОЗНИКНОВЕНИЯ

Рекомендовано редакционно-издательской комиссией по фундаментальным наукам в качестве учебного пособия

САМАРА 2017

УДК СГАУ: 004.9
ББК СГАУ: 32.973-18
И741

Авторы: А.А. Гальцев, Е.С. Сагатов, А.М. Сухов

Рецензенты: д-р техн. наук, проф. С.Б. Попов
д-р физ.-мат. наук, проф. А.Ю. Привалов

В741 **Выявление аномальных сетевых состояний и причин их возникновения**: учеб. пособие / А.А. Гальцев [и др.] / – Самара: Изд-во Самар. гос. аэрокосм. ун-та, 2017 – 112 с. : ил.

Учебное пособие посвящено изложению принципов безопасной эксплуатации компьютерных сетей, причем рассматриваются как теоретические основы, так и их программная реализация. В частности, излагается классификация сетевых состояний на основе потокового подхода, выводятся критерии аномальных состояний для применения их в алгоритмах программ, предназначенных для обнаружения и противостояния сетевым вторжениям.

Учебное пособие ориентировано на подготовку магистров, на его основе может читаться отдельный курс. Технологии, которые рассматриваются в рамках данного учебного пособия, были разработаны как результат исследований, проведенных авторами. Это пособие может быть полезно также для других категорий студентов, которым необходимы знания по современным сетям и основам сетевой безопасности. Учебное пособие подготовлено при финансовой поддержке Министерства образования и науки.

УДК СГАУ: 004.9
ББК СГАУ: 32.973-18

ВВЕДЕНИЕ

Курсы по современным компьютерным сетям и сетевой безопасности входят в учебные планы практически всех специальностей из группы информатика. В настоящее время имеются хорошие учебные пособия по данным дисциплинам, однако все эти учебные пособия излагают ставшие уже стандартными методы, никак не отражая текущую ситуацию, которая постоянно меняется. Появляются новые угрозы и инструменты их реализации. Поэтому, на уровне магистерской подготовки необходимы курсы, которые опираются на современные исследования и отражают современные подходы, а также учат студентов навыкам самостоятельных исследований и применения на практике их результатов.

Настоящее учебное пособие является попыткой изложить для студентов теоретические модели, описывающую состояние сети с использованием понятия потоков (flow), а также аудиторию пользователей интернет сервиса с выделением ее ядра. На основе этих моделей разработаны алгоритмы обнаружения аномальных состояний сети и источников, осуществляющих несанкционированное воздействие.

При чтении данного курса предполагается выполнение вместо лабораторных работ ряда проектных заданий на основе данных, собранных на реальных сетях. Поэтому данное пособие содержит листинг рядя программ для обработки данных, которые должны быть доработаны и запущены студентами.

Данный курс разработан на основе исследований, проведенных авторами учебного пособия в рамках государственного задания. Учебное пособие подготовлено при поддержке Министерства образования и науки.

1 МОДЕЛИРОВАНИЕ ИНТЕРНЕТ ТРАФИКА

1.1 Обзор существующих моделей трафика

Трафик — это объем данных или количество сообщений, переданных через канал за определенный промежуток времени. Трафик также включает отношение между попытками вызова оборудования, чувствительного к трафику, и скоростью выполнения этих вызовов.

Анализ трафика дает возможность определить необходимую ширину полосы пропускания каналов передачи данных и голосовых вызовов. Проектирование трафика направлено на решение проблем качества связи, так как дает возможность определить уровень обслуживания и коэффициент блокирования. Сеть, спроектированная надлежащим образом, имеет низкий коэффициент блокирования и высокий уровень использования канала, т.е. качество обслуживания повышается, а затраты уменьшаются.

В настоящее время существуют различные подходы к описанию трафика, при этом четкой классификации разрабатываемых моделей трафика не существует. В то же время можно выделить два основных направления, в которых идет развитие исследований моделей трафика.

Первое направление основано на описании моделей трафика в виде пакетов. Основная методика паспортизации пакетных сетей дается в рекомендации RFC-2544 [1], в которой определены следующие характеристики сети, используемые для оценки качества ее работы:

- уровень загрузки канала (utilization level);
- время доставки пакета (one way delay), которое в простейшем случае может быть оценено по данным команды ping;
- вариация в задержке пакета или джиттер (Latency Distribution);
- количество потерянных пакетов (packet loss rate).

Последние три характеристики служат в основном для оценки качества соединения между двумя удаленными точками (end-to-end). Характеристика уровня загрузки канала применяется для мониторинга канала между двумя соседними маршрутизаторами (hop).

Крупные зарубежные компании руководствуются своими собственными техническими условиями для оценки пиковых нагрузок магистральных соединений, по достижению которых требуется расширение каналов. Эти пиковые значения варьируются в пределах 35-60%. Так, международный телекоммуникационный гигант Спринт (Sprint) считает невозможным эксплуатацию каналов с пиковой загрузкой выше 50% [2] Подобные вопросы постоянно обсуждаются на ведущих конференциях, таких, как ACM SIGCOMM, IEEE Infocom и др., однако накопленные знания еще не достигли того уровня, когда появляется возможность выработать единый стандарт и соответствующее программное обеспечение для измерений.

К сожалению, пока не сформировалось единое мнение по оценке качества связи, получаемой на магистральной сети и определению узких мест. Правило 50-ти процентной средней загрузки сети также не всегда работает.

При моделировании трафика на уровне пакетов необходимо учитывать следующие параметры:

- Модели поступления вызовов
- Заблокированные вызовы
- Количество источников
- Время удержания

После определения моделей поступления вызовов и заблокированных вызовов, количества источников и времени удержания вызова можно выбирать модель трафика, которая наилучшим образом подходит для вашего окружения. Хотя никакая модель трафика не может точно соответствовать реальной ситуации, эти модели предполагают средние значения для каждой ситуации. Существует множество различных пакетных моделей трафика, и главным является выбор модели, наиболее подходящей существующему окружению.

Модели трафика на уровне пакетов, которые используются чаще всего, — это Эрланг В, Расширенный Эрланг В и Эрланг С. Другие распространенные модели трафика — Энгсет, Пуассон, EART/EARC и Нил-Уилкерсон.

Моделирование трафика на пакетном уровне − задача довольно сложная, так как канальный трафик есть результат сумми-

рования трафика огромного числа отдельных соединений. Обработка таких данных достаточно сложна, в первую очередь из-за их больших объемов, и требует высокопроизводительного оборудования, сложного программного обеспечения и квалифицированных программистов.

1.2 Понятие потока (flow)

Также существует альтернативный подход к описанию трафика основанный на представлении трафика в виде совокупности потоков (flow).

Большинство реализаций методов анализа трафика в виде потоков основаны на протоколе NetFlow. NetFlow – это сетевой протокол, разработанный компанией Cisco Systems для сбора информации об IP-трафике. NetFlow стал промышленным стандартом для мониторинга и биллинга трафика. Протокол поддерживается платформами, отличными от Cisco IOS (Internetwork Operating System) и NX-OS (Nexus Operating System).

Существует несколько определений сетевого потока, в классическом определении компании Cisco поток описывается как семипараметрический объект. То есть поток - это однонаправленная последовательность пакетов со следующими неизменными параметрами:

- IP-адрес источника;
- IP-адрес назначения
- Порт источника для протоколов UDP или TCP, 0 для других протоколов
- Порт назначения для протоколов UDP или TCP, тип и код для ICMP, или 0 для других протоколов
- IP протокол передачи
- Входящий интерфейс (SNMP ifIndex)
- Тип IP-услуг

Преимущество исследования поведения сетей на основе потоковых технологий перед пакетными заключается в том, что в методах, основанных на потоках, требуется обрабатывать только часть из всего объема данных. Было показано, что потоковые данные составляют только 1% от пакетных, при этом не теряя ценности. Поэтому такие подходы оказываются более производительными и

масштабируемыми, требуя при этом намного меньших затрат, в результате чего являются наиболее перспективным направлением в изучении трафика.

Многочисленные авторы [3] проанализировали интернет-трафик и показали, что он ведет себя как самоподобный процесс. Это открытие сделало революционный шаг, положив начало отхода от марковских моделей.

Другие работы (например, [4], [5]) посвящены исследованию вопросов моделирования интернет-трафика на потоковом уровне. Основной целью является показать, каким образом пропускная способность сети является общей для различных потоков, или, что равнозначно, для вычисления длительности потоков. Теория очередей процессоров с разделением во времени используются для моделирования перегруженных каналов сети. В работе [4], предложена модель M/G/∞ количество активных потоков для неперегруженных участков сети. Оно совпадает с частным случаем нашей модели, где все потоки имеют одинаковую скорость. В работе [5], модель очереди в многопроцессорной системе с разделением во времени используется для вычисления длины очереди и вероятности потери пакетов в буфере управления активной очередью, пересекаемом потоками TCP разных размеров. В результате было получено среднее значение длительности потока TCP. Заметим, что во всех вышеназванных моделях на основе потоков делается предположение, что потоки прибывают в соответствии с однородным процессом Пуассона.

Чади Баракат и др. [5] предложили модель, которая позволяет на основе информации о потоках построить модель суммарного трафика на магистральном участке. В процессе моделирования предполагается, что наблюдаемый трафик представляет собой суперпозицию большого количества потоков, которые прибывают случайным образом и остаются активными в течение случайного периода времени.

1.3 Модель Бараката

Математическая модель Интернет-трафика на уровне потоков базируется на теории стохастических сетей. Основы этой модели были представлены на конференции ACM SIGCOMM в августе 2001 года Беном Фреди и др. [4]. На основе представленной модели

Чади Баракат и др. [5] исследовали, в частности, трафик магистральных каналов сети Sprint OC-12 (622 Мбит/с) и усовершенствовали модель. Предложенная модель трафика для неперегруженных участков магистральной сети достаточно проста, и ее можно использовать в управлении сетью. Здесь мы кратко воспроизведем все важнейшие результаты, полученные в этих работах.

Модель Чади Бараката опирается на пуассоновскую модель белого шума [5]. С помощью трех параметров

- λ – интенсивность входного числа потоков (число новых потоков в единицу времени)
- $\mathbb{E}[S_n]$ – средний размер потока в битах
- $\mathbb{E}[S_n^2/D_n]$ – среднее значение для отношения квадрата размера потока к его длительности

можно выразить среднее значение для скорости передачи данных в канале $\mathbb{E}[B(t)]$

$$\mathbb{E}[B(t)] = \lambda \mathbb{E}[S_n] \qquad (1.1)$$

и его вариацию на моделируемом канале $\mathbb{V}[B(t)]$

$$\mathbb{V}[B(t)] = \lambda \mathbb{E}\left[\frac{S_n^2}{D_n}\right] \qquad (1.2)$$

Следует отметить, что уравнение (1.1) справедливо только для идеального случая, когда исследуемый участок сети имеет неограниченную пропускную способность. Это уравнение можно применять только к слабо загруженным участкам. Основной недостаток этого уравнения заключается в отсутствии четко определенной области применения, что объясняется тем фактом, что переменные λ и $E[S_n]$ никак не связаны с текущим состоянием сети. Средний размер потока $E[S_n]$ не зависит от конкретной сети, а является универсальной величиной и определяется эмпирически, в результате исследования свойств глобальной сети.

Постоянная прибытия потоков λ (величина обратная к среднему значению промежутка времени между двумя последовательными интервалами) описывает поведение пользователей сети и не зависит от состояния сети или ее загрузки. Суммарное число потоков (запросов от пользователей), которые прибывают на данный участок сети, остается линейно зависящим от времени наблюдения, даже если сеть начинает испытывать проблемы и не может

удовлетворить все запросы пользователей. Данная модель опирается на следующие основные предпосылки:

1. Появление новых потоков на исследуемом участке магистральной сети описывается однородным Пуассоновским процессом. Измерения, проведенные в работе Бараката и др. говорят о том, что λ остается постоянной в течении, как минимум, 30-ти минутного интервала. В общем случае, приведенные соотношения могут воспроизводится, если процесс прибытия потоков описывается более общими типами процессов, такими как Марковский процесс или неоднородный процесс Пуассона.

Обозначим T_nкак время прибытия *n*-го потока, S_n− его размер в байтах и D_n− его длительность в секундах. Поток считается активным в период времени $T_n \le t \le T_n + D_n$. Определим $X_n(t - T_n)$, как скорость *n*-го потока в *бит/с* в момент времени t, причем X_n равно нулю для $t < T_n$ и $t > (T_n + D_n)$. Другими словами, $X_n(t - T_n)$ равно нулю, если поток *n* неактивен в момент времени t. X_n зависит от S_nи D_nи от динамики, регулирующей скорость потока. Например, для потоков TCP, динамика скорости потока есть функция изменения размера окна, которая в свою очередь является функцией от двусторонней задержки (RTT - Round Trip Time - время между отправкой запроса и получением ответа) и потери пакетов. Отметим, что:

$$\int_0^{D_n} X_n(u)du = S_n \qquad (1.3)$$

2. Последовательности $\{S_n\}$ и $\{D_n\}$ независимы друг от друга и идентично распределены. Утверждение о независимости функций скорости потоков основаны на следующих фактах:

- под каналом связи понимается магистральный канал, не перегруженный инженерными настройками. Поэтому это не результат накопленного опыта, а также это не говорит о зависимости между функциями скорости потоков;
- потоки, проходящие по этому каналу, имеют большое число различных источников и получателей, и используют большое число различных маршрутов движения перед тем, как быть объединенными на магистральном канале. Пря-

мым следствием предположения 2 является то, что последовательности $\{S_n\}$ и $\{D_n\}$ также образуют независимые и идентично распределенные последовательности, хотя для больших n, S_nи D_n явно коррелируют: чем больше S_n, тем больше D_n (в целом). Наконец, мы считаем, что $\mathbb{E}[D_n]$ конечно.

Точный тип распределения зависит от типов рассматриваемых участков сети, однако представляется разумным, что его форма является типичным распределением Парето с тяжелым хвостом (heavy tailed):

$$\Pr(\text{size} \leq x) = 1 - \frac{k}{x^{\beta}}, \text{ для } x \geq k \qquad (1.4)$$

с $1 \leq \beta \leq 2$такое распределение имеет конечное среднее и бесконечно большую вариацию.

Обозначим через $B(t)$ величину траффика (в *bit/s*), обслуживаемого на моделируемом канале в момент времени t. Данный трафик можно рассматривать как совокупность различных потоков его составляющих

$$B(t) = \sum_{n \in \mathbb{Z}} X_n(t - T_n) \qquad (1.5)$$

Сетевое состояние из уравнения (1) может быть также описано числом активных потоков N в момент времени t с учетом очереди $M/G/\infty$ [11a], если $X_n(t - T_n) = 1$ для $t \in [T_n, T_n + D_n]$, где T_n - время начала потока, а D_n его длительность.

Далее, необходимо найти моменты процесса $B(t)$в стационарном режиме. Предполагается, что стационарный режим существует для конечных λ и $\mathbb{E}[D_n]$. Приведем результат для преобразований Лапласа Стилтьеса (ПЛС) функции $B(t)$, что позволяет вычислить все моменты $B(t)$, а также ее распределение первого порядка. Для частных случаев функции скорости потока, представленных на рисунке 1.1, видно, что всего с тремя параметрами (λ, $\mathbb{E}[S_n]$, $\mathbb{E}[S_n^2/D_n]$) с помощью данной модели можно вычислитье средние значения парметров магистрального трафика и их вариацию.

Положим, что $\tilde{B}(\omega) = E[e^{-\omega B(t)}]$ - изображение функции $B(t)$, где $\tilde{B}(\omega) \geq 0$. Пусть *N(t)* — число активных потоков в момент времени *t*. Предположения 1 и 2 подразумевают, что скорость

передачи данных *B(t)* в момент времени *t* — это сумма произвольного числа *N(t)* независимых и идентично распределенных случайных переменных, являющихся скоростями активных потоков. Тогда изображение функции $B(t)$ будет равно:

$$\tilde{B}(\omega) = \exp\left(\lambda \mathbb{E}\left[\int_0^{D_n} e^{-\omega X_n(u)}\,\mathrm{d}u\right] - \lambda E[D_n]\right) \quad (1.6)$$

Дифференцируя данное выражение по ω, и полагая затем ω равной нулю, можно получить все моменты суммарной скорости потоков в стационарном режиме. Первые два момента будут следующие: среднее значение суммарной скорости $\mathbb{E}[B(t)] = \lambda \mathbb{E}[S_n]$, а вариация среднего значения $V_B = \lambda \mathbb{E}[\int_0^{D_n} X_n^2(u)\mathrm{d}u]$.

Среднее значение суммарной скорости потоков и его вариация являются двумя важными показателями производительности и качества предоставляемых услуг провайдеров Интернет. В отличие от других моделей трафика, данная модель говорит о том, что вариация суммарной скорости потоков есть функция длительности потоков и функций их скоростей. Это требует некоторых предположений (или более детальной информации) о поведении скорости потоков. Далее приведем аппроксимации вариации $B(t)$ для некоторых частных случаев функций скорости потоков. На рисунке 1.1 показаны различные варианты форм функции X_n

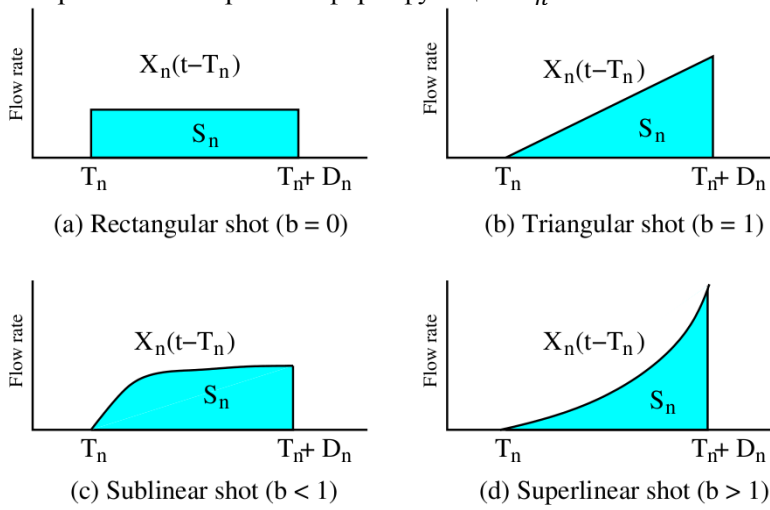

Рисунок 1.1. Модели функции скорости потоков

Рассмотрим два частных случая, показанных на рисунке 1.1a и 1.1b.

1) Прямоугольный вид: в данном случае скорость потока постоянна и равна S_n/D_n (на рисунке 1.1a получается прямоугольник длиной D_n и шириной S_n/D_n). Тогда вариация $B(t)$ равна $V_B = \lambda\mathbb{E}[S_n^2/D_n]$. Данное предположение является самым простым. Им мы охватываем только вариацию суммарной скорости, вызванной вариацией $N(t)$ и вариацией скорости S_n/D_n. Несложно показать, что среди всех возможных форм функции скорости потоков при прямоугольной форме достигается наименьшая вариация V_B.

2) Треугольный вид: скорость потока линейно возрастает со временем (Рисунок 1.1b). Данное предположение взято из динамики передачи данных TCP-соединений, из которых в большинстве случаев и состоят потоки в магистральных каналах. Далее будет показано, что скорость потоков треугольной формы действительно описывает TCP-потоки при определенных условиях. Для размера потока S_nи длительности D_n, предполагается, что скорость увеличивается линейно от нуля до $2S_n/D_n$, со средним значением, равным S_n/D_n. В момент времени t между T_nи $T_n + D_n$, можно сказать, что $X_n(t - T_n) = (2S_n/D_n^2)(t - T_n)$. Тогда вариация *B(t)* равна $V_B = \frac{4}{3}\lambda\mathbb{E}[S_n^2/D_n]$. Как и в первом случае, получается вариация, кратная $\mathbb{E}[S_n^2/D_n]$. Как ожидалось, варация получилась больше, чем в случае с прямоугольной формой.

Средний размер потока $\mathbb{E}[S_n]$ характеризует глобальную сеть в целом, он не меняется в зависимости от конкретного канала, так же, как и интенсивность входящего числа потоков λ характеризует поведение пользователей сети, а не конкретную сетевую инфраструктуру. В представленной модели переменные λ и $\mathbb{E}[S_n]$ никак не связаны с текущим состоянием сети. Средний размер потока $\mathbb{E}[S_n]$ не зависит от конкретной сети, а является универсальной величиной, характеризующей свойства глобальной сети. Постоянная прибытия потоков λ (величина обратная к среднему значению промежутка времени между двумя последовательными интервалами) описывает поведение пользователей сети и не зависит от состояния сети или ее загрузки. Суммарное число потоков (запросов от пользователей), которые прибывают на данный участок сети, остается линейно зависящим от времени наблюдения, даже если в сети

начинаются проблемы, и она не может удовлетворить все запросы пользователей.

В заключение данного раздела приведем еще один закон, который описывает состояние сети. Это закон Литтла

$$N = \lambda \mathbb{E}[D_n], \qquad (1.7)$$

который связывает число активных потоков и среднюю длительность потока.

Следует отметить, что этот закон будет выполняться для любых, в том числе и сильно загруженных каналов связи, с произвольным типом распределения размера потока. Разница в подходах, рассмотренных в уравнениях (1.1) и (1.7), состоит в том, что для описания сети применяются средний размер и длительность потока. В отличии от среднего размера потока $\mathbb{E}[S_n]$, средняя длительность потока $\mathbb{E}[D_n]$ может изменяться и зависит от текущего состояния сети.

1.4 Область эксплуатации сети

В данном разделе хотелось бы представить метод диагностики магистральных каналов связи и его апробацию на действующей сети. Для того чтобы проанализировать качество связи на магистральном канале недостаточно одной переменной, характеризующей загрузку канала $B(t)$ из уравнения (1.1). Ибо одну и ту же загрузку может генерировать различное количество пользователей и их сессий связи.

Представленный метод основывается на модели трафика [6], согласно которой число активных потоков может рассматриваться в качестве важней характеристики реального сетевого состояния. Две переменные, число активных потоков N совместно с утилизацией канала U, наиболее полно описывают текущее сетевое состояние, см. уравнения (1.1) и (1.7). Анализ этих уравнений показывает, что зависимость утилизации U от числа активных потоков N на уровне потоков соответствует зависимости размера потока $\mathbb{E}[S_n]$ от его длительности $\mathbb{E}[D_n]$. Так как размер потока меняться не может, то изменяется только его длительность. И именно эту зависимость можно достаточно просто построить и проанализировать.

Данные о сетевом состоянии представлены отдельными точками на плоскости с осями, на которых отложены число активных

потоков N и утилизация сети U. Подобный подход позволяет выделить на графике 1.2 три участка, соответствующие качественно различным состояниям сети.

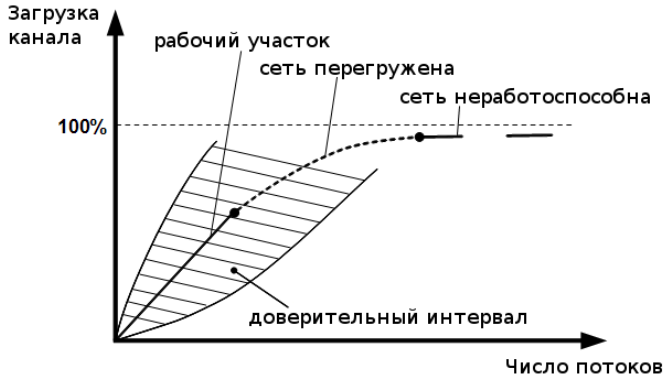

Рисунок 1.2. Сетевые состояния в области эксплуатации сети

При накоплении статистических данных можно перейти от отдельных точек, характеризующих текущее сетевое состояние к кривой, составленной из усредненных значений. Эта кривая будет описывать различные сетевые состояния.

Первая часть кривой, составленной из усредненных значений данных, образует прямую линию, которая заканчивается точкой перегиба. Она соответствует рабочему участку сети и характеризуется отсутствием потерь IP-пакетов. В этой области поведение сети близко к идеальному, при росте размера потока $\mathbb{E}[S_n]$ прямо пропорционально увеличивается время его передачи $\mathbb{E}[D_n]$. При этом угол наклона прямой с графика 1.2 представляет собой среднюю скорость потока b. Сетевые администраторы должны стремиться, чтобы их сеть всегда эксплуатировалась в пределах рабочего участка.

При эксплуатации сети выход за пределы рабочего участка крайне нежелателен, так как он сопровождается ухудшением качества связи. Конец рабочего участка определяется точкой перегиба, которая может быть найдена экспериментально. Положение этой точки зависит от множества факторов, таких как протокол транспортного уровня (ATM, SDH, Ethernet и т.д.), топология сети, размер буфера маршрутизатора и т.д.

Вторая, изогнутая часть кривой соответствует перегруженной сети и характеризуется возможными потерями пакетов до 1 %, что приводит к снижению эффективного размера передаваемого сегмента TCP/IP. Сеть уже не справляется с предложенной нагрузкой, что приводит к снижению средней скорости потока b.

Следует также упомянуть и о точке перегрузки сети, которая совпадает с концом области определения для уравнения (1.7). После этой точки в сети начинаются необратимые явления, приводящие к значительным потерям пакетов – свыше 1%.

Третий, почти горизонтальный, участок кривой соответствует полностью неработоспособной сети со значительной потерей пакетов свыше 1%. Потери пакетов для этого сетевого состояния неизбежны. Сетевые администраторы должны избегать эксплуатации сети на данном участке.

1.5 Доверительный интервал

Напомним еще раз, что конкретные сетевые состояния представляют собой точки на графике с рисунка 1.2. Для того, чтобы построить кривую усредненных значений требуется достаточно большой период времени для сбора статистики, а делать выводы о аномальном сетевом состоянии необходимо достаточно быстро. Для этого можно ввести понятие доверительного интервала для рабочего участка сети.

Поскольку суммарная нагрузка исследуемого канала $B(t)$ есть результат мультиплексирования большого количества активных потоков данных $N(t)$, независимых друг от друга, распределение суммарной нагрузки стремится к нормальному (Гауссову) распределению.

Уравнение (1.2) дает нам выражение для вариации скорости передачи данных в канале, следовательно, отдельные точки, характеризующие текущие сетевые состояния, должны распределяться внутри интервала $\left[\mathbb{E}[B] - A(\epsilon)\sqrt{\mathbb{V}[B]}, \mathbb{E}[B] + A(\epsilon)\sqrt{\mathbb{V}[B]}\right]$ с вероятностью $p = 1 - \epsilon$. Здесь $A(\epsilon)$ нормальная квантильная функция (probit).

С учетом уравнений (1.1) и (1.7) и теорем о среднем можно получить следующее выражение для доверительного интервала с учетом того, что величины D_n и S_n распределены независимо:

$$B = b\left(N \pm \alpha A(\epsilon)\sqrt{N}\right) \quad (1.8)$$

где $A(\epsilon)$ - нормальная квантильная функция, $b = \mathbb{E}[S_n/D_n]$ это средняя скорость потока, α – нормировочная константа, определяемая экспериментально. Уравнение (1.8) говорит о том, что реальное состояние сети, описываемое числом активных потоков N и скоростью потока данных B, будет находиться вне указанных пределов только в ϵ доле от общего времени наблюдения. Доверительный интервал ограничивает двумя параболами область нормальных сетевых состояний.

В случае, если исследуемая сеть получает значительные помехи, то результатом являются отклонения от области нормальных состояний. Такие отклонения должны наблюдаться при отключении одного или нескольких внешних или внутренних каналов, DDoS атаки или сканировании портов и т.д. В этом случае новые состояния сети должны покидать доверительный интервал из уравнения (1.8).

Представленная модель трафика позволяет также сформулировать простой критерий для поиска аномальных состояний сети: если несколько последовательных измерений выйдут за пределы доверительного интервала с ϵ =0.05, то можно уверенно говорить о проблемах на сети. Если снимать данные каждые несколько минут, то статистика за несколько часов даст возможность определить с достаточной точностью все параметры уравнения (1.8).

Основными признаками аномальных состояний являются:

1. Большое число потоков с одного IP-адреса. В данном случае есть большая вероятность осуществления сетевой атаки от источника данных с подобным IP-адресом.
2. Большее число потоков при неизменной загрузке канала.
3. Большая загрузка канала при неизменном числе потоков. Возможно осуществление сетевой атаки типа DDoS.
4. Общее уменьшение загрузки канала при неизменном количестве активных потоков — возможно отказа одного из узлов сети.

Все данные сетевые аномалии позволяет обнаружить предлагаемая модель трафика на уровне потоков. По сравнению с анализом трафика только на основе утилизации канала данная модель дает возможность определить более детально тип аномалии. Также она имеет преимущество перед пакетными моделями, поскольку

требуется намного меньше вычислительных ресурсов для обработки данных на уровне потоков.

Для успешного обнаружения сетевой аномалии в предлагаемой модели трафика необходимо сформулировать определение оптимального периода сбора данных о состоянии сети. В зависимости от выбранного интервала сбора данных будут зависеть параметры распределения утилизации канала. Основными из них являются среднее значение и дисперсия. Среднее значение будет примерно оставаться на том же уровне. Дисперсия же будет меняться обратно пропорционально периоду измерения: чем меньше период измерения, тем большее значение будет иметь дисперсия. Поэтому при анализе данных с разными периодами измерения предполагается, что получится нормальные распределения с примерно одинаковым средним, но различными дисперсиями.

При выборе интервала измерения также важным моментом является то, насколько оперативно необходимо обнаружить сетевую аномалию. Понятно, что наименьшим временем обнаружения будет являться выбранный период измерения. В случае, если вероятность α достаточно высока (несколько %), то выход за пределы доверительного интервала будет происходить достаточно часто. Вероятность повторного последовательного выхода из доверительного интервала будет уже намного меньше, порядка ϵ^2. Это позволяет сделать вывод, что если два последовательных измерения выйдут за пределы доверительного интервала, то в сети возникла 1аномалия.

Таким образом, имеется несколько критериев выбора интервала сбора данных:

1. Оперативность обнаружения аномального состояние: чем меньше период измерения, тем лучше. Оперативность зависит от периода измерения и количества последовательных выходов за пределы доверительного интервала, после которых будет считаться, что аномальное состояние обнаружено.
2. Вероятность ложного срабатывания системы: также необходимо минимизировать. Данный критерий будет зависеть от двух параметров: вероятности выхода за пределы доверительного интервала и количества последовательных выходов за пределы доверительного интервала, после которых будет считаться, что аномальное состояние обнаружено.

1.5 Сетевая инфраструктура

Для того чтобы выяснить детали несанкционированного вторжения в разные годы, начиная с 2007, была проведена серия экспериментов, имитирующих попытки атак. Они проводились на сетях российской научно-образовательной сети FREEnet, ирландской национально-образовательной сети HEAnet, самарского регионального коммерческого провайдера СамараТелеком и Самарского национального исследовательского университета. Каждая из этих сетей имела многочисленные внешние и внутренние каналы. В начале экспериментов пропускная способность магистральных каналов составляла 155 Мбит/с и 622 Мбит/с, с течением времени эта величина росла.

Следует отметить, что поддержка NetFlow должна быть включена на всех задействуемых интерфейсах, иначе результаты измерений будут некорректными.

Для сбора информации о трафике по протоколу NetFlow требуются следующие компоненты:

- **Сенсор.** Собирает статистику по проходящему через него трафику. Обычно это L3-коммутатор или маршрутизатор, хотя можно использовать и отдельно стоящие сенсоры, получающие данные путем зеркалирования порта коммутатора. Сенсор слушает сеть и фиксирует данные сеанса. Также как Snort или любая другая система обнаружения вторжений, сенсор должен иметь возможность подключиться к хабу, "зеркалированному" порту коммутатора или любому другому устройству, для просмотра сетевого трафика. Если вы используете систему пакетной фильтрации на базе BSD или Linux, то это превосходное место для сенсора Netflow, так как весь трафик будет проходить через эту точку. Сенсор будет собирать информацию о сеансах и сбрасывать ее в коллектор.
- **Коллектор.** Собирает получаемые от сенсора данные и помещает их в хранилище. Коллектор прослушивает указанный порт UDP и осуществляет сбор информации от сенсора. Полученные данные он сбрасывает в файл для дальнейшей обработки. Различные коллекторы сохраняют данные в различных форматах.

- **Система обработки и визуализации (анализатор).** Система обработки анализирует собранные коллектором данные и генерирует отчеты в форме, более удобной для человека. Эта система должна быть совместима с форматом данных, предоставляемых коллектором (часто в виде графиков).

В качестве каждого из элементов системы может использоваться несколько разных вариантов программ. Общая структурная схема системы анализа сетевого трафика с помощью протокола NetFlow представлена на рисунке 1.3.

NetFlow использует UDP или SCTP для передачи данных о трафике коллектору. Как правило, коллектор слушает порт 2055, 9555 или 9995.

Сенсор выделяет из проходящего трафика потоки, характеризуемые следующими параметрами:

- Адрес источника;
- Адрес назначения;
- Порт источника для UDP и TCP;
- Порт назначения для UDP и TCP;
- Тип и код сообщения для ICMP;
- Номер протокола IP;
- Сетевой интерфейс (параметр ifindex SNMP);
- Тип сервиса IP.

Потоком считается набор пакетов, проходящих в одном направлении. Когда сенсор определяет, что поток закончился (по изменению параметров пакетов, либо по сбросу TCP-сессии), он отправляет информацию в коллектор. В зависимости от настроек он также может периодически отправлять в коллектор информацию о все еще идущих потоках.

Собранная информация отправляется в виде записей, содержащих следующие параметры (для версии 5):

- Номер версии протокола;
- Номер записи;
- Входящий и исходящий сетевой интерфейс;
- Время начала и конца потока;
- Количество байт и пакетов в потоке;
- Адрес источника и назначения;
- Порт источника и назначения;

- Номер протокола IP;
- Значение Type of Service;
- Для TCP-соединений — все наблюдаемые в течение соединения флаги;
- Адрес шлюза;
- Маски подсети источника и назначения.

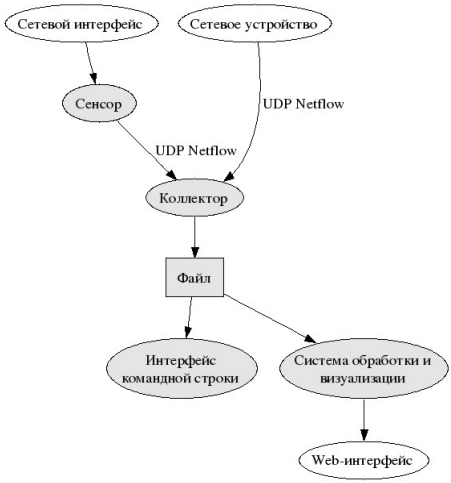

Рисунок 1.3 - Структурная схема системы на базе Netflow

Версия 9 также поддерживает дополнительные поля, такие как заголовки IPv6, метки потоков MPLS и адрес шлюза BGP. Некоторые сенсоры могут также поддерживать номер автономной системы.

Если используется UDP, то потерянная из-за проблем сетью запись не будет получена коллектором. Коллектор может определить потери пакетов по значениям номера записи, которые по стандарту должны быть возрастающими.

Если сенсором выступает сетевое устройство (маршрутизатор или коммутатор), то для экономии ресурсов NetFlow включают только для тех интерфейсов, на которых хотят собирать статистику.

Для экономии ресурсов процессора также применяется «sampled NetFlow». В этом случае сенсор анализирует не все, а каждый *n*-ый пакет, где *n* может быть заданным административно или выбираемым случайным образом. При использовании sampled

NetFlow получаемые значения являются не точными, а оценочными.

Как правило, сеть использует один или несколько внешних каналов, при этом внутренние присоединения остаются гораздо менее загруженными. Поэтому можно строить зависимость числа активных потоков на граничном маршрутизаторе от общей загрузки внешних каналов. Это достигается с помощью следующих команд:

- sh ip cache flow − дает информацию о количестве активных и неактивных потоков, об их параметрах в конкретный момент времени;
- sh int [названия внешних интерфейсов] − выдает информацию о текущей загрузке канала.

Данные, полученные в результате выполнения этих команд, содержат все необходимые значения для построения графика, аналогичного изображенному на рисунке 1.2. Их следует снимать круглосуточно, с интервалом в 30 (5) мин в течение недели, чтобы выявить поведение сети при различной загрузке.

Для GSR маршрутизатора с гигабитной матрицей коммутации снятие данных о количестве активных потоков достигается с помощью следующих команд:

- enable;
- attach [номер слота];
- show ip cache flow.

Эксперимент включает в себя следующие этапы:

1. Настройка сенсора NetFlow на граничном маршрутизаторе сети
2. Настройка NetFlow-коллектора, в который сенсоры будут отправлять информацию о потоковом трафике
3. Сбор данных с граничного маршрутизатора
4. Анализ полученных данных

Схема эксперимента представлена на рисунке 1.4.

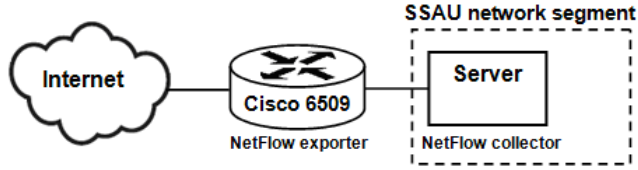

Рисунок 1.4 — Схема эксперимента

В результате проведения эксперимента мы получаем набор данных, включающих в себя два основных параметра: нагрузку канала и количество активных потоков в определенный момент времени.

Эксперименты были проведены на маршрутизаторах, обслуживающих внешние каналы крупных сетей – московской научно-образовательной сети FREEnet, ирландской научно-образовательной сети HEAnet, а также ЗАО «СамараТелеком» в Самаре. Каждый из исследуемых маршрутизаторов обслуживал несколько каналов – внешних и внутренних. Скорости данных подключений составляли 1 Гбит/с для FREEnet, 622 и 155 Мбит/с для HEAnet и 8 Мбит/с для СамараТелеком. Загрузка каналов варьировалась от 5% до 60%.

В Самаре измерения проводились на маршрутизаторе Cisco 7206, в сети HEAnet – на маршрутизаторе Cisco 12008. Следует отметить, что поддержка NetFlow должна быть включена на всех задействуемых интерфейсах, иначе результаты измерений будут некорректными. Так, наша первая попытка провести эксперимент на Ирландской научно-образовательной сети HEAnet оказалась неудачной из-за неполной настройки технологии NetFlow на маршрутизаторах.

В СамараТелеком на основе NetFlow построена биллинговая система, поэтому получение данных ограничено текущими настройками программного обеспечения. По сравнению с коммерческими, научно-образовательные сети не требуют очень подробной детализации трафика, связанной с тарифными планами и оплатой, в них имеется большая свобода в выборе настроек. Но в то же время обслуживающий персонал не обладает достаточными навыками по детализации трафика, а закупленное программное обеспечение часто не позволяет делать подобные точные измерения.

Как правило, сеть использует один или несколько внешних каналов, при этом внутренние присоединения остаются гораздо менее загруженными. Поэтому можно строить зависимость числа активных потоков на граничном маршрутизаторе от общей загрузки внешних каналов.

1.6 Результаты экспериментов

На рисунке 1.5 приведены данные о зависимости нагрузки канала от числа активных потоков для сети СамараТелеком. Эти данные снимались вручную в течение нескольких дней.

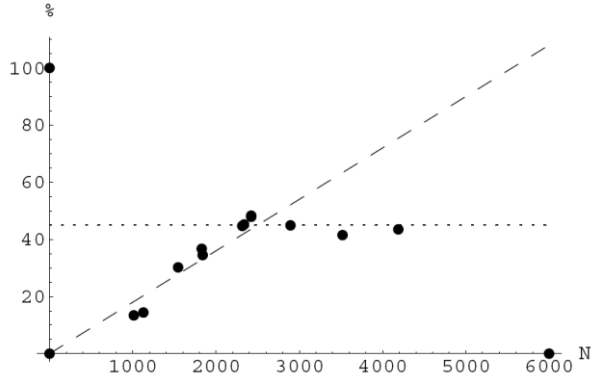

Рисунок 1.5 − Зависимость загрузки внешнего канала от числа активных потоков (СамараТелеком), 2004 год

Следующие данные (рисунок 1.6) относятся к внешнему каналу HEAnet (155 Мбит/с). Эти сведения собирались в течение трех суток, запросы проводились в автоматическом режиме каждые 5 минут.

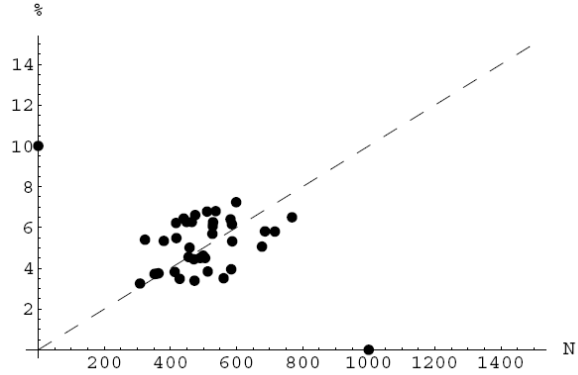

Рисунок 1.6 − Зависимость загрузки внешнего канала от числа активных потоков (HEAnet), 2004 год

Данные, полученные с сети FREEnet, для первого (рисунок 1.7) и второго (рисунок 1.8) маршрутизаторов, собирались в течение трех месяцев 2008 года, запросы проводились в автоматическом режиме каждые 30 минут. Суммарная нагрузка канала варьировалась в пределах сотен Мбит/с (100-200 Мбит/с) для первого маршрутизатора и десятков Мбит/с для второго маршрутизатора.

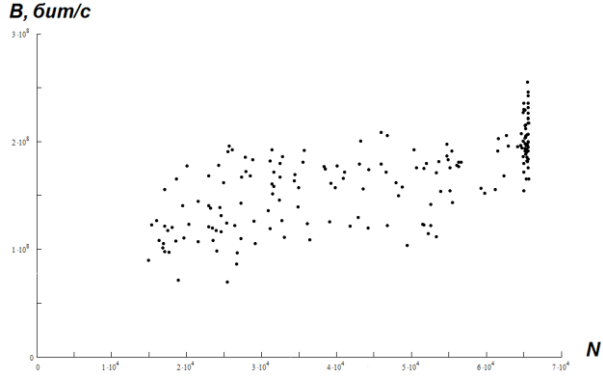

Рисунок 1.7 − Зависимость загрузки внешнего канала от числа активных потоков (FREEnet, набор данных 1)

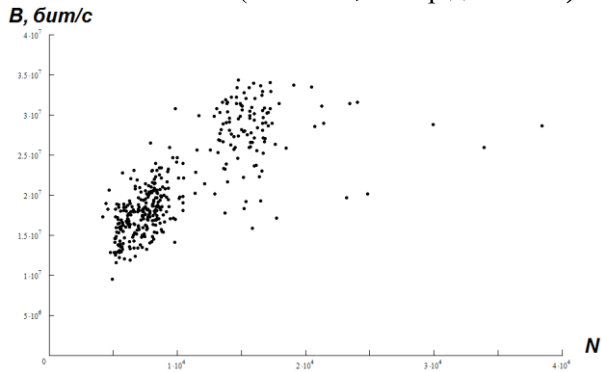

Рисунок 1.8 − Зависимость загрузки внешнего канала от числа активных потоков (FREEnet, набор данных 2)

Как уже упоминалось выше, первое тестирование проводилось на граничном маршрутизаторе ЗАО "СамараТелеком". В качестве внешних каналов использовались четыре потока E1, ведущих к различным магистральным Интернет-провайдерам.

Нами было получено несколько десятков значений для различной загрузки сети, и результат этих измерений представлен на графике с рисунка 1.5, где на оси абсцисс отложено количество активных потоков, а на оси ординат - загрузка канала в процентах от максимальной величины.

Отдельные точки на графике соответствуют реальным состояниям сети, а пунктирная прямая описывает ее идеальное состояние. Угол наклона этой прямой представляет усредненное отношение *B*/*N*, полученное для тех состояний, когда загрузка сети не превышала 40%. Для того чтобы найти рабочий участок и доверительный интервал для состояний сети был построен график с нанесенными на него дополнительными линиями (рисунок 1.9).

Сверху рабочий участок ограничен прямой в виде отдельных точек. Эта прямая характеризует поведение сети при больших нагрузках. Пересечение двух прямых позволяет найти длину рабочего участка (0%-45%). Когда число потоков превышает 2500, сеть начинает испытывать перегрузки, приводящие к замедлению ее работы и ухудшению качества связи. Причем с ростом запросов общая загрузка канала практически не растет, а качество связи значительно снижается.

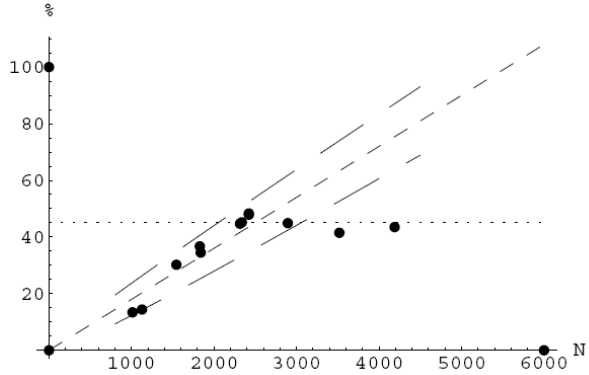

Рисунок 1.9 − Доверительный интервал и рабочий участок сети СамараТелеком

Отметим, что на исследуемой сети практически отсутствует переходный участок (см. рисунок 1.2), а в часы наибольшей нагрузки сети ее реальная пропускная способность сети более чем вдвое ниже объявленной.

В данном случае правило 50% для среднесуточной утилизации не работает. Даже предельная (а не среднесуточная) нагрузка в 45% приводит к сбоям в сети. То есть приведенный в данной работе тест позволяет судить о состоянии сети более тщательно, чем принятые до сих пор правила. Такая ситуация объяснялась тем, что из-за дороговизны каналов первичной транспортной сети и нежелания магистральных провайдеров предоставлять возможность передачи голоса и видео через публичные каналы, средняя утилизация магистральных каналов превышала 80%, а их пропускная способность была, по меньшей мере, втрое меньше пропускной способности подключенных к ним операторов. Предложенный тест показал наличие недостатков у вышестоящих поставщиков услуг интернет без доступа к их инфраструктуре связи.

На рисунке 1.10 представлен график состояния сети с нанесенными доверительными интервалами. Средняя скорость потока в сети HEAnet равняется 15 Кбит/с, что на порядок больше, чем у СамараТелеком и имеется возможность передавать голос, а на отдельных направлениях и качественное видеоизображение. Следует отметить, что по нашим наблюдениям среднее значение скорости потока в Самарской региональной сети науки и образования в 2003 году было равным 7-8 Кбит/с. Этот факт объясняется тем, что в рамках федеральной межведомственной программы по развитию сети для науки и образования RBNet была одной из первых магистральных сетей, перешедших на транспортные каналы верхнего уровня иерархии PDH (45 Мбит/с).

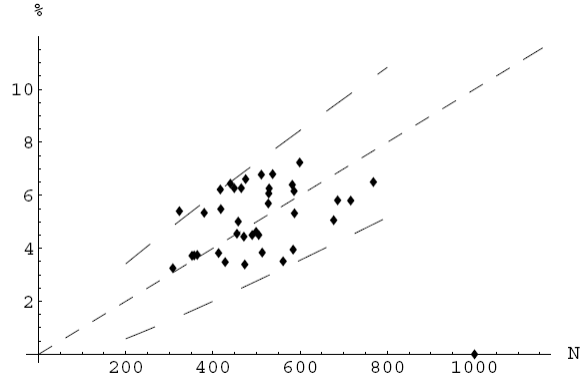

Рисунок 1.10 − Доверительный интервал и рабочий участок сети HEAnet

Для сети FREEnet графики состояния сети с нанесенными доверительными интервалами представлены на рисунках 9 и 10.

Полученные экспериментальные данные были разделены на интервалы в зависимости от числа активных потоков. Для каждого интервала вычислены параметры, характеризующие активные потоки. Результаты этих вычислений представлены в таблицах 1.1 и 1.2, соответственно, для первого и второго маршрутизатора. Здесь N – среднее значение числа активных потоков на интервале с номером n, B – средняя нагрузка на маршрутизаторе (мегабит в секунду, Мбит/с), b – средняя нагрузка на один поток (бит в секунду, бит/с), $\sigma(B)$ и $\sigma(b)$ – стандартное отклонение для потоковых нагрузок B и b.

Более ранние эксперименты, проведенные на сетях СамараТелеком и HEAnet не позволяют выполнить проверку адекватности модели с высокой точностью, поскольку содержат мало точек состояний сети.

Для проверки параболической формы доверительного интервала вычислен коэффициент корреляции между значениями $\sigma_i(B)$ и $\sqrt{N_i}$. Для первого и второго маршрутизатора коэффициенты равны соответственно 0.70 и 0.93. Это позволяет сделать вывод о высокой корреляции между теоретической моделью и данными, полученными в результате эксперимента.

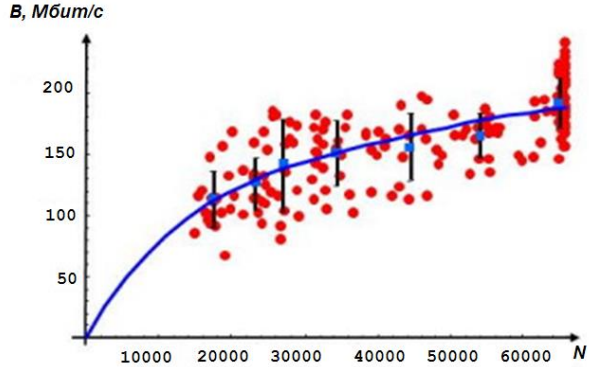

Рисунок 1.11 — График зависимости загрузки канала от числа активных потоков. Сеть FREEnet, набор данных 1.

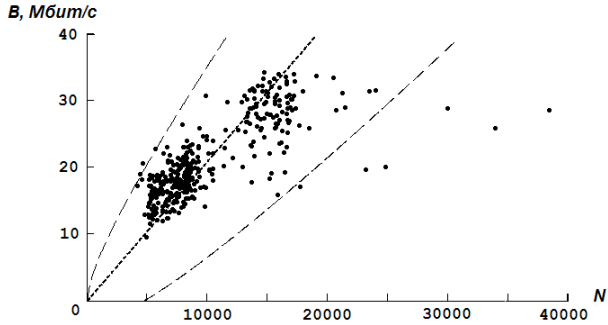

Рисунок 1.12 — Рабочий участок и доверительный интервал сети FREEnet, набор данных 2.

Определение длины рабочего участка, как это было теоретически обосновано на Рисунке1.2 проиллюстрировано на Рисунках 1.13 и 1.14

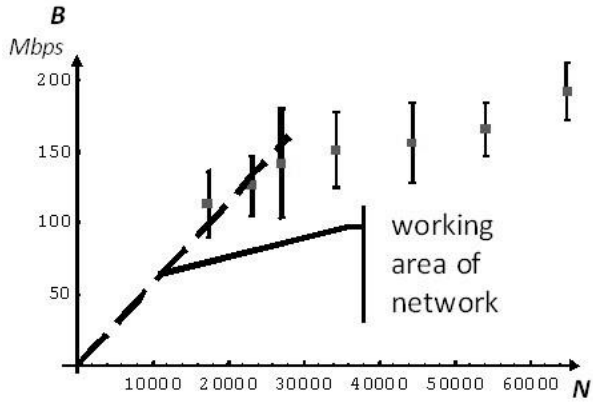

Рисунок 1.13 – Определение рабочего участка сети FREEnet, набор данных 1.

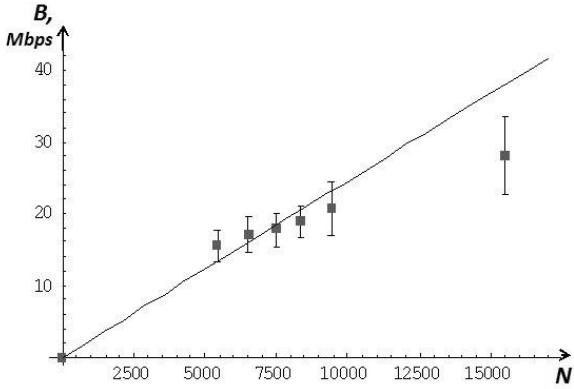

Рисунок 1.14 - Определение рабочего участка сети FREEnet, набор данных 1.

Проверка на нормальное распределение нагрузки на поток проведена с помощью критерия Пирсона χ^2. Результаты проверки приведены в таблицах 1.3 и 1.4. Во втором столбце таблиц показаны значения χ^2 для α=0.95, а в круглых скобках − табличные значения χ^2.

Таблица 1.1 — Параметры активных потоков сети FREEnet, набор-данных 1.

n	*N*	*B, Мбит/с*	*σ(B), Мбит/с*	*b, бит/с*	*σ(b), бит/с*
1	17489	113.1	23.10	6784	1386
2	23260	126.0	21.4	5682	965
3	27007	152.0	39.2	5628	1452
4	34902	156.7	26.9	4990	770
5	45104	163.9	33.9	3634	752
6	55019	176.3	33.2	3205	604
7	64778	215.4	42.2	3325	652

Таблица 1.2 — Параметры активных потоков сети FREEnet, набор данных 2.

n	*N*	*B, Мбит/с*	*σ(B), Мбит/с*	*b, бит/с*	*σ(b), бит/с*
1	5446	15.42	2.25	2843	413

2	6531	17.11	2.45	2364	377
3	7508	17.74	2.35	2364	313
4	8370	18.92	2.24	2261	268
5	9443	20.67	3.81	2190	404
6	15495	28.05	5.40	1811	349

Таблица 1.3 − Статистические тесты. Сеть FREEnet, набор данных 1.

n	χ^2 для α=0.95	Гауссов тест	Коэффициент корреляции
1	недостаточно данных		
2	недостаточно данных		
3	недостаточно данных		
4	3.49 (9.49)	+	0.70
5	недостаточно данных		
6	3.45 (7.81)	+	
7	0.50 (9.49)	+	

Таблица 1.4 − Статистические тесты. Сеть FREEnet, набор данных 2.

n	χ^2 для α=0.95	Гауссов тест	Коэффициент корреляции
1	9.15 (12.6)	+	
2	10.0 (11.1)	+	
3	3.24 (14.1)	+	0.93
4	10.0 (14.1)	+	
5	недостаточно данных		
6	1.94 (14.1)	+	

По данным таблиц 1.1 и 1.2 были рассчитаны числовые коэффициенты $\alpha_1 \approx 13$ и $\alpha_2 \approx 4.5$ для уравнения (1.2)

Полученные результаты подтверждают предположение о нормальном распределении суммарной нагрузки канала. Таким образом, модель трафика, основанная на понятии потоков, получила экспериментальное подтверждение.

В период нашего наблюдения был отмечен инцидент, заключавшийся во временном прекращении работы одного крупного

WWW/ftp узла у клиента, имеющего широкополосное подключение к исследуемой сети. Полученные данные сразу вышли за пределы доверительного интервала и образовали отдельный кластер значений, как это показано на Рис. 1.15.

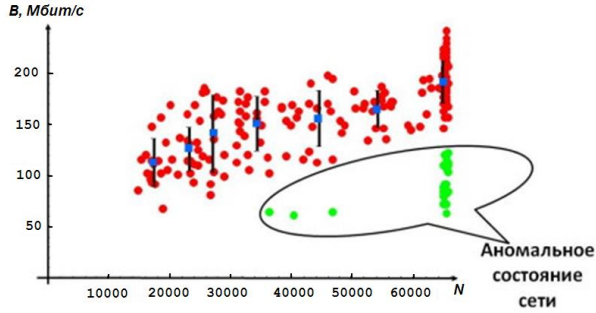

Рисунок 1.15 - Обнаружение аномальных состояний сети

1.7. Планирование и проведение экспериментов по определению параметров сетевых атак

На следующем этапе по проверке модели трафика необходимо выяснить, можно ли применить данную модель для задач сетевой безопасности, в частности, - для обнаружения сетевых атак.

Для того чтобы выяснить детали несанкционированного вторжения было решено провести эксперименты, имитирующие попытки атак. Они проводились на сети Самарского государственного аэрокосмического университета (СГАУ).

В качестве источника атаки использовались удаленные персональные компьютеры, подключенные к сети Интернет, находящиеся во внешней сети по отношению к исследуемой. Целью атаки являлся один из внутренних серверов сети СГАУ. В качестве NetFlow-сенсора был выбран пограничный маршрутизатор сети СГАУ Cisco 6509, NetFlow-коллектор — тот же сервер, который подвергался атаке.

При проведении сканирования был задействован только один компьютер, поскоольку атака сканирования портов производится с одиночных источников. Для сканирования применялась программа Nmap [7], которой было предписано провести полное сканирование всех портов атакуемого сервера.

Nmap — свободная утилита, предназначенная для разнообразного настраиваемого сканирования IP-сетей с любым количеством объектов, определения состояния объектов сканируемой сети (портов и соответствующих им служб). Nmap использует множество различных методов сканирования, таких как UDP, TCP (connect), TCP SYN (полуоткрытое), FTP proxy (прорыв через ftp), Reverse-ident, ICMP (ping), FIN, ACK, Xmas tree, SYN- и NULL-сканирование.

При осуществлении DDoS-атаки в качестве атакуемой цели был выбран тот же веб-сервер, что и при сканировании. Источниками атаки служили несколько компьютеров, находящихся во внешней сети. В первой части эксперимента атакующие компьютеры одновременно отправляли в течение получаса ping-запросы, осуществляя атаку ICMP-flood. Во второй части эксперимента атакующие компьютеры проводили DDoS-атаку при помощи специализированной программы LOIC. В течение часа веб-сервер подвергался атаке с применением различных типов трафика: HTTP, UDP, TCP. В ходе всех экспериментов производился сбор данных, которые впоследствии анализировались для выявления закономерностей разных типов атак.

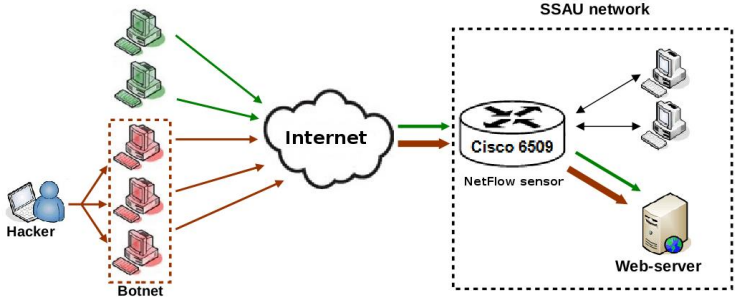

Рисунок 1.16 – Схема эксперимента

Данные о потоках, которые служат основой для анализа, собирались с пограничного маршрутизатора сети Cisco 6509. Для сбора данных с маршрутизатора использовался NetFlow-коллектор nfdump [8]. Экспорт NetFlow данных для анализа проводится с периодичностью пять минут. Каждые пять минут формируется файл с указанием параметров всех потоков, зафиксированных на маршрутизаторе в это время. Эти параметры перечислены во введении

и включают в себя: время начала потока, длительность потока, протокол передачи данных, адрес и порт источника, адрес и порт назначения, число переданных пакетов, число переданных данных в байтах.

В результате анализа данных, собранных во время сканирования сети, было выявлено резкое увеличение числа активных потоков при практически неизменном количестве переданного трафика (см. Рис.1.16). Каждый сканирующий компьютер генерировал в течении 5 минут порядка 10-20 тысяч очень коротких потоков (размером до 50 байт). При этом суммарное число активных потоков на маршрутизаторе, генерируемое всеми пользователями, составляло порядка 50-60 тысяч.

На рисунке 1.17 изображен график состояния сети, по оси абсцисс откладывается число завершившихся потоков N, по оси ординат - суммарная нагрузка канала в Мегабит в секунду (Мбит/с). Каждая точка на графике отражает состояние исследуемой сети за предшествующий пятиминутный интервал, показывая зависимость средней нагрузки канала от числа активных потоков. Точки соответствуют нормальным состояниям сети, а треугольники - состояниям сети, зафиксированным во время сканирования портов. Отрезки, изображенные на графике и параллельные оси ординат, показывают доверительные интервалы для средней нагрузки, рассчитанные для пяти промежутков потоков (20000-30000, 30000-40000, 40000-50000, 50000-60000, 60000-70000).

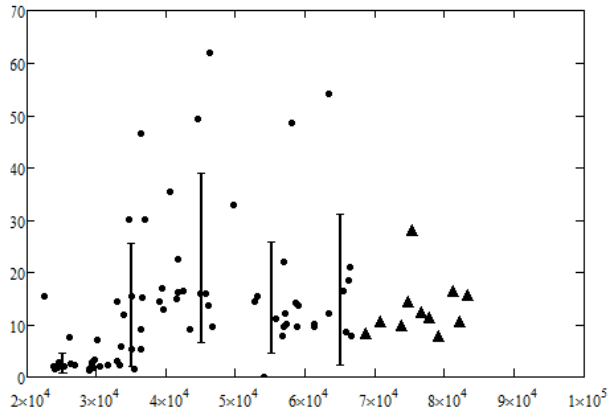

Рисунок 1.17 – Сканирование портов

По итогам эксперимента с ping-запросами было выяснено, что на каждый атакующий компьютер приходился всего один очень длинный поток ICMP трафика, если посылать запросы по единственному порту. Поскольку данные об одном потоке записываются только по его завершению, то необходимые данные были записаны в файл nfdump уже по завершению атаки. Было обнаружен один аномально длинный поток трафика по протоколу ICMP, источником являлся атакующий компьютер. Таким образом, в результате анализа экспериментальных данных удалось определить атаку типа ICMP-flood. Следует отметить, что для достижения результата – сбоев в работе информационной системы одного активного потока ICMP-трафика явно недостаточно, счет должен идти на десятки тысяч запросов.

Анализ эксперимента по моделированию DDoS атаки утилитой LOIC также показал резкое увеличение числа активных потоков наряду с увеличением передаваемого трафика. Утилита параллельно отсылает данные на разные порты цели, создавая тем самым большое количество коротких потоков длительностью до минуты (см. Рис.1.18). Треугольниками изображены состояния сети, зафиксированные во время атаки.

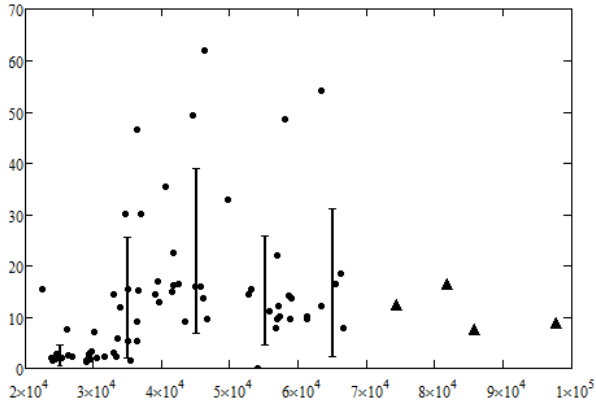

Рисунок 1.18 – DDoS-атака

Таким образом, стало очевидным, что при помощи протокола NetFlow возможно выявить не только момент начала атаки, но и определить ее тип. Подробное описание алгоритмов обнаружения

атак и работ по созданию защищенного хостинга можно найти в следующих разделах.

Литература

1. Bolla R., Bruschi R. RFC 2544 performance evaluation and internal measurements for a Linux based open router //High Performance Switching and Routing, 2006 Workshop on. – IEEE, 2006. – С. 6 pp.

2. Fraleigh C. et al. Packet-level traffic measurements from the Sprint IP backbone //IEEE network. – 2003. – Т. 17. – №. 6. – С. 6-16.

3. Park K., Kim G., Crovella M. On the relationship between file sizes, transport protocols, and self-similar network traffic //Network Protocols, 1996. Proceedings., 1996 International Conference on. – IEEE, 1996. – С. 171-180.

4. Fred S. B. et al. Statistical bandwidth sharing: a study of congestion at flow level //ACM SIGCOMM Computer Communication Review. – ACM, 2001. – Т. 31. – №. 4. – С. 111-122.

5. Barakat C. et al. A flow-based model for internet backbone traffic //Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment. – ACM, 2002. – С. 35-47.

6. Sukhov A. M. et al. Active flows in diagnostic of troubleshooting on backbone links //Journal of High Speed Networks. – 2011. – Т. 18. – №. 1. – С. 69-81.

7. Lyon G. F. Nmap network scanning: The official Nmap project guide to network discovery and security scanning. – Insecure, 2009.

8. Haag P. Watch your Flows with NfSen and NFDUMP //50th RIPE Meeting. – 2005.

2 Ранговые распределения для определения пороговых значений сетевых переменных и анализа DDoS атак

2.1 Введение

Экспоненциальный рост интернет трафика и числа информационных источников сопровождается быстрым увеличением числа аномальных состояний сети. Аномальные состояния сети объясняются как причинами техногенного характера, так и человеческим фактором. Распознание аномальных состояний, созданных злоумышленниками достаточно тяжело из-за того, что они имитируют действия обычных пользователей [1]. Поэтому такие аномальные состояния крайне сложно выявить и заблокировать. Задачи обеспечения надёжности и безопасности Интернет сервисов требуют изучения поведения пользователей [2,3] на конкретном ресурсе.

В данной статье пойдёт речь о выявлении аномальных сетевых состояний и методах противодействия DDoS атакам [4,5]. (Distributed Denial of Service, распределённая атака типа «отказ в обслуживании») – это такой тип атак, при котором некоторое множество компьютеров в сети Интернет, называемых «зомби», «ботами» или бот сетью (ботнет), по команде злоумышленника начинают отправлять запросы на сервис жертвы. Когда число запросов превышает возможности серверов жертвы, новые запросы от настоящих пользователей перестают обслуживаться и становится недоступным. При этом жертва несёт финансовые убытки.

Исследования, которые описаны в данной главе учебного пособия, используют унифицированный математический подход. Был выделен ряд важнейших сетевых переменных, которые генерирует внешний единичный IP адрес при обращении к заданному серверу или локальной сети. К таким переменным относятся: частота обращения к веб серверу (по заданному порту), число активных потоков, величина входящего TCP, UDP и ICMP трафика и т.д. Построенная инфраструктура позволила измерять величины для вышеперечисленных сетевых переменных.

После нахождения данных величин для анализируемых переменных в произвольный момент времени необходимо построить ранговое распределение. Для этого найденные значения располагаются в порядке убывания. Анализ сетевых состояний будет производиться путем сравнения соответствующих распределений.

Особенно наглядно это сравнение, когда распределения для аномального и обычного состояния сети построены на одном графике. Подобный подход позволяет легко определить границу между обычным и аномальным состоянием сети.

Эксперименты по DDoS атаке на сервис можно провести с помощью эмуляции в лабораторных условиях. При этом ценность полученных результатов значительно меньше, чем при DDoS атаке на введённый в эксплуатацию коммерческий сервис, так как эмулятор не может полностью воспроизвести реальную компьютерную сеть. Кроме того, для полноценного понимания принципов и методов DDoS атаки необходим опыт работы с ней. Поэтому авторы анонимно договорились о проведении реальной DDoS атаки на специально подготовленный веб сервис. В процессе атаки был записан сетевой трафик, собрана статистика NetFlow. Изучение ранговых распределений для числа потоков и различных типов входящего трафика, генерируемых единичным внешним IP адресом, что позволило определить пороговые значения. Превышение пороговых значений можно классифицировать как признак атакующего узла, что позволяет сделать выводы об эффективности способов обнаружения и методов противодействия.

2.2 Обзор предшествующих работ

Идея определения некоторых пороговых значений для выявления аномальных сетевых состояний была высказана достаточно давно, начиная с момента появления DoS атак [6]. Вопрос состоит только в том, для каких переменных необходимо искать эти пороговые значения. В работе [7] было предложено находить некую статистическую функцию, которая может рассматриваться как энтропия и по ее поведению делать вывод о начале атаки. Этот подход может быть использован для выявления малозаметных атак с запросами низкой интенсивности [8].

Работа [9] содержит анализ атак на DNS сервера при помощи специально сконструированной величины (detection value) для которой рассчитывается пороговое значение. Эта величина строиться с учетом числа запросов к DNS серверу и ответов на запросы в течении фиксированного временного интервала. Распознавание атаки с выявлением аномального поведения сети на основе нестандартных отклонений от обычного поведения, то есть когда откло-

нения значений важнейших сетевых переменных находятся в области превышают три сигмы от нормальных значений описано в статье [10].

Множество работ посвящено обнаружению несанкционированных вторжений при помощи анализа данных протокола NetFlow. Достаточно полный обзор можно найти в статье [11]. Среди методов обнаружения вторжений можно отметить статистический подход [12,13], когда анализ информации о потоках с помощью простейших законов теории вероятности позволяет выявить источники атак. Статистический подход отличается простотой, даёт надежные результаты, но область применения ограничена хорошо изученными типами атак.

Греческие авторы работы [14] предложили использовать для анализа атаки 5 переменных, отношение входящего UDP и ICMP трафика к исходящему, количество потоков, состоящих из одного пакета, количество потоков длительностью меньше 10 миллисекунд, а также количество новых потоков в секунду. Для этих переменных были определены пороговые уровни, при превышении которых можно говорить о начале атаки. Эта работа наиболее близка к предлагаемому нами подходу.

Одна из отличительных особенностей предлагаемого в данной работе подхода это применение рангового анализа для данных NetFlow [15]. Обнаружение атаки базируется на отклонениях от распределения Зипфа [16]. Для этого анализируется информация об активных или завершённых потоках за некоторый промежуток времени. Минимальное время сбора статистики NetFlow может варьироваться от одной до пяти минут. При нормальном функционировании сети ранжированный список количества потоков, генерируемых уникальным IP адресом, представляет типичное распределение Зипфа [17]. Атакующие адреса могут быть идентифицированы по превышению установленного порогового значения для числа активных или завершённых потоков [15]. Во время атак число потоков возрастает многократно.

В настоящее время особое внимание должно быть уделено противостоянию наиболее опасному виду DDoS атак по переполнению внешнего канала, ведущего к отдельному серверу, локальной организации или автономной системе. Следует упомянуть

одну из первых аналитических статей [18] по атакам на переполнение внешнего канала, в ней дается обзор источников атаки и стратегий защиты, приведены данные об атакующих мощностях.

2.2 Ранговые распределения и распознание аномальных сетевых состояний.

В 1994 Стив Глассман [19] впервые описал при помощи рангового распределения процесс резервирования интернет трафика. В дальнейшем область применения ранговых распределений для анализа сетевых процессов расширилась, с их помощью были описаны такие интернет процессы как запросы к поисковым системам, обращения к DNS серверам, популярность документов на сайте и многое другое. В настоящее время доступно несколько хороших обзоров [17,20], посвященных применению ранговых распределений для описания сетевых процессов.

Обычно интернет процессы описываются распределением Зипфа, которое гласит

$$p_i = \frac{p_1}{i^\alpha} \qquad (2.1)$$

здесь p_1 - наибольшее значение исследуемой величины, i - порядковый номер в ранжированном списке (списке по убыванию), а α - показатель степени. Следовательно, эти три величины и должны использоваться при анализе атаки.

Для распознания атаки и выявления ее источников сравниваются два ранговых распределения. Одно из этих распределений строиться в текущий момент времени, другое в некоторый предыдущий момент, который рассматривается в качестве нормального состояния сети. Ранее нами было предложено анализировать ранговые распределения для количества потоков, которые генерирует единичный IP адрес [15]. Установлено, что в момент атаки эта величина возрастает не менее, чем на порядок, как это показано на рис. ??

То есть для обнаружения момента начала атаки следует использовать величину k

$$k = \frac{p_1}{p_{tr}} \qquad (2.2)$$

Теперь вопрос заключается в том, как определить пороговое значение p_{tr}, которая стоит в знаменателе дроби из уравнения (2.2).

Для этого следует построить зависимость наибольшего значения исследуемой величины от времени $p_1(t)$. Необходимо собрать и обработать статистику за значительный период времени. Этот период должен составлять не менее недели, чтобы сгладить колебания. О практической реализации этого метода речь пойдет в главе 3. Здесь же заметим, что на основании этой зависимости можно найти пороговое значение p_{tr}, которое не должно превышаться в процессе нормальной эксплуатации сети ($p_1(t) \leq p_{tr}$). Именно это значение будет использовано для вычисления коэффициента k из уравнения (2.2).

Следующий вопрос состоит в определении набора сетевых переменных, для которых необходимо рассчитывать пороговые значения. Выбор сетевых переменных зависит от типа DDoS атаки. Если атака нацелена на нарушение какого-либо интернет сервиса, например, отказ в функционировании веб сервера, то следует анализировать число запросов к атакуемому ресурсу. Если атака ставит целью переполнение входящих каналов, то требуется собирать данные о всех типах входящего трафика (TCP, UDP, ICMP), а также информацию о числе активных потоков.

Поскольку тип атаки заранее неизвестен, то пороговые значения требуется вычислять для значительного числа переменных. Подобный набор переменных должен включать

- Общее число активных потоков на граничном маршрутизаторе
- Число активных потоков, которые генерирует единичный внешний IP адрес
- Входящий трафик, которые генерирует единичный внешний IP адрес, отдельно для каждого типа трафика (TCP, UDP, ICMP)
- Число запросов, которые генерирует единичный внешний IP адрес, отдельно для каждого типа сервиса (HTTP, FTP, mail, proxy, ssh, samba, MySQL, и т.д.)

После того, как пороговые значения p_{tr} для важнейших сетевых переменных будут найдены, необходимо регулярно рассчитывать соответствующие значения коэффициентов, задаваемых урав-

нением (2.2). Если значение этого коэффициента значительно превышает единицу, то следует говорить об аномальном состоянии сети.

2.3 Распределение количества потоков, генерируемых одиночным IP-адресом

Цель данного раздела состоит в уточнении параметров трафика, которые используются в программном обеспечении для предотвращения атак. Наиболее важными являются два параметра, а именно: максимальное число активных потоков, которое может генерировать один IP адрес и тип распределения ранжированного по популярности количества потоков с одного IP адреса. Следует отметить, что сбор статистики для анализа производится программным обеспечением один раз в минуту и все данные, которые анализируются относятся именно к этому временному промежутку.

Для нахождения переменных были использованы данные Netflow статистики с внешнего маршрутизирующего сервера СГАУ. Полученные результаты изображены графически на Рис. 2.1.

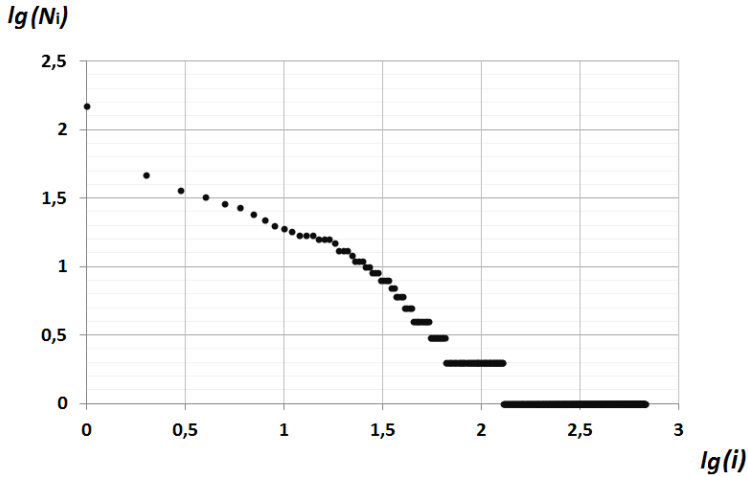

Рисунок 2.1. Количество активных потоков на один IP адрес в порядке убывания

Типичный график распределения активных потоков с одного IP адреса, ранжированных по популярности изображен на Рис.2.1.

По осям в логарифмическом масштабе отложены количество активных потоков и порядковый номер адреса в ранжированном списке. Точки на графике ложатся на прямую, что свидетельствует о том, что данное распределение подчиняется закону Зипфа (уравнение (2.1))

Для нахождения максимального количества активных потоков, приходящихся на один IP-адрес, была проанализирована статистика потокового трафика за неделю и получены графики, изображенные на Рис. 2.2 и 2.3.

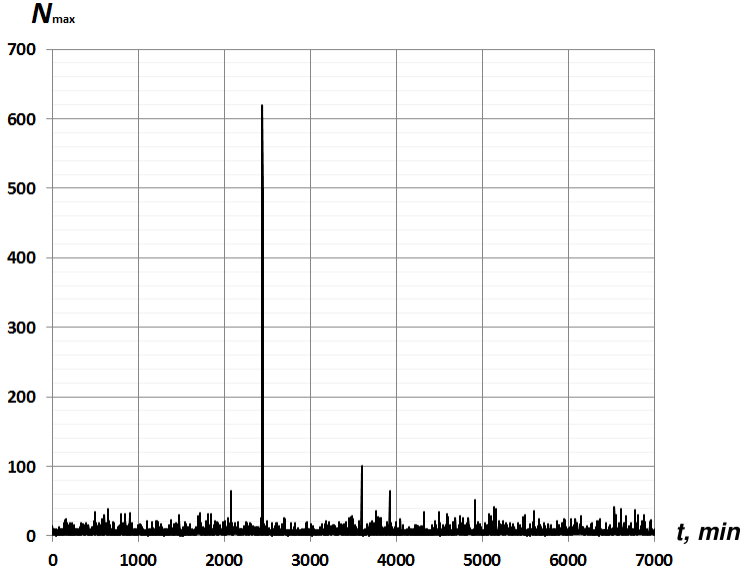

Рисунок 2.2. Динамика изменения количества активных потоков с сервера 91.222.128.200

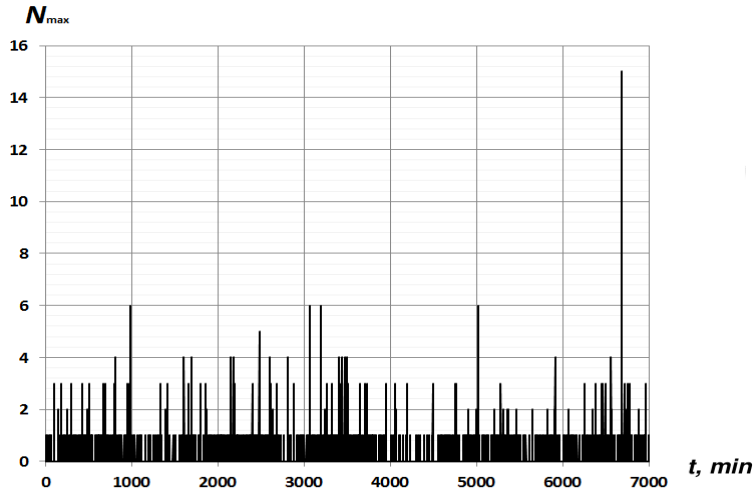

Рисунок 2.3. Динамика изменения количества активных потоков с сервера 91.222.129.201

Графики с Рис. 2.2 и 2.3 показывают, что максимальное число потоков с одного IP адреса, при нормальной работе сети не превышает 100 для сервера 91.222.129.201 и не превышает 20 для 91.222.128.200.

Таким образом, на основании закона Зипфа и данных о потоках, получаемых с маршрутизаторов, можно сформулировать правило определения границы отсечения по числу потоков для подозрительных IP-адресов: уровнем отсечения будет являться верхняя граница среди максимального количества потоков, приходящихся на единичный IP-адрес.

2.4 Алгоритм выявления сетевых атак двух типов: DDoS и сканирование портов

В результате проведенных исследований нам удалось выявить закономерности, позволяющие на основании NetFlow-данных определить IP-адреса компьютеров, с которых проводятся DDoS-атаки и сканирование портов. На основании этих закономерностей, был разработан алгоритм обнаружения атак. Прежде чем сформулировать данный алгоритм уточним формат записи данных для потока:

- Дата и время начала потока

- Длительность потока (в секундах, с точностью до тысячных)
- Название протокола передачи
- IP-адрес и порт источника
- IP-адрес и порт назначения
- Количество переданных пакетов
- Количество переданных байтов

Разработанный алгоритм обнаружения атак типа DDoS и сканирования портов заключается в следующем:

1. Найти IP-адреса источников, которые генерируют большое количество потоков.
   а) Если размер данных потоков очень короткий (44 байт), то происходит сканирование портов;
   б) Если размер потоков больше 44 байт, то с данного IP-адреса возможно осуществляется атака типа DoS.
2. Найти IP-адреса источников, которые генерируют очень длинные потоки (длительностью свыше 5 минут). В этом случае на IP-адрес назначение может проводиться DoS-атака.

В случае если одновременно обнаружено множество IP-адресов, с которых производится DoS-атака, то возможно проводится атака типа DDoS.

Для любой сетевой атаки важно своевременное обнаружение для того, чтобы как можно быстрее принять меры по её нейтрализации. NetFlow-данные поступают с маршрутизатора периодически, в зависимости от выставленных нами настроек. В то же время необходим баланс между частотой сбора статистики по потокам и временем, необходимым для ее обработки. Поэтому решено было установить периодичность запроса NetFlow данных один раз в одну минуту.

Следует отметить, что NetFlow-статистика содержит информацию о уже завершившихся потоках. Поскольку поток считается активным в течение определенного времени по его завершению, то подобные завершившиеся потоки также необходимо считать активными.

2.5 Листинг скрипта

Скрипт выполняет обработку файлов NetFlow-коллектора nfdump, поступающего с граничного маршрутизатора СГАУ Cisco

6509, после чего формирует список подозрительных IP-адресов, занося их в базу данных MySQL и блокируя на защищаемых серверах при помощи брандмауэра iptables.

Сначала объявляются все необходимые для обработки переменные: массивы записей параметров потоков, учетные данные входа в базу MySQL, папки с файлами nfdump, временные данные. Далее выполняется поиск последнего файла nfdump (папка /var/cache/nfdump/), поступившего с маршрутизатора. Файл nfdump представляет собой данные по всем потокам, завершившимся за последнюю минуту минуту. Этот файл преобразуется из бинарного формата в текстовый и сохраняется в указанную папку на сервере. После этого открывается текстовый файл и построчно записывается в массив (lines). В каждой строке выделяются все параметры потока и записываются в отдельные массивы. Таким образом, под каждый параметр потока отводится отдельный массив, номер элемента которого соответсвует порядковому номеру потока в исходном файле nfdump. Также формируются два массива, содержащих в себе информацию о потоках, завершенных на защищаемых серверах 91.222.128.200 и 91.222.129.201. Эти два массива сортируются по IP-адресам источников для того, чтобы подсчитать число потоков, приходящихся на каждый IP-адрес. Выполняется подсчет числа потоков, и если на один IP-адрес приходится более 300 потоков, то данный адрес заносится в список подозрительных адресов, блокируется брандмауэром iptables и заносится в базу данных MySQL. В конце блокировка снимается на те адреса, которые были заблокированы больше 5 минут назад. Адреса, которые были заблокированы, сохраняются в файле-отчете, где указывается IP-адрес и число потоков, приходящихся на него.

****

```
#!/usr/bin/perl
use 5.8.8; use strict; use warnings; use DBI;

my $hostname = "localhost";
my $database = "admin";
my $user = "admin";
my $password = "Rdhw89pF%lk9D&2";
my $dbh = 0;
my $sth = 0;
```

```
my $rc = 0;

my $path = "/var/cache/nfdump/";                    #путь к файлам
nfcapd.xxxxxxxxxxxx
my $path1 = "/root/DATA/";

#Исходные данные
my @lines = ();                 #массив строк с исходными дан-
ными
my $date = 0;                   #дата файла
my @time = ();                  #массив записей времени начала по-
токов
my @duration = ();              #массив записей длительностей по-
токов
my @protocol = ();              #массив записей названий протоко-
лов потоков
my @srcIP = ();                         #массив записей адресов ис-
точников потоков
my @srcPort = ();               #массив записей портов источников
потоков
my @dstIP = ();                         #массив записей адресов
назначения потоков
my @dstPort = ();               #массив записей портов назначения
потоков
my @packets = ();               #массив записей числа переданных в
потоке пакетов
my @bytes = ();                         #массив записей количества
переданных байт в
my $fl = 0;                     #Число потоков
my $fl2 = 0;                  # #Число потоков для второго сервера
my @masIP = ();                         #Массив отсортированных
IP-адресов
my @masIP2 = ();              # #Массив отсортированных IP-адресов
для второго сервера
my @masIPstream = ();           #Массив отсортированных IP-адре-
сов stream
my $count = 0;                  #Счетчик числа потоков
#Вторичные данные
```

```
my @mIP = ();
my @mIP2 = ();                          #
#Временные данные
my $i = 0;
###############

#поиск последнего файла
chdir $path;
my (@file_list) = glob "nfcapd.20*";
my $flowfile = 0;
$flowfile = $file_list[$#file_list];
##############################
#преобразование из формата nfdump в txt и копирование в директорию $path1
my $comand = 0;
$comand = "nfdump -r ".$path.$flowfile." >> ".$path1.$flowfile.".txt";
system $comand;
#$comand = "nfdump -r ".$path.$flowfile." >> /var/www/galcev/data/".$flowfile.".txt";
#system $comand;
$comand = "rm -f ".$path.$flowfile;
system $comand;

open(FileData,$path1.$flowfile.".txt");
        #открытие файла с исходными данными
@lines = <FileData>;                                    #запись в массив строк с исходными данными
close(FileData);                                        #закрытие файла с исходными данными

for ($i = 1; $i < @lines-4; $i++) {                     #заполнение массивов исходных данных

        if( $lines[$i] =~ m/\d+\:\d+\:\d+\.\d+/g ){
                                #поиск в строке времени
                $time[$fl]=substr($lines[$i],pos($lines[$i])-length($&),length($&));
        }
```

```
        if( $lines[$i] =~ m/\d+\.\d+/g ){
                        #поиск в строке длительности потока
                $duration[$fl]=substr($lines[$i],pos($lines[$i])-
length($&),length($&));
        }
        if( $lines[$i] =~ m/\w+/g ){
                                #поиск в строке названия протокола
                $protocol[$fl]=substr($lines[$i],pos($lines[$i])-
length($&),length($&));
        }
        if( $lines[$i] =~ m/\d+\.\d+\.\d+\.\d+/g ){
                                #поиск в строке IP-адреса источника
пакета
                $srcIP[$fl]=substr($lines[$i],pos($lines[$i])-
length($&),length($&));
        }
        if( $lines[$i] =~ m/\:\d+/g ){
                                #поиск в строке порта источника па-
кета
                $srcPort[$fl]=substr($lines[$i],pos($lines[$i])-
length($&)+1,length($&));
        }
        if( $lines[$i] =~ m/\d+\.\d+\.\d+\.\d+/g ){
                                #поиск в строке IP-адреса назначе-
ния
                $dstIP[$fl]=substr($lines[$i],pos($lines[$i])-
length($&),length($&));
        }
        if( $lines[$i] =~ m/\:\d+/g ){
                                #поиск в строке порта назначения
                $dstPort[$fl]=substr($lines[$i],pos($lines[$i])-
length($&)+1,length($&));
        }
        if( $lines[$i] =~ m/\d+/g ){
                                #поиск в строке числа пакетов в по-
токе
                $packets[$fl]=substr($lines[$i],pos($lines[$i])-
length($&),length($&));
```

```
        }
        if( $lines[$i] =~ m/\d+\.\d+\s\w+|\d+/g ){
                                #поиск в строке числа байтов в по-
токе
                $bytes[$fl]=substr($lines[$i],pos($lines[$i])-
length($&),length($&));
    #ФИЛЬТР ПО АДРЕСУ IP4TV.RU(91.222.128.200) и  сервер
трансляций (91.222.129.201)


                if ($dstIP[$fl] eq "91.222.128.200") {
                        push(@masIP2 , [$srcIP[$fl],$time[$fl]]);
                        ++($fl2);
                }
                if ($dstIP[$fl] eq "91.222.129.201") {
                        push(@masIP , [$srcIP[$fl],$time[$fl]]);
                        ++($fl);
                }

        }
    }
    --($fl);
--($fl2);
#СТАТИСТИКА

#массив mIP - сортировка данных по IP-адресу источника
@mIP = sort {
   ( $a->[0] cmp $b->[0] )
        } @masIP;

@mIP2 = sort {
   ( $a->[0] cmp $b->[0] )
        } @masIP2;
#Формирование файла (IP-адрес источника;Число потоков > 300)
$count = 1;
open(IPToBan,">> /var/www/galcev/banIP.txt");
$dbh  =  DBI->connect("DBI:mysql:$database:$hostname",  $user,
$password)|| print IPToBan "Got error". $dbh->errstr ."\n";
```

```
my $statement = 0;

for ($i = 1; $i < $fl; $i++) {
        if ($mIP[$i-1][0] eq $mIP[$i][0]) {++($count);}
        else {
                if ($count > 300){

                        my $ipT = $mIP[$i-1][0];

                        $statement = "select ban('".$ipT."');";
                        print IPToBan substr($flowfile,13,2),"\.",sub-
str($flowfile,11,2),"\.",substr($flowfile,7,4)," ",$mIP[$i-1][1],"
",$ipT," ",$count,"\n";
                        $sth = $dbh->prepare($statement);
                        $sth->execute;
                        if ($sth == 1){
                                $comand = "iptables -I fail2ban-galcev
-s ".$ipT." -j DROP";
                                system $comand;
                        }

                }
                $count = 1;
        }
    }

my $ref = 0;
$statement = "SELECT id,INET_NTOA(ip) FROM banlist WHERE
time<CURRENT_TIMESTAMP-300;";
$sth = $dbh->prepare($statement);
$sth->execute;
while ($ref = $sth->fetchrow_arrayref) {
        $comand = "iptables -D fail2ban-galcev -s ".$$ref[1]." -j
DROP";
        system $comand;
        my $stm = "DELETE FROM banlist WHERE
id='".$$ref[0]."';";
        $sth = $dbh->prepare($stm);
```

```
        $sth->execute;
}


$rc = $sth->finish;
$rc = $dbh->disconnect;
close(IPToBan);
##########################################
open(IPToBan2,">> /var/www/galcev/banIP2.txt");
$count = 1;
for ($i = 1; $i < $fl2; $i++) {
        if ($mIP2[$i-1][0] eq $mIP2[$i][0]) {++($count);}
        else {
                if ($count > 0){
                        print IPToBan2 substr($flowfile,13,2),"\.",sub-
str($flowfile,11,2),"\.",substr($flowfile,7,4),"          ",$mIP2[$i-1][1],"
",$mIP2[$i-1][0]," ",$count,"\n";
                }
                $count = 1;
        }
    }
    close(IPToBan2);
    ####################################
    exit( 0 );
```

2.6 Требования к веб-хостингу с точки зрения сетевой безопасности

Развитие глобальной сети привело к тому, что все больше информации обычные люди стали получать из Интернет. Сейчас уже трудно найти государственное учреждение, общественную организацию или коммерческую фирму, которая не имеет собственного сайта. Но создание сайта, его успешная поддержка и регулярное обновление еще не гарантия успешного представительства в глобальной сети.

Всякая успешно действующая структура нуждается в минимальной защите, особенно в России, особенно в области информационного обеспечения. Не секрет, что появилась целая отрасль, ко-

торая занимается целенаправленными атаками на информационные ресурсы [21]. Мощности нападения постоянно растут, так же совершенствуются и механизмы атак. В связи с этим необходимо представлять, насколько защищена ваша собственная информационная инфраструктура [22].

В данной работе будет сделана попытка проанализировать параметры хостинга и понять, какой мощности атака может вывести из строя базирующуюся информационную систему [23]. Будут сформулированы минимальные требования к веб хостингу и описаны основные переменные, влияющие на производительность. Кроме того, будут сформулированы пути повышения устойчивости хостинга к атакам, которые можно реализовать без значительных затрат [24].

2.7 Методики обнаружения сетевых атак

Для того, чтобы обнаружить несанкционированное сетевое вторжение необходимо проанализировать либо входящий трафик, либо его обобщенные состояния. При атаке сеть зараженных компьютеров (ботнет) посылает одинаковые запросы к хостингу, чаще всего с одинаковой периодичностью. Эти свойства атакующего трафика и положены в основу механизмов обнаружения атак.

На уровне трафика необходимо анализировать данные, которые содержатся в пакетах. Запросы от атакующих хостов содержат одинаковое тело пакетов, различаются лишь их заголовки. Сравнивая пакетные данные можно выбрать из заголовков атакующие адреса, составив базу для дальнейшего блокирования. Недостаток метода заключается в том, что необходимо хранить полный трафик и анализировать его, что приводит к значительным временным затратам, требуются вычислительные мощности и дисковая память.

На практике полный трафик не храниться, обычно сохраняются только его агрегированные состояние, данные о потоках (flow). Они содержат IP адреса отправителей и получателя, номера портов, объем переданной информации, время соединения и т.д. Этот формат позволяет сжать объем хранимой информации в 250 раз.

На уровне потоков также разработаны методы по обнаружению DDoS атак. Начало атаки характеризуется резким ростом числа активных потоков. При этом атакующие IP адреса также ге-

нерируют значительное количество потоков. На практике нам доступны данные о потоках, окончившихся за последнюю минуту (или пять, в зависимости от настроек NetFlow). Анализируя этот файл можно выделить данные об атакующих адресах.

На практике оба метода могут комбинироваться, первоначальное обнаружение атаки и ее источников может проводиться с помощью NetFlow. А уточнение методов и точный список атакующих адресов на уровне пакетов.

Чем большее количество запросов отправляет атакующий хост, тем быстрее его можно засечь. Для того, чтобы затруднить обнаружение атаки необходимо постоянное изменение содержания запросов и несоблюдение периодичности запросов. При большом размере ботнета запросы можно делать через значительные промежутки времени, что также затрудняет идентификацию атаки.

2.8 Параметры хостинга

Обычно для размещения сайта или портала арендуют серверные и сетевые мощности в дата центре, который обеспечивает, по меньшей мере, минимальные требования к безопасности. Интернет провайдеры и крупные потребители информации могут размещать информационные ресурсы в своей собственной сети, обеспечивая безопасность самостоятельно.

Попробуем сформулировать минимальные требования к вебхостингу, которые должен предоставлять оператор дата центра. Но до этого остановимся на перечне важнейших переменных, описывающих состояние сети. Эти переменные нам понадобятся в дальнейшем, при вычислении параметров ботнетов и проводимых ими атак [25].

Основная сетевая переменная это суммарная ширина каналов $C = \sum_i C_i$, обеспечивающих связанность информационного сервера. Причем важны обе компоненты сетевого соединения, как емкость входящих каналов C^{int}, так и исходящих C^{out}. Причем исходящие каналы даже важнее, так как пользователи именно по ним получают данные.

Следующая группа переменных относится к параметрам вебсервера(ов), которые осуществляют формирование веб странички для пользователей. Таких параметра три:

- N - число одновременно обслуживаемых запросов. То есть, сколько страниц сервер может формировать одновременно. Для стандартных серверов начального уровня (два четырехядерных процессора, 8 GB RAM, RAID SAS 300 GB) этот параметр варьируется от 10 до 20.
- N^q – длина очереди запросов. То есть, сколько запросов на обслуживание может храниться в системе для последующей обработки. Обычно этот параметр равен 500, при превышении этого числа запросы просто отбрасываются без обслуживания.
- T^s – время формирования одной веб странички. Это время, которое тратит сервер, чтобы сгенерировать страничку после начала обработки запроса. Это время очень сильно зависит от типа используемого движка. Статические страницы выдаются почти мгновенно, в то время как CMS Joomla тратит на формирование страницы 280 *ms*, а CMS Wordpress – 430 *ms*. Поэтому сайты с наибольшей загрузкой работают на собственных движках.

Попробуем рассчитать теперь разумную суточную загрузку стандартного веб сервера начального уровня с приведенными выше параметрами. Для этого воспользуемся теоремой Литтла:

$$N = \lambda T^s, \tag{2.3}$$

то есть, для одних суток T^z число запросов может без проблем достигать

$$N^z = \frac{NT^z}{3T^s}, \tag{2.4}$$

коэффициент 3 выбран для того, чтобы сгладить суточные пиковые нагрузки. Подставив значения, получим, что такой сервер может без труда сгенерировать не менее миллиона веб страниц в день. В обычных условиях на такой базе можно разместить сотни серверов небольших компаний.

Попробуем оценить верхнюю границу для сетевого трафика такого сервера, при условии, что каждая страница состоит в среднем из n =10 файлов средним размером S =10 KB. Тогда

$$\mathrm{C} = \frac{nNS}{T^s} \approx 30\ Mbps, \tag{2.5}$$

Ширина исходящего канала сервера может достигать 30 *Mbps*.

2.9 Расчет атакующих мощностей

Достигнуть сбоев в работе вебхостинга можно двумя способами [26,27]:

- Создать значительный поток запросов к серверу, так, что он не только не сможет его обслужить, но и сохранить все запросы в очереди.
- Создать значительный поток трафика из запросов, переполнив, таких образом, емкость входящего канала. И сделав невозможной дальнейшее получение запросов на обслуживание

То есть в обоих случаях требуется создавать значительный входящий поток. Успешно решить данную задачу возможно только в том случае, когда число атакующих хостов велико, и каждый из них генерирует небольшую нагрузку. В противном случае достаточно легко засечь IP адреса, с которых ведется атака, и заблокировать трафик с них [28].

Обозначим через M - число компьютеров в ботнете и пусть каждый из них посылает запросы с частотой k. Тогда для входящего потока с интенсивностью λ из уравнения (2.3)

$$\lambda = Mk, \tag{2.6}$$

для перегрузки сервера запросами необходимо от каждого из M хостов посылать запросы не чаще, чем один раз в минуту. Такая частота запросов достаточно трудно обнаруживается, но для генерации критического потока запросов необходим ботнет из

$$M = \frac{N}{kT^S} \approx 4200 \tag{2.7}$$

Такой размер ботнета достаточно большой, но все равно при анализе достаточно большой выборки данных, приблизительно несколько часов, этот ботнет может быть обнаружен по данным NetFlow, так как IP адреса его хостов будут постоянно повторяться.
Попробуем проанализировать теперь уровень атаки на переполнение канала к стандартному серверу, который тяжело обнаружить. Предположим, что каждый из M хостов ботнета генерирует трафик со скоростью b *Kbps*. Принимая средний размер потока равен среднему размеру файла $S = 10$ KB, оценим уровень обнаружения d в один поток в минуту. Тогда для малозаметного нарушения работоспособности сервера с шириной внешнего канала $C = 100$ *Mbps* требуется ботнет из

$$M = \frac{3C}{dS} \approx 225\ 000 \quad (2.8)$$

225 000 компьютеров при трехкратном превышении емкости канала. При этом каждый из компьютеров будет генерировать чуть больше 1 *Kbps* трафика. Мощность подобной атаки может быть повышена, но тогда сократится время на ее обнаружение. Создать такого размера ботнет очень тяжело, их в мире не более 10. Отсюда можно сделать вывод, что при подключении по каналу 1 Gbps необходима атака очень высокой мощности, порядка 100 *Kbps* с приблизительно 25 тысяч хостов. Такая атака может быть легко обнаружена и заблокирована при применении современных методов борьбы.

2.10 Специальные настройки для хостинга

Несмотря на приведенные данные, которые говорят о возможности хостинга на стандартных серверах начального уровня для большинства информационных ресурсов, ряд проектов нуждаются в особых условиях для хостинга [29]. Это информационные ресурсы с большой посещаемостью и особыми требованиями к трафику (например, сервера интернет телевидения). Для таких сайтов необходимо улучшение параметров хостинга [30].

Уравнение (2.3) определяет два пути для повышения устойчивости к атакам ботнетов [31]. Первый путь состоит в повышении количества одновременно обрабатываемых запросов. Это достигается путем распараллеливания обработки запросов, когда высокопроизводительный сервер распределяет запросы по нескольким синхронизированным базам, часто разнесенным по географическому признаку. Есть даже специальные хостинги с применением технологии Content Delivery Network [32].

Второй путь состоит в уменьшении времени обработки запросов, для этого крупные сайты, использующие систему динамического формирования веб страницы, пишут свои собственные движки, оптимизированные к структуре информации. При этом, время обработки запроса может быть уменьшено на порядок по сравнению с традиционными CMS, до 10-50 миллисекунд.

Проведение этих двух мероприятий позволяет без особых затрат увеличить количество обрабатываемых запросов с 40 до 10 000 в секунду (параметр λ из уравнения (1)). Дальнейшее повышение требует применения специальных технологий.

Есть еще один резерв повышения производительности. Как известно, разные страницы информационного ресурса по-разному востребованы пользователями. Обычно, популярность p_i (количество запросов) описывается распределением Зипфа. Это ранговое распределение, в котором документы располагаются в порядке убывания популярности, следуя зависимости (2.1)

Как правило, наиболее популярной ($i = 1$) является начальная страничка сайта. Поэтому целесообразно сделать ее статической, чтобы снизить до минимума время ее формирования. Также статическими рекомендуется сделать десять наиболее популярных страниц. По этому пути идут многие поисковые системы, например, Google.

2.11 Выводы

В данной главе была сделана попытка проанализировать условия, предоставляемые веб хостингом с точки зрения защиты от атак со стороны ботнетов. Рассмотрены основные параметры хостинга, влияющие на мощность атаки, которая приведет к нарушению нормальной работы информационного ресурса. Для двух типов атак – превышение предельного количества запросов и переполнение канала сформулированы предельные размеры ботнета и интенсивность запросов, которые могут осуществить малозаметную атаку.

На основании расчетов по определению уровня атаки были сделаны выводы по улучшению параметров веб-хостинга, которые можно осуществить за счет сокращения время обработки запроса, при параллельной обработке запросов с целью повышения количества одновременно обрабатывающихся запросов или заменой наиболее популярных страниц на статические.

Тем не менее, исследования в данном направлении должны быть продолжены, с тем, чтобы определить критерии малозаметного уровня атаки. Необходимо также постоянно отслеживать время формирования страниц на наиболее популярных движках CMS, как коммерческих, так и свободных. Эта величина, как и количество одновременно обслуживаемых запросов, должна измеряться в зависимости от производительности серверной платформы.

Литература

[1] Singh S., Gyanchandani M. Analysis of Botnet behavior using Queuing theory //International Journal of Computer Science & Communication. – 2010. – ISSN: 0973-7391. – Т. 1. – №. 2. – С. 239-241.
[2] Stanton J. M., Stam K. R., Mastrangelo P., Jolton, J. Analysis of end user security behaviors //Computers & Security. – 2005. – ISSN: 0167-4048. – DOI: 10.1016/j.cose.2004.07.001. – Т. 24. – №. 2. – С. 124-133.
[3] Hochheiser, H. Shneiderman, B. Understanding patterns of user visits to web sites: interactive starfield visualizations of WWW log data //Proceedings of the ASIST Annual Meeting – 1999. – ISSN 0044-7870.
[4] Mirkovic J., Reiher P. A taxonomy of DDoS attack and DDoS defense mechanisms //ACM SIGCOMM Computer Communication Review. – 2004. – ISSN: 0146-4833. – DOI: 10.1145/997150.997156. – Т. 34. – №. 2. – С. 39-53.
[5] Douligeris C., Mitrokotsa A. DDoS attacks and defense mechanisms: classification and state-of-the-art //Computer Networks. – 2004. – ISSN: 1389-1286. – DOI: 10.1016/j.comnet.2003.10.003. – Т. 44. – №. 5. – С. 643-666.
[6] Jiang J., Papavassiliou S. Detecting network attacks in the internet via statistical network traffic normality prediction //Journal of Network and Systems Management. – 2004. – Т. 12. – №. 1. – С. 51-72.
[7] Feinstein L. et al. Statistical approaches to DDoS attack detection and response //DARPA Information Survivability Conference and Exposition, 2003. Proceedings. – IEEE, 2003. – Т. 1. – С. 303-314.
[8] Bhuyan M. H., Bhattacharyya D. K., Kalita J. K. Information metrics for low-rate DDoS attack detection: A comparative evaluation //Contemporary Computing (IC3), 2014 Seventh International Conference on. – IEEE, 2014. – С. 80-84.
[9] Sun C., Liu B., Shi L. Efficient and low-cost hardware defense against DNS amplification attacks //IEEE GLOBECOM 2008-2008 IEEE Global Telecommunications Conference. – IEEE, 2008. – С. 1-5.
[10] Tan Z. et al. A system for denial-of-service attack detection based on multivariate correlation analysis //IEEE transactions on parallel and distributed systems. – 2014. – Т. 25. – №. 2. – С. 447-456.
[11] Li B., Springer J., Bebis G., Hadi Gunes M. A survey of network flow applications //Journal of Network and Computer Applications. –

2013. – ISSN: 1084-8045. – DOI: 10.1016/j.jnca.2012.12.020. – Т. 36. – №. 2. – С. 567-581.
[12] Proto A., Alexandre L. A., Batista M. L., Oliveira I. L., Cansian,A. M. Statistical model applied to netflow for network intrusion detection //Transactions on computational science XI. – Springer Berlin Heidelberg, 2010. – ISBN: 978-3-642-17696-8. – ISSN: 0302-9743. – DOI: 10.1007/978-3-642-17697-5_9. – С. 179-191.
[13] Sawaya Y., Kubota A., Miyake Y. Detection of attackers in services using anomalous host behavior based on traffic flow statistics //Applications and the Internet (SAINT), 2011 IEEE/IPSJ 11th International Symposium on. – IEEE, 2011. – ISBN: 978-1-4577-0531-1. – DOI: 10.1109/SAINT.2011.68. – С. 353-359.
[14] Siaterlis C., Maglaris V. Detecting incoming and outgoing DDoS attacks at the edge using a single set of network characteristics //10th IEEE Symposium on Computers and Communications (ISCC'05). – IEEE, 2005. – С. 469-475.
[15] Sukhov A. M., Sidelnikov D. I., Platonov A. P., Strizhov M. V., Galtsev A. A. Active flows in diagnostic of troubleshooting on backbone links //Journal of High Speed Networks. – 2011. – ISSN: 0926-6801. – DOI: 10.3233/JHS-2011-0447. – Т. 18. – №. 1. – С. 69-81.
[16] Zipf G. K. Relative frequency as a determinant of phonetic change //Harvard Studies in Classical Philology. – 1929. – С. 1-95.
[17] Крашаков С. А., Теслюк А. Б., Щур Л. Н. Об универсальности рангового распределения популярности веб-серверов //Вестник РФФИ. – 2004. – ISSN: 1605-8070.– С. 46-66.
[18] Geva M., Herzberg A., Gev Y. Bandwidth Distributed Denial of Service: Attacks and Defenses //IEEE Security & Privacy. – 2014. – Т. 12. – №. 1. – С. 54-61.
[19] Glassman S. A caching relay for the World Wide Web //Computer Networks and ISDN Systems. – 1994. – v. 27. – n. 2. – p. 165-173.
[20] Dorogovtsev S. N., Mendes J. F. F. Evolution of networks: From biological nets to the Internet and WWW. – Oxford University Press, 2013.
[21] W.Lee and S. J. Stolfo. Data mining approaches for intrusion detection, Proceedings of the 7th USENIX Security Symposium
[22] X. Ye, Countering ddos and xdos attacks against web services, in IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, vol. 1, 2008, pp. 346–352

[23] D. Menasce, V. Almeida, Capacity planning for Web performance: Metrics, models, and methods, Prentice Hall, NJ, 1998 (ISBN 0136938221)
[24] A. Greenhalgh, F. Huici, M. Hoerdt, P. Papadimitriou, M. Handley, and L. Mathy. Flow Processing and The Rise of Commodity Network Hardware. SIGCOMM CCR, Apr. 2009
[25] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing. RFC 2827, May 2000
[26] K.J. Houle, G.M. Weaver, Trends in denial of service attack technology, CERT, October 2001
[27] G. Gu, R. Perdisci, Junjie Zhang, and W. Lee. BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In Proceedings of the 17th USENIX Security Symposium (Security'08), San Jose, CA, July 2008
[28] A. Karasaridis, B. Rexroad, and D. Hoeflin. Wide-scale botnet detection and characterization. In USENIX Workshop on Hot Topics in Understanding Botnets (HotBots'07), 2007
[29] Y. Wang, H. Yang, X. Wang, and R. Zhang, Distributed intrusion detection system based on data fusion method, Intelligent control and automation, WCICA 2004
[30] KANDULA, S., SENGUPTA, S., GREENBERG, A., PATEL, P., AND CHAIKEN, R. The Nature of Data Center Traffic: Measurements & Analysis. In Proceedings ACM IMC 2009
[31] R. Sommer and V. Paxson. Outside the Closed World: On Using Machine Learning For Network Intrusion Detection. In Proceedings of the IEEE Symposium on Security and Privacy, 2010
[32] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F.Jahanian, "Internet Traffic and Content Consolidation," 77th Internet Engineering Task Force, 2010

3. ИЗУЧЕНИЕ ПОЛЬЗОВАТЕЛЬСКОЙ АУДИТОРИИ ДЛЯ ЗАДАЧ СЕТЕВОЙ БЕЗОПАСНОСТИ

3.1. Постановка задачи

В этой главе пойдёт речь о методах противодействия DDoS атакам [1,2] (Distributed Denial of Service, распределённая атака типа «отказ в обслуживании») – это такой тип атак, при котором некоторое множество компьютеров в сети Интернет, называемых «зомби», «ботами» или бот сетью (ботнет), по команде злоумышленника начинают отправлять запросы на сервис жертвы. Такие сети имитируют действия обычных пользователей [3], поэтому их крайне сложно выявить и заблокировать. Когда число запросов превышает возможности серверов жертвы, новые запросы от настоящих пользователей перестают обслуживаться и становится недоступным. При этом жертва несёт финансовые убытки.

Задачи обеспечения надёжности и безопасности Интернет сервисов требуют изучения поведения пользователей [4,5] на конкретном ресурсе. Все пользователи Интернет сервиса могут быть разделены на две категории: пользовательское ядро и новички. К пользовательскому ядру следует отнести тех посетителей, которые регулярно обращаются к исследуемому ресурсу [6]. Причем перерывы по времени между обращениями от одного пользователя могут достигать нескольких недель. Первой задачей данного исследования является определение периода времени, за которое анализ статистики посещений позволит выявить пользовательское ядро, вторая задача состоит в том, чтобы оценить предельную долю адресов из пользовательского ядра в общей базе IP адресов.

Дополнительной задачей является определение особенностей поведения новых (посетивших ресурс впервые) пользователей и процента запросов, принадлежащих им. К особенностям поведения можно отнести количество и долю запросов при пользовании сервисом, включая ранжированный список для всех новых пользователей, период времени непрерывного пользования сервисом и т.д. [7]. Прежде всего, следует сравнить поведение новых и старых пользователей, и найти корреляцию между частотой посещений и количеством запросов к сервису.

При отражении DDoS атаки можно использовать разные стратегии обороны. Например, искать классификационные признаки IP

адресов, с которых происходит атака, и по ним ограничивать доступ к ресурсу [8]. Однако в начале атаки всегда требуется время на распознание особенностей атаки, формулировку квалификационных признаков для нежелательных запросов и идентификацию атакующих IP адресов. Длительность этих операций может достигать нескольких часов, в течение которых сервис может быть недоступен. В это время можно обрабатывать запросы только постоянной аудитории, то есть пользовательского ядра. Знание статистических особенностей поведения новых пользователей облегчит поиск типов атакующих запросов и формирование списков блокировки доступа атакующих IP адресов.

Эксперименты по DDoS атаке на сервис можно провести с помощью эмуляции в лабораторных условиях. При этом ценность полученных результатов значительно меньше, чем при DDoS атаке на введённый в эксплуатацию коммерческий сервис, так как эмулятор не может полностью воспроизвести реальную компьютерную сеть. Кроме того, для полноценного понимания принципов и методов DDoS атаки необходим опыт работы с ней. Поэтому авторы анонимно договорились о проведении реальной DDoS атаки на специально подготовленный веб сервис. В процессе атаки был записан сетевой трафик, собрана статистика NetFlow и сделаны выводы об эффективности способов обнаружения и методов противодействия.

3.2. Обзор предшествующих работ

Множество работ посвящено обнаружению несанкционированных вторжений при помощи анализа данных протокола NetFlow. Достаточно полный обзор можно найти в статье [9]. Среди методов обнаружения вторжений можно отметить статистический подход [10,11], когда анализ информации о потоках с помощью простейших законов теории вероятности позволяет выявить источники атак. Статистический подход отличается простотой, даёт надежные результаты, но область применения ограничена хорошо изученными типами атак.

Неизвестные методы вторжений требуют для обнаружения более сложных подходов, таких как различные схемы машинного обучения [12]. Для обнаружения новых угроз используются

нейронные сети [13], генетические алгоритмы [14], опорные векторные машины [15] и т.д.

Наш подход [16] имеет отличительные особенности. Во-первых, обнаружение атаки базируется на отклонениях от распределения Зипфа [17]. Для этого анализируется информация об активных или завершённых потоках за некоторый промежуток времени. Минимальное время сбора статистики NetFlow может варьироваться от одной до пяти минут. При нормальном функционировании сети ранжированный список количества потоков, генерируемых уникальным IP адресом, представляет типичное распределение Зипфа [18].

Атакующие адреса могут быть идентифицированы по превышению установленного порогового значения для числа активных или завершённых потоков [16]. Во время атак число потоков возрастает многократно.

Однако этот метод не будет работать для атаки типа «UDP flood», при которой не меняются порты источника и назначения пакетов [19]. В этом случае число потоков с одного атакующего IP адреса будет небольшим, в то же время скорость потока может достигать десятков Мбит/с. Поэтому необходим дополнительный критерий для обнаружения атаки указанного типа.

Второе отличие предлагаемого подхода состоит в том, чтобы выделить пользовательское ядро. При атаке можно блокировать не ограниченное число обнаруженных атакующих адресов, а все адреса вне пользовательского ядра.

3.3. Выявление пользовательского ядра Интернет сервиса

Для выявления пользовательского ядра Интернет сервиса можно использовать любые журналы доступа, в которые вносятся IP адреса посетителей, либо данные NetFlow. В рассматриваемом в данной работе случае атака проводится на веб сервер популярного Интернет портала, поэтому в качестве источника IP адресов используются файлы журналов веб сервера *Nginx*, работающего в операционной системе Debian GNU/Linux. IP адреса посетителей заносятся в базу данных и в дальнейшем считаются принадлежащими пользовательскому ядру. После формирования ядра, пользователи, IP адреса которых не внесены в базу данных, считаются новыми.

На Рис. 3.1 изображён график измерения доли новых IP адресов η в ежесуточной аудитории. Под долей η понимается отношение количества IP адресов новых посетителей к их общему числу за каждые сутки.

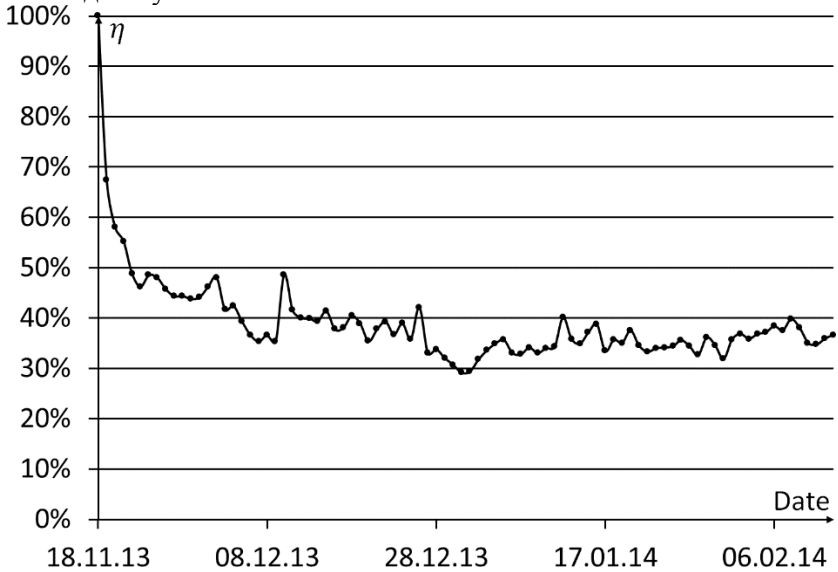

Рисунок 3.1 - График зависимости доли новых IP адресов за сутки от времени, прошедшего с начала измерений

Из графика видно, что через четыре-шесть недель с начала сбора статистики процент новых IP адресов, с которых происходят посещения сайта, стабилизируется в интервале от 30 до 40%.

Для того чтобы понять структуру аудитории необходимо построить ранжированные списки пользовательских IP адресов [18]. В качестве величины для упорядочивания следует выбрать частоту посещений R с каждого адреса, а пользовательские IP адреса расположить в порядке убывания данной частоты. В качестве ранга n будет использован порядковый номер IP адреса в убывающем списке, тогда функция $R(n)$ и будет искомым ранговым распределением.

Для того, чтобы понять природу поведения посетителей с новых адресов, необходимо построить два ранговых распределения.

Первое распределение должно содержать анализ суточной выборки IP адресов. Вторая выборка должна анализировать данные за более длительный срок, не менее четырех месяцев.

Для построения суточного ранжированного списка можно выбрать различные исходные статистические данные, например, журналы веб сервера или данные NetFlow.

Журналы доступа веб сервера Nginx позволяют получить зипфоподобное (Zipf like) распределение, изображенное на Рис. 3.2. Здесь $R(n)$ – это количество запросов с одного IP адреса, а n ранг этого запроса, т.е. порядковые номера IP адресов в списке по убыванию количества запросов.

Рисунок 3.2 - Распределение Зипфа для посещений, зафиксированных в журнале доступа

Условно график можно разделить на две части. Вначале идёт пологая часть, а затем резкий спад количества посещений с одного IP адреса. В исследуемом случае спад начинается с 55 запросов на IP адрес в сутки. Анализ содержимого запросов позволяет сделать вывод о том, что с IP адресов в пологой части делаются запросы ко всем документам с загружаемой HTML страницы, то есть закачиваются картинки, скрипты, стили и другие, подключаемые на странице, файлы. В части спада запрашиваются только HTML страницы, которые генерируются каждый раз заново, что говорит о

том, что остальные файлы уже были сохранены в кэше браузера пользователя, а значит, он ранее уже посещал данный сайт. Стоит отметить, что пользователи, запросившие от 1 до 3 файлов, как правило, вообще не заходили на сайт, а загрузили картинку с сайта через поисковую систему или обновили RSS ленту.

Версию кэширования подтверждает и аналогичное ранговое распределение, построенное с учётом запросов только HTML файлов и другого содержимого, которое не остаётся в кэше браузера пользователя (см. Рис. 3.3).

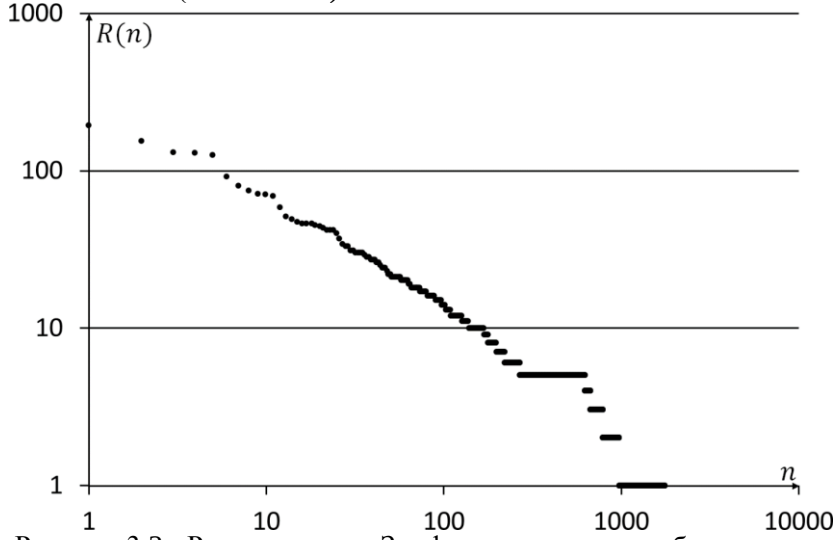

Рисунок 3.3 - Распределение Зипфа для запросов к обновляемым данным

Такое поведение свидетельствует о том, что пользователи из второй части графика с Рис. 3.2 ранее уже посещали сайт, но имели другой IP адрес. Как правило, такие пользователи подключаются к сети Интернет через сеть своего провайдера, который выдаёт ему IP адрес из своего блока динамического адресов. Указанные блоки адресов так же должны быть занесены в пользовательское ядро, так как вероятно пользователь сайта получит тот или иной адрес из этого блока в будущем.

Статистика NetFlow для новых IP адресов за сутки даёт похожий результат, изображенный на Рис. 3.4.

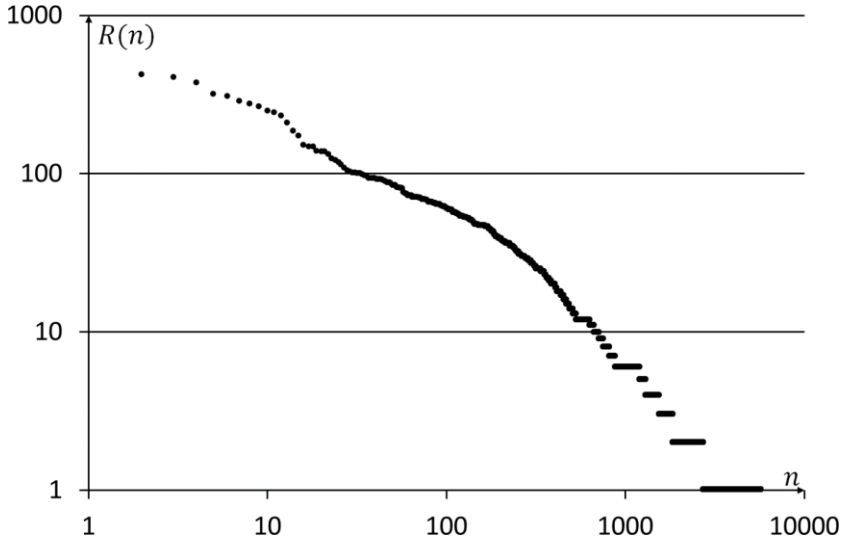

Рис. 4. Суточная статистика посещений для новых адресов по данным NetFlow.

На Рис. 3.5 изображена суточная статистика посещений по данным NetFlow для всех адресов, включая новые и встречавшиеся ранее. Этот график наиболее близок к прямой.

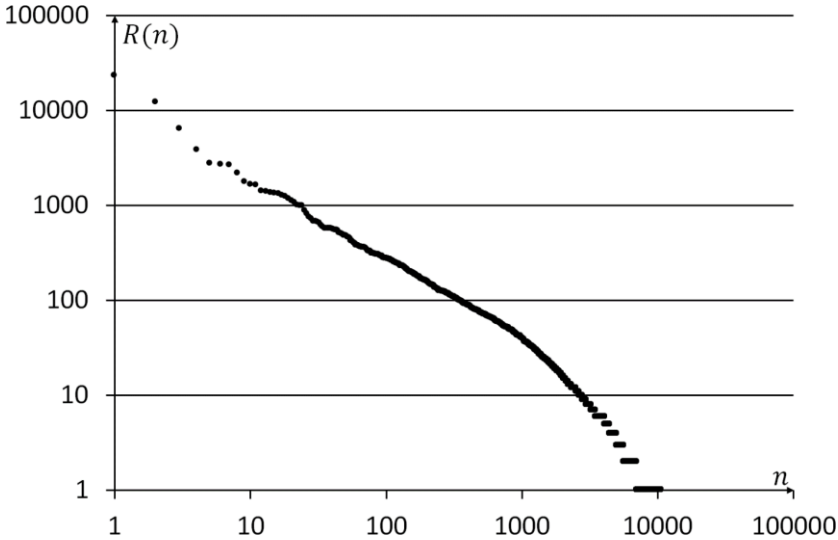

Рисунок 3.5 – Полная суточная статистика посещений по данным NetFlow

Для того чтобы понять происхождение новых IP адресов и почему их доля не уменьшается со временем была собрана и проанализирована статистика посещений за пять месяцев. В качестве величины $R(n)$ для упорядочивания данных было выбрано количество суток, в течении которых с исследуемого IP адреса поступали запросы к исследуемому сервису. Итоговый ранжированный график изображен на Рис. 3.6.

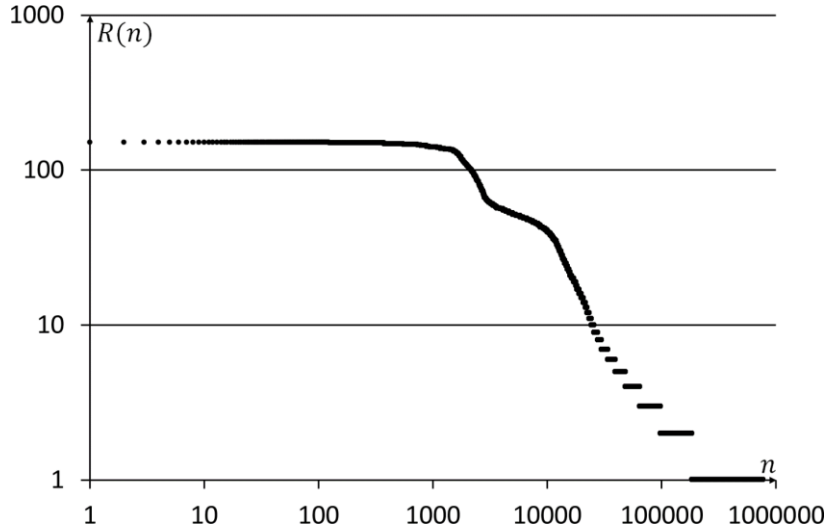

Рисунок 3.6 – Ранговое распределение посещений сайта за пять месяцев

Ранжированный список показывает, что большинство пользователей посещали сайт только в течение одних суток. Однократные посещения зафиксированы с более чем 590 тысяч адресов из 775 тысяч, что составляет 76.2% от общего количества адресов. Назовем IP адреса, с которых посещения фиксировались только в течение одних суток, случайными IP адресами. Сравнивая данные из списка случайных IP адресов со списком новых IP адресов за сутки, получается, что доля случайных IP адресов среди новых IP адресов составляет 91±5%. Этот факт объясняет большой процент новых IP адресов в суточной аудитории. Предсказать IP адреса таких посетителей невозможно.

3.4. Формирование списка постоянных посетителей

Исследование аудитории пользователей сервиса проведенное выше, позволяет сформировать список постоянных посетителей. За базу такого списка можно взять аудиторию за какой-то значительный промежуток времени (от двух недель, а лучше месяц). Прежде всего, из такого списка следует удалить все случайные адреса, то есть адреса, с которых посещали сайт только в течение одних суток.

Он будет не полон, так как часть IP адресов выбирается из блоков динамических адресов Интернет провайдеров. Для пополнения списка IP адресов был разработан специальный скрипт, в основе которого лежит нижеследующий алгоритм. В качестве исходных данных берётся список IP адресов, с которых производились запросы к сайту за время стабилизации количества новых IP адресов (четыре-шесть недель в исследуемом случае). Адреса разбиваются на группы по сетевой маске /24. Если группа заполнена адресами на 30%, то блок IP адресов по маске /24 заносится целиком в список IP адресов пользовательского ядра. Если группа заполнена меньше, чем на 30%, то она разбивается на две равные части по сетевой маске /25. Аналогичная операция по проверке заполнения и расширению ядра выполняется для полученных частей. Если какой-то из блоков адресов заполнен меньше порогового значения в 30%, то операция повторяется до сетевой маски /29 включительно.

Следует отметить, что разработанный скрипт по расширению пользовательской базы необходимо запускать до описанного в предыдущем разделе скрипта по удалению случайных посетителей. Адреса посетителей из подсетей, принадлежащих пользовательскому ядру, не считаются случайными и не удаляются при чистке.

Применение разработанного скрипта дополнения базы данных позволило увеличить число входящих в неё адресов с 273030 до 393273, то есть на 44%. На Рис. 3.7 изображён график зависимости процента новых IP адресов за сутки η от времени, пошедшего с начала измерений, с учетом дополнения базы данных. После расширения ядра доля новых адресов находится в интервале от 25 до 35%.

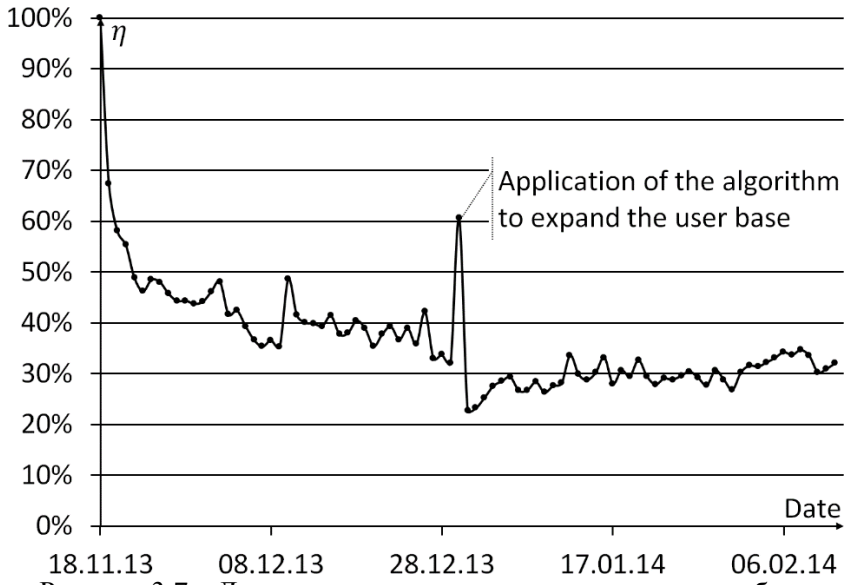

Рисунок 3.7 – Доля новых адресов с учетом расширения базы

При определении момента начала атаки, необходимо иметь в виду, что общее количество запросов различается как по дням недели, так и по времени суток, но эти показатели цикличны. Например, на Рис. 3.8 видно, что количество запросов с 2 до 5 часов ночи более чем в 5 раз ниже, чем количество запросов с 10 часов утра до 16 часов дня. Количество запросов в выходные дни в 2-3 раза ниже, чем в будни.

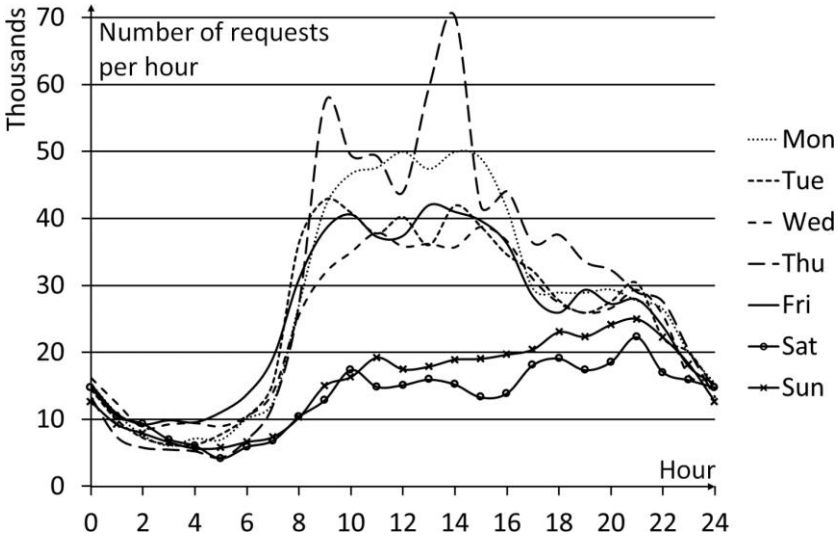

Рисунок 3.8 – Количество запросов по дням недели и часам

На Рис. 3.9 изображён типичный суточный график изменения числа завершённых потоков за рабочий день. Сбор данных о потоках происходит каждые пять минут. В случае начала реальной атаки число активных потоков возрастёт, как минимум, на порядок.

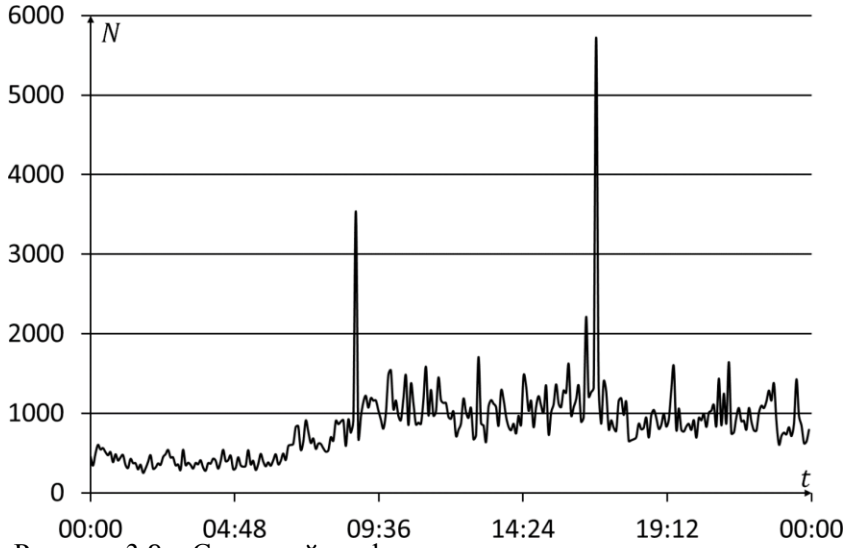

Рисунок 3.9 – Суточный график изменения количества завершившихся потоков

В заключение этого раздела необходимо установить, какой части пользователей будет отказано в обслуживании при установке фильтра, позволяющего обслуживать запросы только с IP адресов, входящих в пользовательское ядро. Из графика на Рис. 3.7 видно, что доля новых IP адресов стремится к пределу, который можно обозначить как η. Тогда за время t, потраченное на обнаружение и блокировку атаки, защищаемый сервис не сможет обслужить следующее количество запросов

$$K = \eta S t,$$

где S – среднее количество уникальных IP адресов за сутки. Время t также следует исчислять в сутках. Для времени t, равного 6 часам (0.25 суток) и $\eta = 32\%$ следует, что всего 8% от среднесуточной аудитории будет отказано в обслуживании.

3.5. Защитная инфраструктура и её испытания

Для проведения тестовых испытаний и проверки защиты в условиях реальной DDoS атаки была создана сетевая инфраструктура на базе веб хостинга, на который был перемещён популярный Интернет портал. Принципиальная схема сетевой инфраструктуры приведена на Рис. 3.10.

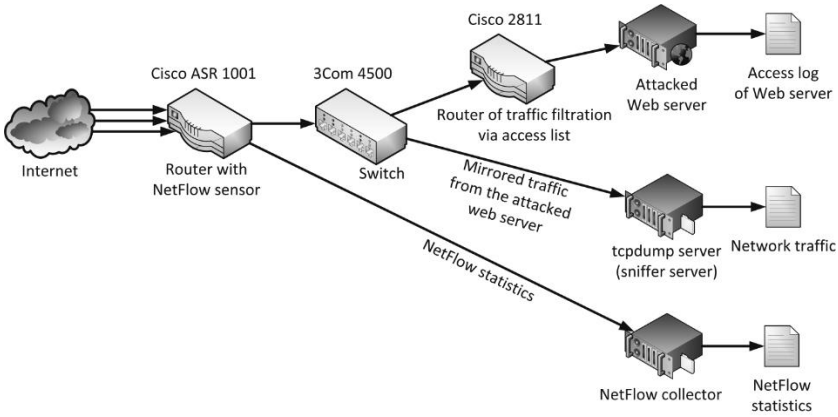

Рисунок 3.10 – Сетевая инфраструктура для проведения исследований.

Эта инфраструктура включает следующие элементы:

– NetFlow установлен на граничном маршрутизаторе Cisco ASR 1001;

– специальный скрипт на базе NetFlow, который выделяет адреса, генерирующие число потоков выше порогового значения и заносящее их в стоп лист для последующей блокировки;

– специальный скрипт, определяющий начало атаки по резкому росту числа активных потоков;

– дополнительный маршрутизатор Cisco 2811 перед атакуемым сервером, на котором устанавливались стоп листы;

– коммутатор 3Com 4500 используется для дублирования сетевого трафика с порта, идущего к веб серверу, на сервер, с установленным сетевым сниффером tcpdump [20], который сохраняет весь трафик в файл для последующего анализа;

– специально сформированный список постоянных посетителей, который активируется в момент начала атаки для ограничения посетителей атакуемого сайта.

Выделение для дублирования трафика отдельного устройства обусловлено тем, что необходимо собрать весь трафик во время атаки для последующего анализа, а потом уже попытаться защитить от неё веб сервер. При вводе в коммерческую эксплуатацию списки доступа должны загружаться на вышестоящем уровне у Интернет провайдера.

Перед проведением испытаний в условиях реальной атаки было проведено комплексное лабораторное испытание с участием 10 атакующих компьютеров, находящихся, как в сети предприятия, так и за её пределами.

Атака на количество запросов к веб серверу выполнялась с помощью Apache HTTP server benchmarking tool [21], Low Orbit Ion Cannon [22], BoNeSi [23]. UDP flood атака с помощью Low Orbit Ion Cannon и BoNeSi. В дополнение проводилось испытание скорости фильтрации IP адресов на Cisco 2811. Ни одна из атак не вывела из строя оборудование, а веб сервер отвечал на запросы пользователей. Стоит отметить, что испытания проводились на введённой в коммерческую эксплуатацию системе, не являющейся лабораторным стендом. По этой причине невозможно полноценно провести эмуляцию большого ботнета с реальным участием всего нескольких атакующих компьютеров.

Лабораторные испытания не могут заменить опыт, полученный при реальной атаке, поэтому авторы анонимно обратились с просьбой о проведении комбинированной DDoS атаки на описанный выше веб сервер. Статистика использования данного сервера собиралась в течение пяти месяцев и была приведена в предыдущем разделе статьи.

Реальный опыт отражения DDoS атаки кардинально поменял мнение авторов о типе и особенностях проведения атаки. Перед атакой планировалось осуществлять дистанционно мониторинг состояние оборудования, но первые минуты DDoS атаки показали, что делать это через атакуемый канал связи невозможно. Начало DDoS атаки сопровождалось резким ростом (более чем в сто раз) числа активных потоков, что было своевременно зафиксировано скриптами наблюдения. Затем в первые минуты DDoS атаки произошло переполнение внешних каналов связи, и веб сервер стал недоступен из внешней сети. Все остальные службы и сервера, находящиеся в данной серверной так же стали недоступны, удалённое управление было утеряно, несмотря на три внешних канала связи. Поэтому пришлось приезжать и брать управление из внутренней сети. Переполнение одного из внешних каналов демонстрирует следующий график, изображенный на Рис. 3.11.

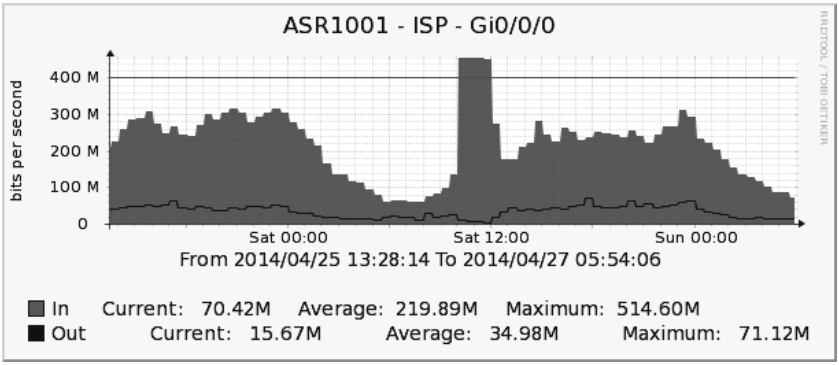

Рисунок 3.11 – График загрузки внешнего канала во время DDoS атаки

Сплошная линия показывает предельно допустимую нагрузку на канал от сервис провайдера. В течении всей атаки она была значительно превышена.

Выход из строя каналов связи произошёл из-за переполнения входящим UDP трафиком (DDoS атака типа «UDP flood»). Причем количество серверов, генерирующих этот трафик, было сравнительно небольшим, их насчитывалось не более 200. Приблизительно половина этих серверов, практически не меняла порты источника и назначения, другая половина делала это регулярно.

При атаке применялись UDP пакеты двух типов. Первый тип это UDP пакеты минимальной длины (см. Рис. 3.12). Эти пакеты содержат один символ, который повторяется во всех пакетах. Второй тип представляет собой UDP пакеты максимальной длины. Эти пакеты заполнены случайными данными (см. Рис. 3.13). Все пакеты помечены, как фрагменты для их последующего объединения сервером в один большой пакет (см. Рис. 3.14).

Time	Source	Destination	Protocol	Length	Info
0.000000	193.200.38.21	91.222.129.218	UDP	60	Source port: 53186 Destination port: 29578
0.000004	123.231.6.14	91.222.129.218	UDP	60	Source port: 33509 Destination port: 21545
0.000014	123.237.252.198	91.222.129.218	UDP	60	Source port: 43149 Destination port: 64098
0.000018	86.34.167.242	91.222.129.218	UDP	60	Source port: 50633 Destination port: 34012
0.000027	195.138.198.208	91.222.129.218	UDP	60	Source port: 50832 Destination port: 10760
0.000031	190.151.39.254	91.222.129.218	UDP	60	Source port: 28755 Destination port: 52633
0.000041	184.22.62.140	91.222.129.218	UDP	60	Source port: 47576 Destination port: stx
0.000044	86.34.167.242	91.222.129.218	UDP	60	Source port: 50633 Destination port: 34012
0.000054	195.138.198.208	91.222.129.218	UDP	60	Source port: 50832 Destination port: 10760
0.000058	190.151.39.254	91.222.129.218	UDP	60	Source port: 28755 Destination port: 52633
0.000067	202.129.38.5	91.222.129.218	UDP	60	Source port: 46313 Destination port: 52478
0.000071	86.34.167.242	91.222.129.218	UDP	60	Source port: 50633 Destination port: 34012
0.000081	195.138.198.208	91.222.129.218	UDP	60	Source port: 50832 Destination port: 10760
0.000085	83.172.21.22	91.222.129.218	UDP	60	Source port: 45002 Destination port: openvms-
0.000095	202.129.38.5	91.222.129.218	UDP	60	Source port: 46313 Destination port: 52478
0.000099	86.34.167.242	91.222.129.218	UDP	60	Source port: 50633 Destination port: 34012
0.000108	195.138.198.208	91.222.129.218	UDP	60	Source port: 50832 Destination port: 10760
0.000111	83.172.21.22	91.222.129.218	UDP	60	Source port: 45002 Destination port: openvms-
0.000121	202.129.38.5	91.222.129.218	UDP	60	Source port: 46313 Destination port: 52478
0.000125	86.34.167.242	91.222.129.218	UDP	60	Source port: 50633 Destination port: 34012
0.000134	195.138.198.208	91.222.129.218	UDP	60	Source port: 50832 Destination port: 10760
0.000138	203.140.65.78	91.222.129.218	UDP	60	Source port: 64852 Destination port: 59538
0.000148	188.163.245.82	91.222.129.218	UDP	60	Source port: 50917 Destination port: 29088
0.000152	89.175.183.208	91.222.129.218	UDP	60	Source port: 53611 Destination port: 8778
0.000162	193.200.173.110	91.222.129.218	UDP	60	Source port: 56732 Destination port: 59330
0.000166	203.140.65.78	91.222.129.218	UDP	60	Source port: 64852 Destination port: 59538
0.000175	122.103.236.204	91.222.129.218	UDP	60	Source port: 45982 Destination port: 8110
0.000179	89.175.183.208	91.222.129.218	UDP	60	Source port: 53611 Destination port: 8778
0.000188	62.75.161.46	91.222.129.218	UDP	60	Source port: 35388 Destination port: 11265

Рисунок 3.12 – UDP пакеты минимальной длины

```
⊞ Frame 2975: 754 bytes on wire (6032 bits), 754 bytes captured (6032 bits)
⊞ Ethernet II, Src: Ibm_f0:2a:a8 (00:21:5e:f0:2a:a8), Dst: Cisco_43:3d:e0 (00:1b:54:43:3d:e0)
⊟ Internet Protocol Version 4, Src: 119.6.254.144 (119.6.254.144), Dst: 91.222.129.218 (91.222.129.218)
    Version: 4
    Header length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 740
    Identification: 0x570b (22283)
  ⊞ Flags: 0x00
    Fragment offset: 7400
    Time to live: 108
    Protocol: UDP (17)
  ⊞ Header checksum: 0x9e11 [correct]
    Source: 119.6.254.144 (119.6.254.144)
    Destination: 91.222.129.218 (91.222.129.218)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
  ⊞ [6 IPv4 Fragments (8120 bytes): #2970(1480), #2971(1480), #2972(1480), #2973(1480), #2974(1480), #2975(720)]
⊟ User Datagram Protocol, Src Port: 1028 (1028), Dst Port: 5842 (5842)
    Source port: 1028 (1028)
    Destination port: 5842 (5842)
    Length: 8120
  ⊞ Checksum: 0x8790 [validation disabled]
⊟ Data (8112 bytes)
    Data: 1b075550055b9018ecee2d6675a27f8c1058feb030430163...
    [Length: 8112]
```

```
0000  00 1b 54 43 3d e0 00 21  5e f0 2a a8 08 00 45 00   ..TC=..! ^.*...E.
0010  02 e4 57 0b 03 9d 6c 11  9e 11 77 06 fe 90 5b de   ..W...l. ..w...[.
0020  81 da 09 62 59 73 03 08  ee d6 21 2a 4e 36 f6 cb   ...bYs.. ..!*N6..
0030  f6 b0 fb d8 2e a7 c7 09  c2 82 d7 a4 80 73 28 67   ........ .....s(g
0040  c4 bd 9a 8e e4 cd 64 9e  e9 70 17 34 21 a2 df ef   ......d. .p.4!...
0050  dd ab 1d 96 10 73 00 d0  5b d4 8a 97 b3 cc eb ab   .....s.. [.......
0060  b5 b2 9d ea 2e 68 36 5c  ea 6c ee 35 dc 72 3b e5   .....h6\ .l.5.r;.
0070  7e 9b db fd f7 14 59 68  31 3a 6c 18 db 00 63 03   ~.....Yh 1:l...c.
0080  5b bd 52 b3 c5 05 39 0e  8b 76 1e e7 98 06 db d4   [.R...9. .v......
0090  e7 04 9a 07 3b f0 bd 32  49 9e 3b b3 98 be b7 2b   ....;..2 I.;....+
00a0  d0 af 82 4c b6 40 2b b1  e7 ef 6e 55 d5 eb 20 68   ...L.@+. ..nU.. h
00b0  59 f3 7d da a7 1e ce 03  7c 82 97 c9 7f d1 4a 7e   Y.}..... |.....J~
00c0  32 69 80 98 8f 66 52 83  12 a0 7d 3d 2d 51 cc 83   2i...fR. ..}=-Q..
00d0  52 be cc 11 77 75 30 52  e2 81 0a d2 2d 8c d8 d9   R...wu0R ....-...
```

Рисунок 3.13 – UDP пакеты со случайным заполнением данными

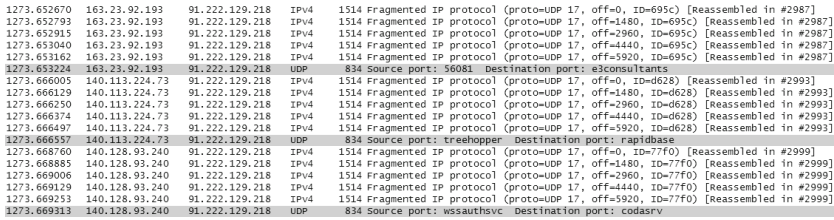

Рисунок 3.14 – Фрагментированные UDP пакеты

Небольшое количество атакующих серверов компенсировалось суммарной скоростью UDP потоков, генерируемых атакующим каждым адресом. С ряда адресов эта скорость достигала 60 Мбит/с и могла бы быть увеличена, если бы не ограничения, наложенные нашим провайдером на внешний канал. Поверка месторасположения атакующих серверов показала, что большинство их расположено в США, хотя переписка с руководством ботнета происходила на русском. По заверениям руководства ботнета его мощность в атаке на нас была использована только на 2%. При этом пострадал только наш веб хостинг с его внешними каналами шириной порядка 1 Гбит/с. Каналы нашего внешнего провайдера суммарной мощностью не меньше 100 Гбит/с не пострадали.

К сожалению, наш хостинг не имел соглашения об ограничении входящего трафика с провайдерами, простейший запрет UDP трафика на атакуемый сервер сразу бы помог решить большинство проблем.

TCP запросы (DDoS атака типа «TCP flood») тоже участвовали в DDoS атаке. Число атакующих серверов было порядка 1500. При атаке применялись TCP запросы двух типов. Первый тип – это запросы файлов с веб сервера, имитирующие действия пользователей. Второй тип представлен множеством SYN/ACK пакетов минимального размера – по всей видимости, это «TCP SYN flood» DDoS атака. Но существенного ущерба эти атаки не нанесли ввиду переполнения канала и активации алгоритмов ограничения запросов с одного IP адреса на веб сервере.

Анализ атаки на уровне потоков показал, что начало атаки сопровождалось резким ростом числа активных потоков во внешнем канале веб хостинга. Число завершенных потоков, как упоминалось выше, возросло более чем на два порядка. Этот рост был сразу зафиксирован мониторинговой системой. Отдельные атакующие

IP адреса легко определяются по количеству генерируемых потоков, как активных, так и завершенных. На Рис. 3.15 приведен ранжированный список адресов по количеству сгенерированных потоков. Верхний график описывает момент атаки, на нижнем графике приведено типичное распределение при отсутствии атаки.

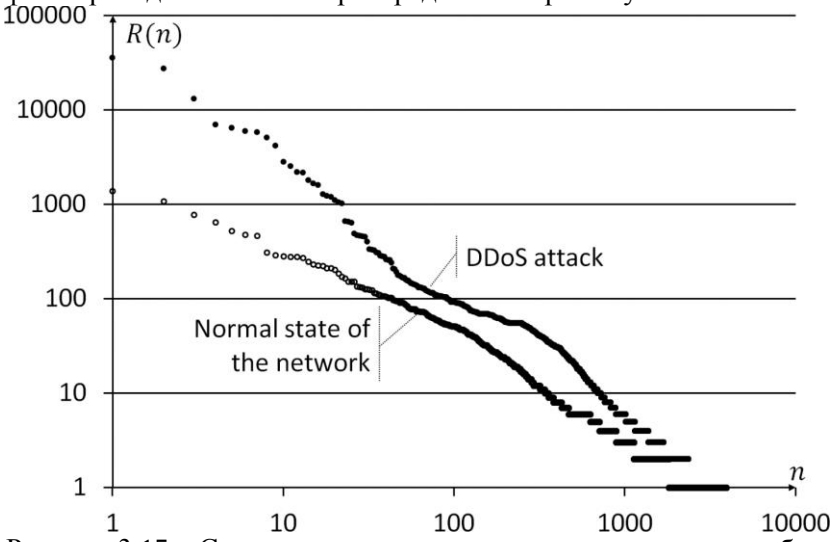

Рисунок 3.15 – Сравнение числа потоков во время атаки и в обычном состоянии

Сравнение графиков с Рис. 3.15 позволяет сформулировать критерий для определения принадлежности IP адреса к ботнету. Все адреса, расположенные выше, чем самый популярный сервер в обычном состоянии сети и не принадлежащие к пользовательскому ядру, должны быть отнесены к ботнету [16]. Для полного выявления атакующих адресов необходимо построить ранжированные распределения для входящего UDP, ICMP и TCP трафика, генерируемого с единичного IP адреса в момент атаки. Обычное состояние сети используется для определения порога отсечения, как это было сделано в работе [16]. Порог отсечения должен быть определен для каждого сервиса и типа трафика заранее, причем раз в полгода эти значения необходимо пересчитывать.

Применение двух независимых критериев по числу потоков и объему входящего трафика (UDP, ICMP или TCP) позволяет оперативно (в течение 5 минут) составить список атакующих адресов для блокировки на фильтрующем оборудовании.

3.6. Выводы

Для организации защиты от несанкционированного воздействия на сервисы компьютерных сетей было проведено исследование пользовательской аудитории крупного Интернет портала. Изучались вопросы выделения постоянной аудитории, или пользовательского ядра, и расширения этой базы для учёта блоков динамических адресов провайдеров.

В работе предлагаются методы составления разрешительного списка, который содержит IP адреса пользовательского ядра, и запретительного, который включает в себя атакующие IP адреса в момент DDoS атаки.

Для проверки алгоритмов составления запретительного списка была проведена тестовая атака с использованием реального ботнета, что позволило сделать следующие выводы.

Устойчивая работа сетевой инфраструктуры серверной невозможна без наличия, как минимум, двух внешних Интернет каналов связи по 10 Гбит/с. Каналы порядка нескольких Гбит/с не могут обеспечить отказоустойчивость при DDoS атаке.

Список блокированных IP адресов должен устанавливаться у провайдеров верхнего уровня. Для этого должно быть заключено соглашение с каждым из провайдеров, а процесс передачи стоп листа должен быть автоматизирован и осуществляться без вмешательства администраторов.

Идеально, чтобы стоп лист транслировался к провайдерам второго уровня, то есть к провайдерам провайдеров. Такая защита позволит избежать трудностей защиты от подавляющего большинства существующих ботнетов. По нашим наблюдениям, перенос защиты на третий уровень провайдеров позволит полностью блокировать атаки.

Наибольшую опасность представляет атака UDP flood, нацеленная на переполнение каналов доступа. Поскольку DDoS атака чаще всего ведётся на один или несколько адресов, то полное ограничение UDP трафика у провайдера верхнего уровня на атакуемые

адреса поможет избежать переполнения каналов связи. Требуется рассмотрение вопроса об ограничении скорости UDP потоков с одиночного IP адреса. Второй тип атакующих IP адресов, ведущих атаку с помощью TCP запросов, легко идентифицируется с помощью критерия по числу потоков. Одновременное применение указанных методов даёт возможность выявить IP адреса атакующих серверов (ботов) в течение 30 минут.

Основным организационным мероприятием по отражению атаки является налаживание взаимодействия с провайдерами. С ними необходимо иметь соглашение об оперативной передаче листов доступа. Это может быть разрешительный список, он большого размера и должен загружаться заранее.

После начала атаки формируется стоп лист, который также может быть двух типов. Первый тип ограничивает UDP пакеты на атакуемые адреса. Это позволяет избежать переполнения канала. Второй список действует с начала атаки и разрешает доступ к сайту только постоянной аудитории сайта, сформированной заранее. После установления списка атакующих адресов необходимо заменить разрешающий список на запрещающий. Для его формирования достаточно 30 минут при наличии заранее установленной защитной инфраструктуры.

Литература

[1] Mirkovic J., Reiher P. A taxonomy of DDoS attack and DDoS defense mechanisms //ACM SIGCOMM Computer Communication Review. – 2004. – ISSN: 0146-4833. – DOI: 10.1145/997150.997156. – Т. 34. – №. 2. – С. 39-53.

[2] Douligeris C., Mitrokotsa A. DDoS attacks and defense mechanisms: classification and state-of-the-art //Computer Networks. – 2004. – ISSN: 1389-1286. – DOI: 10.1016/j.comnet.2003.10.003. – Т. 44. – №. 5. – С. 643-666.

[3] Singh S., Gyanchandani M. Analysis of Botnet behavior using Queuing theory //International Journal of Computer Science & Communication. – 2010. – ISSN: 0973-7391. – Т. 1. – №. 2. – С. 239-241.

[4] Stanton J. M., Stam K. R., Mastrangelo P., Jolton, J. Analysis of end user security behaviors //Computers & Security. – 2005. – ISSN: 0167-

4048. – DOI: 10.1016/j.cose.2004.07.001. – Т. 24. – №. 2. – С. 124-133.
[5] Hochheiser, H. Shneiderman, B. Understanding patterns of user visits to web sites: interactive starfield visualizations of WWW log data //Proceedings of the ASIST Annual Meeting – 1999. – ISSN 0044-7870.
[6] Chen J., Nairn R., Nelson L., Bernstein M., Chi E. Short and tweet: experiments on recommending content from information streams //Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. – ACM, 2010. – ISBN: 978-1-60558-929-9. – DOI:10.1145/1753326.1753503. – С. 1185-1194.
[7] Lehmann J., Lalmas M., Yom-Tov E., Dupret G. Models of user engagement //User Modeling, Adaptation, and Personalization. – Springer Berlin Heidelberg, 2012. – ISSN: 0302-9743. – ISBN: 978-3-642-31453-7. – DOI: 10.1007/978-3-642-31454-4_14. – С. 164-175.
[8] Ma J., Saul L. K., Savage, S. Voelker G. M. Beyond blacklists: learning to detect malicious web sites from suspicious URLs //Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining. – ACM, 2009. – ISBN: 978-1-60558-495-9. – DOI: 10.1145/1557019.1557153. – С. 1245-1254.
[9] Li B., Springer J., Bebis G., Hadi Gunes M. A survey of network flow applications //Journal of Network and Computer Applications. – 2013. – ISSN: 1084-8045. – DOI: 10.1016/j.jnca.2012.12.020. – Т. 36. – №. 2. – С. 567-581.
[10] Proto A., Alexandre L. A., Batista M. L., Oliveira I. L., Cansian,A. M. Statistical model applied to netflow for network intrusion detection //Transactions on computational science XI. – Springer Berlin Heidelberg, 2010. – ISBN: 978-3-642-17696-8. – ISSN: 0302-9743. – DOI: 10.1007/978-3-642-17697-5_9. – С. 179-191.
[11] Sawaya Y., Kubota A., Miyake Y. Detection of attackers in services using anomalous host behavior based on traffic flow statistics //Applications and the Internet (SAINT), 2011 IEEE/IPSJ 11th International Symposium on. – IEEE, 2011. – ISBN: 978-1-4577-0531-1. – DOI: 10.1109/SAINT.2011.68. – С. 353-359.
[12] Sommer R., Paxson V. Outside the closed world: On using machine learning for network intrusion detection //Security and Privacy (SP), 2010 IEEE Symposium on. – IEEE, 2010. – ISSN: 1081-6011. – ISBN: 978-1-4244-6894-2.
– DOI: 10.1109/SP.2010.25. – С. 305-316.

[13] Corchado E., Herrero Á. Neural visualization of network traffic data for intrusion detection //Applied Soft Computing. – 2011. – ISSN: 1568-4946. – DOI: 10.1016/j.asoc.2010.07.002. – Т. 11. – №. 2. – С. 2042-2056.
[14] Martinez L. P., Espitia H. E. Proposal and adjustment of a Colombian migration model for the Pacific region //Instrumentation and Measurement, Sensor Network and Automation (IMSNA), 2013 2nd International Symposium on. – IEEE, 2013. – DOI: 10.1109/IMSNA.2013.6742812. – С. 37-41.
[15] Zhao G., Ji C., Xu C. Survey of Techniques for Internet Traffic Identification [J] //Journal of Chinese Computer Systems. – 2010. – ISSN: 1000-1220. – Т. 8. – С. 010.
[16] Sukhov A. M., Sidelnikov D. I., Platonov A. P., Strizhov M. V., Galtsev A. A. Active flows in diagnostic of troubleshooting on backbone links //Journal of High Speed Networks. – 2011. – ISSN: 0926-6801. – DOI: 10.3233/JHS-2011-0447. – Т. 18. – №. 1. – С. 69-81.
[17] Zipf G. K. Relative frequency as a determinant of phonetic change //Harvard Studies in Classical Philology. – 1929. – С. 1-95.
[18] Крашаков С. А., Теслюк А. Б., Щур Л. Н. Об универсальности рангового распределения популярности веб-серверов //Вестник РФФИ. – 2004. – ISSN: 1605-8070.– С. 46-66.
[19] Chen Y., Hwang K., Ku W. S. Collaborative detection of DDoS attacks over multiple network domains //Parallel and Distributed Systems, IEEE Transactions on. – 2007. – ISSN: 1045-9219. – DOI: 10.1109/TPDS.2007.1111. – Т. 18. – №. 12. – С. 1649-1662.
[20] Luis M. G. TCPDUMP/LIBPCAP public repository //Online document, www.tcpdump.org. – 2009.
[21] ab – Apache HTTP server benchmarking tool – Apache HTTP Server Version 2.2 //The Apache Software Foundation. http://httpd.apache.org/docs/2.2/programs/ab.html
[22] LOIC | Free Security & Utilities software downloads at SourceForge.net //Dice Holdings, Inc. http://sourceforge.net/projects/loic/
[23] bonesi - BoNeSi - the DDoS Botnet Simulator //Google Project Hostingрон. https://code.google.com/p/bonesi/

4. ПРАКТИКА ВНЕДРЕНИЯ МЕТОДОВ ОБНАРУЖЕНИЯ DDOS АТАК

4.1. Введение в систему журналирования ОС Linux

Все примеры и описания в данной главе выполнены с использованием Debian GNU/Linux версии 7.6.0. В данной работе приведён обзор файлов журналов операционной системы Linux.

Каждый системный администратор, сталкиваясь с той или иной проблемой в операционной системе, первым делом должен отследить последовательность операций, которая приводит к сбоям. Это позволяет выявить в каком демоне, модуле или ядре происходят действия, приводящие к проблеме, а часто получить информацию по их разрешению.

В операционной системе Linux для этих целей ведутся файлы журналов. Они, как правило, расположены в директории /var/log и её поддиректориях, а сами файлы имеют расширение *.log – поэтому их часто называют лог-файлами или логами. Список основных файлов журналов и их предназначение представлен в таблице 4.1.

Таблица 4.1 – Основные файлы журналов Debian GNU/Linux и их предназначение.

Файл	**Предназначение**
auth.log	Информация по аутентификации пользователей в системе. Эта информация должна быть закрыта от доступа обычных пользователей.
boot.log	Сообщения при запуске системы. Содержит сообщения, полученные с момента запуска системы до её загрузки и вывода приглашения пользователя.
cron.log	Сообщения от планировщика задач.
daemon.log	Сообщения, поступающие от демонов.
debug	Любые сообщения с отладочной информацией.
dmesg	Содержит информацию из циклического буфера сообщений ядра. В начале загрузки системы в него записываются сообщения об аппаратном обеспечении, процессе обнаружения оборудования и запуску модулей ядра. При заполнении бу-

	фера новыми сообщениями старые будут удаляться. Содержимое буфера можно посмотреть командой dmesg.
dpkg.log	Содержит сообщения от системы управления пакетами в системах на основе Debian GNU/Linux.
kern.log	Сообщения от ядра Linux.
lastlog	Содержит время прошлого входа пользователей в систему после запуска. Это не обычный текстовый файл и для его чтения используется ко манда lastlog.
lpr.log	Сообщения от демона печати.
mail.log	Все сообщения от почтовой системы.
messages	Все информационные сообщения и предупреждения, кроме сообщений аутентификации, планировщика задач и почтовой системы.
syslog	Все сообщения, кроме сообщений аутентификации пользователей.
user.log	Все сообщения уровня пользователя.

Доступ к этим файлам, как правило, осуществляется любым удобным текстовым редактором.

nano /var/log/messages	Открывает файл в текстовом редакторе.
cat /var/log/messages	Выводит на консоль всё содержимое файла. Плохой вариант, для просмотра файла и удобен только при совместном использовании с фильтрами grep и т.п.
tail –n 15 /var/log/messages	Удобный вариант, если нужно вывести только несколько последних сообщений. Опция -n задаёт количество выводимых сообщений (в строках). По умолчанию выводятся 10 последних строк.
less /var/log/messages	Просмотр файла. Для справки нужно нажать клавишу h, для выхода из

	программы клавишу q, shift+f обеспечивает автоматическую прокрутку отображения файла при его обновлении, ctrl+c возврат к ручному режиму прокрутки.

Помните, что большинство журналов доступны для чтения только привилегированным пользователям! Для получения привилегий воспользуйтесь командами su или sudo.

Для ведения журналов в операционной системе Debian GNU/Linux используется демон rsyslogd. Его базовая настройка выполняется в файле /etc/rsyslog.conf, а дополнительные (пользовательские) правила размещаются в файлах, находящихся в директории /etc/rsyslog.d. Указанный демон имеет развитую систему правил, благодаря которой каждое пришедшее сообщение может быть классифицировано и сохранено в том или и ином файле журнала. Более подробно узнать о том, как это делается можно из документации [1].

Не смотря на все плюсы rsyslogd, исторически так сложилось, что высоконагруженные приложения, которые обязаны сохранять всю без исключения информацию о действиях пользователя, ведут файлы журналов самостоятельно. Дело в том, что rsyslogd и его предшественник syslogd принимают сообщения, используя протокол UDP, который не гарантирует доставку сообщений, а значит, в файлах журналов могут быть сохранены не все события.

Для определения ядра пользовательской аудитории веб-сайта в целях противодействия DDoS-атакам необходимо собрать информацию по пользователям, их IP-адресам и по тому, когда и какую информацию они просматривают на сайте. Для этих целей удобно использовать файлы журналов веб-сервера.

На сегодняшний день на высоконагруженных веб-сайтах на запросы пользователей в первую очередь отвечает веб-сервер nginx. Такой приоритет ему отдан благодаря его лёгкости, но в тоже время функциональности и отказоустойчивости. После получения запроса от пользователя nginx принимает решение в зависимости от настроек ответить ли пользователю самостоятельно (например, переслать существующий файл), передать запрос для дальнейшей

обработки в другую программу (например, в apache с модулем php), поставить запрос в очередь, либо прервать его выполнение. Таким образом, можно быть уверенным, что все HTTP запросы к веб-сайту будут отображены в журналах доступа nginx.

Как правило, размещение файлов-журналов nginx настраивается в конфигурационных файлах каждого узла в папке /etc/nginx/sites-available. За это отвечают параметры access_log и error_log в секции server.

- ✓ access_log – путь к журналу доступа, куда пишутся все выполненные запросы и IP-адреса клиентов.
- ✓ error_log – путь к журналу ошибок, куда пишутся все запросы, завершённые с ошибкой и IP-адреса клиентов.

Nginx ведёт файлы журналов самостоятельно, без использования демона rsyslogd, так как в случае большого количества запросов (например, в случае DDoS-атаки) rsyslogd не сможет гарантировать запись всех обращений пользователей в журнал, что приведёт к проблемам в безопасности системы.

Если системный администратор не предусмотрел иного размещения файлов-логов, то журналы доступа хранятся в /var/log/nginx/example.com-access.log, а ошибок в /var/log/nginx/example.com-error.log, где example.com это название узла, для которого сохраняется журнал в этом файле. Как правило, для каждого узла используется свой файл журнала. Иногда системные администраторы перемещают соответствующие файлы журналов узла в подпапку пользователя на виртуальном хостинге, чтобы дать доступ пользователю к журналам узла, за который он отвечает по протоколу FTP и т.п.

Необходимо понимать, что директория, в которой хранятся файлы журналов имеет конечный размер и рано или поздно место в ней закончится, если не удалять старые файлы. Для этих целей в Debian GNU/Linux предоставляется утилита logrotate. Это мощное средство, позволяющее задать для разных типов журналов различный период ротации (удаление старых файлов и создание новых), сжатие и удержание в течение определённого времени старых файлов журналов.

Глобальные настройки logrotate хранятся в файле

/etc/logrotate.conf, но часто они конкретизированы для определённых журналов в /etc/logrotate.d. Например, по умолчанию файл /etc/logrotate.d/nginx выглядит следующим образом:

```
/var/log/nginx/*.log { #фильтр файлов, к которым применяются правила
        daily #ежедневная ротация файлов журнала
        missingok #в случае отсутствия файла журнала перейти к обработке следующего не выдавая сообщения об ошибке
        rotate 52 #файл журнала будет удалён через 52 ротации
        compress #старые версии файлов журнала будут сжаты (по умолчанию gzip)
        delaycompress #отложить сжатие предыдущего файла журнала до следующего циклического сдвига
        notifempty #не сдвигать журнал, если он пуст
        create 0640 www-data adm #непосредственно после обращения (перед выполнением скрипта postrotate) создать файл журнала (с тем же именем, что и только что сдвинутый журнал), задать режим доступа к файлу 0640, владельца www-data и группу adm
        sharedscripts #указывает, что скрипты prerotate и postrotate будут выполняться только один раз, вне зависимости от количества файлов, подходящих под шаблон /var/log/nginx/*.log
        prerotate #нижеследующий скрипт будет выполнен перед ротацией журнала и только в случае, если журнал действительно будет сдвинут
                #выполняет файлы в директории httpd-prerotate в случае существования (по умолчанию отсутствует)
                if [ -d /etc/logrotate.d/httpd-prerotate ]; then \
                        run-parts /etc/logrotate.d/httpd-prerotate; \
                fi; \
        endscript
        postrotate #нижеследующий скрипт будет выполнен после ротации журнала и только в случае, если журнал действительно будет сдвинут
                #отправляет демону nginx сигнал на ротацию логов
                [ ! -f /var/run/nginx.pid ] || kill -USR1 `cat /var/run/nginx.pid`
        endscript
}
```

Приведённая выше конфигурация говорит о том, что файлы в директории /var/log/nginx с расширением *.log будут ротироваться раз в сутки. При этом к имени файла будет добавляться номер его ротации, например example.com-access.log.1, а на его месте будет создан новый файл с именем example.com-access.log. Начиная со второй ротации, файл будет сжат и к его первоначальному имени помимо номера ротации будет добавлено расширение архива, например example.com-access.log.2.gz. Такое поведение обусловлено механизмом inode в файловой системе Linux, при которой имя файла используется лишь для поиска соответствующей inode в файловой системе. В дальнейшем доступ к файлу происходит именно по inode вне зависимости от имени файла, его размещения и даже существования, так как можно отправить команду удаления файла, которая удалит имя файла, но фактически он будет доступен, пока есть программы, которые держат открытыми inode этого файла. То есть сначала logrotate изменяет имя файла журнала, затем создаёт новый файл со старым именем и отправляет уведомление о ротации демону nginx, который в свою очередь имеет возможность завершить все необходимые операции ввода-вывода со старым файлом и только после этого начать запись в новый файл. В связи с тем, что у logrotate нет информации о том, в какой момент nginx освободит старый файл, архивирование при первой ротации не производится. При следующей ротации файл example.com-access.log.2.gz будет переименован в example.com-access.log.3.gz и так далее до 52. При ротации файл example.com-access.log.52.gz будет удалён.

Более подробную информацию по настройкам logrotate можно получить по команде man logrotate или в русскоязычном варианте в источнике [2].

Для формирования пользовательского ядра аудитории необходимо собрать, а затем обработать файлы журналов веб-сервера nginx минимум за месяц. Конечно, такой длительный рутинный процесс нужно автоматизировать. В данном случае можно либо поменять настройки logrotate таким образом, чтобы файлы логов не удалялись более длительное время и собирались без сжатия. Для этого нужно удалить опции compress и delaycompress для отмены сжатия, а также notifempty – чтобы ежедневно иметь новый файл,

даже если он пустой. Rotate нужно установить в значение 100 и более, но необходимо учесть, что слишком большое значение приведёт к замедление выполнения logrotate и лишней нагрузке на файловую систему, так как при каждой ротации будут переименоваться большое количество файлов.

Более универсальным решением будет оставить настройки ротации журнала nginx по умолчанию, а вместо этого создать директорию /etc/logrotate.d/httpd-prerotate и в ней файл nginx следующего содержания:

```
cp    /var/log/nginx/example.com-access.log.1    /var/ar-
chive/`date +\%F`
```

Запуск этого скрипта задан в файле /etc/logrotate.d/nginx (ранее он был рассмотрен). По умолчанию он выполняется один раз перед ротацией журналов. Скрипт состоит всего из одной строчки, написанной на bash, и производит копирование файла /var/log/nginx/example.com-access.log.1 в файл с именем /var/archive/`date +\%F`. Последняя часть команды взята в апострофы и при интерпретации bash будет заменена результатом выполнения команды date +\%F, то есть текущей датой в формате гггг-мм-дд. Директорию /var/archive необходимо создать вручную и позаботиться, чтобы в ней всегда хватало места для накопления архивов журнала.

Как правило, по умолчанию любой веб-сервер ведёт файлы журналов доступа в стандартизованном NCSA формате Common Log Format [3] или в более актуальном Extended Common Log Format (Combined Log Format) [4]. Они представляют собой несколько последовательно записанных через запятую параметров:

```
remotehost rfc931 authuser [date] "request" status bytes
referrer agent
```

remotehost

IP-адрес или DNS-имя клиента, сделавшего запрос.

rfc931

Имя пользователя удалённой машины, с которой отправлен запрос. Современные веб-сервера не запрашивают эту информацию, поэтому данное поле всегда “-”.

authuser

Имя пользователя при включенной HTTP-аутентификации на веб-сервере. Как правило, на общедоступных серверах HTTP-аутентификация не производится и это поле задано в “-”.

[date]

Дата и время получения запроса.

"request"

Первая строка заголовка запроса, полученная от клиента.

status

Статус ответа на запрос.

bytes

Размер переданного документа в байтах.

Следующие два параметра записываются только при использовании Extended Common Log Format. Эта информация передаётся браузером пользователя и ей нельзя полностью доверять.

referrer

Страница, с которой производится переход.

agent

Информация о браузере пользователя, его операционной системе и некоторых дополнительных настройках программного окружения.

В настоящее время по умолчанию nginx настроен на ведение журнала в ECLF формате. Пример нескольких записей ECLF из файла журнала nginx:

```
127.0.0.1 - frank [09/Aug/2014:06:25:02 +0400] "GET
/apache_pb.gif HTTP/1.0" 200 2326 "http://www.exam-
ple.com/start.html" "Mozilla/4.08 [en] (Win98; I ;Nav)"
31.184.219.88 - - [09/Aug/2014:06:25:03 +0400] "GET
/groups/19541 HTTP/1.0" 200 4334 "-" "Mozilla/5.0 (Mac-
intosh; Intel Mac OS X 10_9_2) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/33.0.1750.152 Safari/537.36"
5.255.253.92 - - [09/Aug/2014:06:25:05 +0400] "GET
/friends/18332 HTTP/1.1" 200 4057 "-" "Mozilla/5.0 (com-
patible; YandexBot/3.0; +http://yandex.com/bots)"
```

4.2. Настройка мониторинга трафика с помощью NetFlow

В данном пункте работы рассматривается мониторинг сетевого трафика с использованием протокола NetFlow на базе операционной системы Debian GNU/Linux версии 7.6.0.

NetFlow — сетевой протокол, предназначенный для учёта сетевого трафика, разработанный компанией Cisco Systems. Является фактическим промышленным стандартом и поддерживается не только оборудованием Cisco, но и многими другими устройствами (в частности, Juniper и Enterasys). Также существуют свободные реализации для UNIX-подобных систем. [wiki: NetFlow]

Для сбора информации о трафике по протоколу NetFlow требуются следующие компоненты:

Сенсор – собирает статистику по проходящему через него трафику. Обычно это L3-коммутатор или маршрутизатор, но в рамках данной работы будет показана установка сенсора fprobe на базе компьютера с операционной системой Debian.

Коллектор – собирает получаемые от сенсора данные и помещает их в хранилище. Он может быть организован на отдельном оборудовании с большим дисковым массивом для накопления данных, либо, как в данной работе, установлен совместно с сенсором. Коллектор может быть реализован с помощью программы nfcapd.

Анализатор – обрабатывает собранные коллектором данные и формирует пригодные для чтения человеком отчёты. С целью автоматического обнаружения DDoS-атак анализатор выполнен в виде скрипта и будет рассмотрен далее. Для преобразования в текстовый вид бинарных файлов с записями NetFlow с коллектора используется программа nfdump.

Принципиальная схема защиты сети от DDoS-атак приведена на рисунке 4.1.

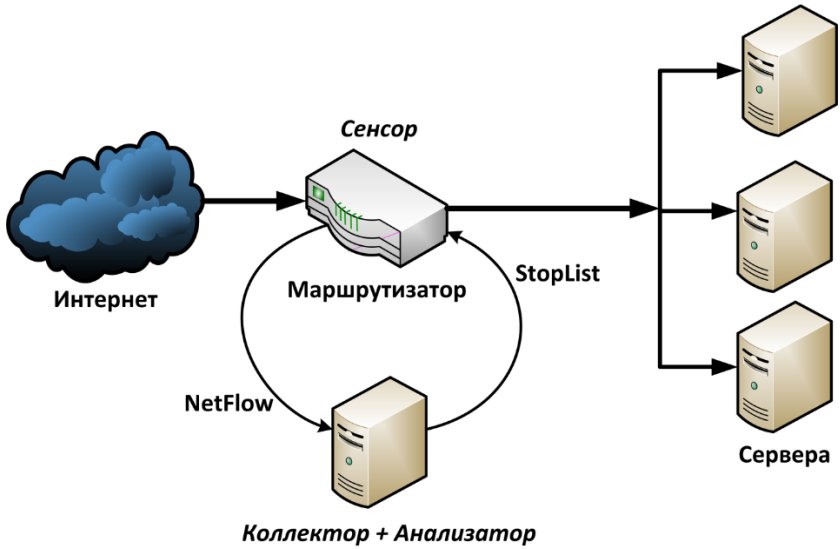

Рисунок 4.1 – Принципиальная схема защиты сети

В рамках данной работе будет показаны принципы организовации защиты на своём рабочем месте, то есть сенсор, коллектор и анализатор необходимо поставить на одном компьютере.

Вывод программы nfdump после обработки бинарного файла NetFlow показан на рисунке 4.2.

Date flow start	Duration	Proto	Src IP Addr:Port		Dst IP Addr:Port	Packets	Bytes	Flows
2014-08-20 19:46:58.713	186.112	UDP	91.222.129.245:18232	->	82.117.232.113:26848	7	383	1
2014-08-20 19:46:58.841	304.064	UDP	91.222.129.236:59611	->	2.93.40.176:60041	40	3820	1
2014-08-20 19:47:00.057	302.144	UDP	91.222.129.236:61450	->	78.63.183.7:37338	42	5722	1
2014-08-20 19:47:00.889	300.288	TCP	91.222.128.105:38557	->	176.223.215.186:5938	15	929	1
2014-08-20 19:47:01.656	234.176	UDP	91.222.129.245:18232	->	109.237.10.43:23878	31	3389	1
2014-08-20 19:47:04.280	289.984	UDP	91.222.129.236:61450	->	94.233.50.134:35691	32	4197	1
2014-08-20 19:47:04.216	229.312	UDP	91.222.129.236:59611	->	77.50.107.40:49865	33	6626	1
2014-08-20 19:47:04.728	299.136	TCP	91.222.129.163:2346	->	217.69.139.215:443	38	23344	1
2014-08-20 19:47:04.728	299.264	TCP	91.222.129.163:1714	->	217.69.139.216:443	28	16769	1
2014-08-20 19:47:05.048	264.448	UDP	91.222.129.236:61450	->	213.141.148.176:61070	65	13684	1
2014-08-20 19:47:05.048	212.928	UDP	91.222.129.236:59611	->	78.37.200.254:10028	43	8464	1
2014-08-20 19:47:05.112	296.320	UDP	91.222.128.128:64735	->	5.18.84.91:57857	64	9782	1
2014-08-20 19:47:05.432	300.224	TCP	91.222.128.181:63311	->	94.100.180.216:443	21	15502	1
2014-08-20 19:47:05.688	197.376	UDP	91.222.129.245:18232	->	128.72.34.134:12914	7	383	1

Рисунок 4.2 – Информация NetFlow после обработки программой nfdump

Таким образом, для исследования доступны следующие информационные поля:

Data flow start – время начала потока.

Duration – длительность потока.

Src IP Addr:Port – IP-адрес и порт источника потока.

Dst IP Addr:Port – IP-адрес и порт назначения потока.

Packets – количество пакетов, входящих в поток данных.

Bytes – объём данных, переданных в потоке.

Flows – указывает для какого количества потоков собрана информация. По умолчанию, если не включена агрегация потоков, в этом поле информация об одном потоке.

Формат может быть изменён при необходимости [5].

Первым делом узнаем, какие сетевые интерфейсы присутствуют на компьютере и с какого необходимо собирать статистику. Для этого необходимо выполнить команду ifconfig:

```
root@debian:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:78:4f:cf
          inet addr:192.168.2.93  Bcast:192.168.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe78:4fcf/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:16890880 errors:0 dropped:0 overruns:0 frame:0
          TX packets:11342 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:16550426545 (15.4 GiB)  TX bytes:1908453 (1.8 MiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:13002 errors:0 dropped:0 overruns:0 frame:0
          TX packets:13002 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:4669028 (4.4 MiB)  TX bytes:4669028 (4.4 MiB)
```

Как правило, отображаются два интерфейса: eth0 – сетевой адаптер Ethernet и lo – сетевой интерфейс замыкания на себя, так называемый localhost. Эта информация понадобится в дальнейшем при настройке. Если на вашем компьютере присутствуют другие сетевые интерфейсы, то необходимо выбрать тот, с которого будет собираться статистика. Как правило, интересует интерфейс, на котором наибольшее количество принятых/отправленных пакетов (поля RX packets и TX packets).

Помните, что большинство команд администрирования доступны только привилегированным пользователям! Для получения привилегий воспользуйтесь командами su или sudo.

Теперь установим NetFlow-сенсор fprobe. Для этого можно воспользоваться программой aptitude, имеющей удобный псевдографический интерфейс, или программой apt-get:

```
apt-get update #не забудьте обновить списки пакетов, если
давно этого не делали, ведь вполне возможно ваши списки
```

```
пакетов уже не соответствуют тем, что находятся на сер-
верах обновлений.
apt-get install fprobe
```

При установке будет задано несколько вопросов, изображённых на рисунках 4.3 и 4.4. Необходимо указать интерфейс, выбранный ранее для прослушивания, как правило, это eth0.

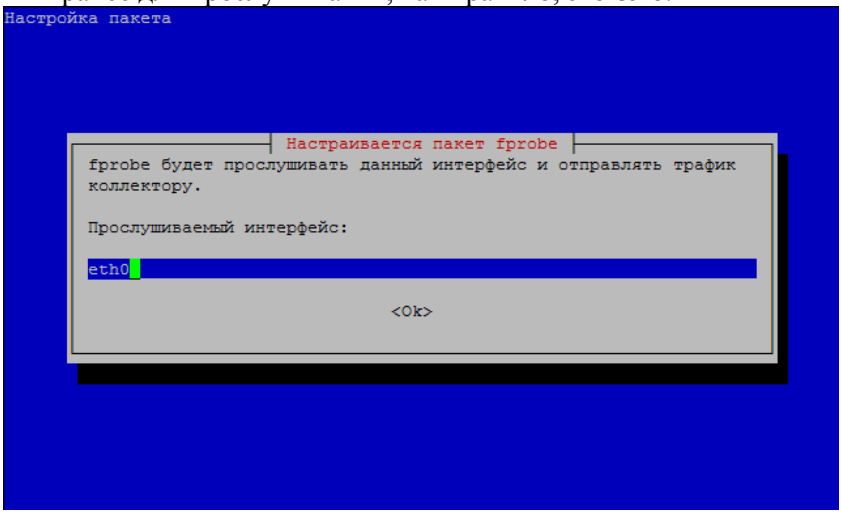

Рисунок 4.3 – Установка fprobe: прослушиваемый интерфейс

Так как коллектор расположен на той же машине, что и сенсор, в качестве адреса для отправки данных на коллектор необходимо указать адрес замыкания на себя. Порт можно оставить по умолчанию 555, но могут так же использоваться порты 2055, 9555 или 9995.

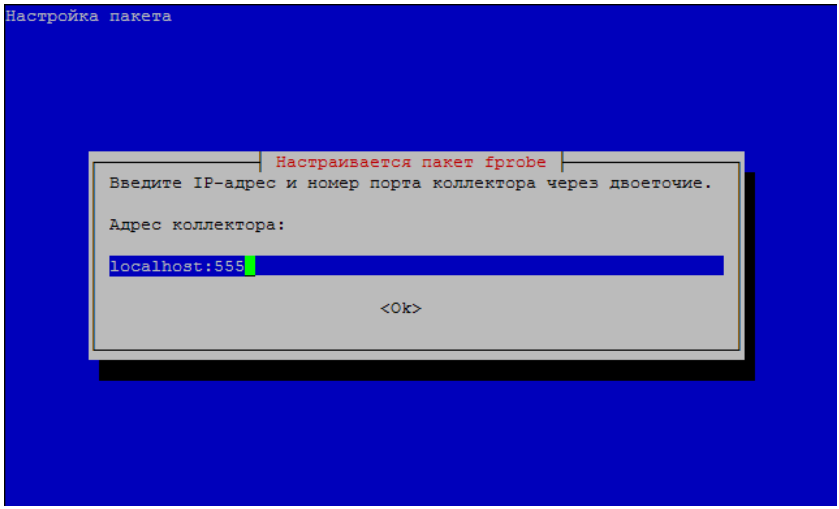

Рисунок 4.4 – Установка fprobe: адрес коллектора

Процесс установки выполнит необходимые операции и запустит сенсор. Узнать, запущен ли сенсор, можно следующей командой:

```
root@debian:~# ps aux | grep fprobe
root      3816  0.4  0.2  47632  5672 ?        Ssl  17:46   0:00 /usr/sbin/fprob
e -ieth0 -fip localhost:555
root      3831  0.0  0.0   7856   856 pts/0    S+   17:46   0:00 grep fprobe
root@debian:~#
```

Вывод команды говорит о том, что процесс fprobe запущен. Демон настраивается с помощью редактирования файла /etc/default/fprobe:

```
#fprobe default configuration file

INTERFACE="eth0" #интерфейс, с которого собирается статистика
FLOW_COLLECTOR="localhost:555" #адрес и порт коллектора

#fprobe can't distinguish IP packet from other (e.g. ARP)
OTHER_ARGS="-fip" #дополнительные настройки, передаваемые демону. По умолчанию установлен фильтр только на IP пакеты. Получить более подробную информацию можно набрав в консоли man fprobe.
```

После изменения файла необходимо перезапустить демон командой:

```
service fprobe restart
```

На практике, демон может не всегда запуститься с первого раза. Проверьте, запустился ли демон, как было показано ранее и в случае необходимости выполните дополнительную команду запуска:

```
service fprobe start
```

Параметры «по умолчанию» вполне подходят для выполнения данной работы.

Перейдём к установке коллектора nfcapd. Указанная программа входит в пакет nfdump. Установить его можно, например, таким образом:

```
apt-get install nfdump
```

Демон nfcapd имеет свой файл настроек /etc/default/nfdump:

```
# nfcapd is controlled by nfsen
nfcapd_start=no #демон nfcapd по умолчанию не запускается
автоматически
```

В указанном файле только одна опция, но существуют и другие. Посмотреть их можно в файле скрипта, для запуска этого демона /etc/init.d/nfdump. Необходимо сделать так, чтобы коллектор nfcapd прослушивал порт 555, на который сенсор fprobe присылает NetFlow данные, и сохранял эти данные в файл каждую минуту, вызывая скрипт для обработки этих данных. Для этого подсмотрим в файле /etc/init.d/nfdump опцию:

```
DATA_BASE_DIR="/var/cache/nfdump"
PIDFILE=/var/run/$NAME.pid
DAEMON_ARGS="-D -l $DATA_BASE_DIR -P $PIDFILE"
```

-D означает, что программа nfcapd будет запущена в режиме демона, то есть в фоновом режиме без взаимодействия с пользователем.

-l указывает директорию, в которую будут сохраняться двоичные файлы NetFlow.

-P указывает файл, в который будет сохранён идентификатор процесса демона при запуске. Это необходимо для дальнейшего управления этим процессом.

Из man nfcapd очевидно, что для указания порта, нужно использовать опцию “-p”, для ротации файлов каждую минуту опции “-w” и ”-t”, а для запуска скрипта при каждой ротации опцию “-x”. Изменённый файл /etc/default/nfdump теперь будет выглядеть следующим образом:

```
# nfcapd is controlled by nfsen
```

```
nfcapd_start=yes
SCRIPT="/usr/local/sbin/nfdump.pl"
PORT=555
DAEMON_ARGS="-w -t 60 -D -l $DATA_BASE_DIR -x $SCRIPT -p
$PORT
-P $PIDFILE"
```

Файл nfdump.pl буден описан в далее в работе.

Пришло время запустить демон следующей командой:

```
service nfdump start
```

Проверим, что демон действительно запустился и работает:

```
root@debian:/etc/default# ps aux | grep nfcapd
root      5413  0.0  0.0  14340   564 ?        S    22:29   0:00 /usr/bin/nfcapd
 -w -t 60 -D -l /var/cache/nfdump -x /usr/local/sbin/nfdump.pl -p 555 -P /var/ru
n/nfcapd.pid
root      5414  0.0  0.0   8696   500 ?        S    22:29   0:00 /usr/bin/nfcapd
 -w -t 60 -D -l /var/cache/nfdump -x /usr/local/sbin/nfdump.pl -p 555 -P /var/ru
n/nfcapd.pid
root      5423  0.0  0.0   7856   852 pts/1    S+   22:33   0:00 grep nfcapd
```

В результате вы должны увидеть два процесса nfcapd с указанными вами настройками. В случае возникновения затруднений, вы можете посмотреть последние записи в файле /var/log/daemon.log, как это было описано в разделе 4.1, и попытаться устранить ошибки.

Если процесс запущен, осмотрите директорию /var/cache/nfdump. В ней должны начать собираться двоичные файлы NetFlow с именами вида nfcapd.201408222240 и один файл с именем nfcapd.current. Файл с расширением current заполняется демоном nfcapd в настоящий момент и через минуту он будет автоматически переименован в файл с расширением из цифр (201408222240 → 22 августа 2014г. 22:40), которые означают текущую дату и время создания файла, а вместо него появится новый файл current.

Позаботьтесь о том, чтобы директория /var/cache/nfdump не переполнилась! Для этого старые файлы нужно своевременно удалять. Эту работу возьмёт на себя скрипт nfdump.pl, который будет рассмотрен далее в работе.

4.3. Обработка статистики журналов доступа к веб-серверу

После изучения прошлой части главы мы научились создавать и накапливать архив журналов доступа к веб-серверу по дням. Ежедневно в специально отведённой для архива директории /var/archive/ будут появляться новые файлы вида гггг-мм-дд (год-

месяц-день, например, 2014-06-20, 2014-06-21, 2014-06-22, 2014-06-23 и т.д.). Эти файлы будут создаваться один раз в сутки. Они именуются согласно текущим суткам, но содержат в себе данные за последние 24 часа, то есть, за двое суток. Время создания файлов может быть различным, в зависимости от времени ротации журналов на конкретном компьютере. Как правило, это каждые 24 часа после запуска компьютера.

Так как время ротации журналов может отличаться, файлы необходимо привести к одному временному промежутку. Можно сделать, чтобы каждый файл начинался с 00 часов 00 минут 00 секунд каждых суток и заканчивался в 23 часа 59 минут 59 секунд этих же суток, но сутки активности аудитории могут не совпадать с календарными сутками. Логично начинать новые сутки (и новый файл) в момент наименьшей активности аудитории.

Напишем скрипт для определения момента наименьшей активности аудитории. В данной работе скрипты будут писаться с использованием php и bash. Для установки интерпретатора php скриптов выполним следующую команду:

```
apt-get install php5-cgi
```

Помните, что большинство команд администрирования доступны только привилегированным пользователям! Для получения привилегий воспользуйтесь командами su или sudo.

Скрипт работает по следующему алгоритму:

1. Каждый файл содержит в себе частичные данные за двое суток: за те, что указаны в его имени, и за прошлые. Необходимо выбрать данные одних календарных суток, которые находятся в двух файлах, например файлы 2014-06-20 и 2014-06-21 содержат полный журнал от 20 июня 2014 года.

2. Обработаем первый файл. Учитывая только данные за 20 июня подсчитаем, сколько обращений было за каждый час в отдельности. Запишем результат в массив.

3. Аналогично нужно поступить со вторым файлом, но новые данные нужно добавлять в тот же массив.

4. Сохраним массив, содержащий почасовое распределение количества запросов к веб-серверу, в файл.

Далее рассмотрим текст скрипта с комментариями:

```
<?php // php activity-time.php 2014-06-20 2014-06-21
//Пример использования скрипта
```

```
if ($argc!=3) exit("Неверное число параметров!\n");
//Проверяется передано ли нужное количество параметров
$rgx = '/^201\d-\d\d-(\d\d)$/'; //Регулярное выражение
для выборки дня для анализа из первого переданного имени
файла. То есть для 2014-06-20 будет выбрано 20.
preg_match($rgx, $argv[1], $res); //Сравнение первого па-
раметра с регулярным выражением $rgx и сохранение резуль-
тата в виде массива в переменную $res
$day=$res[1]; //Получаем из массива $res день, для кото-
рого проводится анализ, и сохраняем в переменную $day

//Инициализация массива для сбора статистики посещений за
каждый час в сутках
$hours=array( '00' => 0, '01' => 0, '02' => 0, '03' =>
0, '04' => 0, '05' => 0, '06' => 0, '07' => 0, '08' =>
0, '09' => 0, '10' => 0, '11' => 0, '12' => 0, '13' =>
0, '14' => 0, '15' => 0, '16' => 0, '17' => 0, '18' =>
0, '19' => 0, '20' => 0, '21' => 0, '22' => 0, '23' => 0
);

day2hours($argv[1], $day, $hours); //Обработка первого
файла
day2hours($argv[2], $day, $hours); //Обработка второго
файла

function day2hours($filename, $day, &$hours){ //Функция
обработки файлов журналов
    $handle = @fopen($filename, 'r'); //Открываем файл
$filename для чтения
    if ($handle) { //Если файл успешно открыт
        $rgx                    =                    '/^.+
\[(\d\d)\/...\/201\d:(\d\d):\d\d:\d\d \+0400\]/'; //Ре-
гулярное выражение для выборки дня месяца и часа из каждой
записи в файле журнала
        while (($data = fgets($handle, 8000)) !== false)
{ //Считываем строку длиной до 8000 символов из файла
журнала
            preg_match($rgx, $data, $res); //Проводим
поиск по регулярному выражению $rgx в строке $data и вы-
водим все совпадения в массив $res
            if ($res[1]==$day) $hours[$res[2]]++; //Если
первый аргумент массива $res равен анализируемому дню
$day, значит в массиве $hours прибавляем одно посещение
к определённому часу $res[2]
        }
```

```
        if (!feof($handle)) { //Если предыдущий цикл за-
кончился, а конец файла всё ещё не достигнут, значит
произошла ошибка чтения из файла.
            echo "Error: unexpected fgets() fail\n";
//Выводим ошибку
        }
        fclose($handle); //Закрываем открытый файл
    }else exit("Файл не найден!\n"); //Выводим ошибку и
выходим, если файл не удалось открыть
}

$date=$argv[1].'.count'; //Добавляем к имени первого
файла расширение count и сохраняем в переменную $date

$hours=implode("\r\n",$hours); //Конвертируем массив с
почасовой статистикой $hours в текстовый вид (таблицу)
file_put_contents($date,$hours); //Сохраняем таблицу с
почасовой статистикой за сутки в файл $date
```

Обратите внимание на формат файлов журналов nginx, приведённый ниже:

```
127.0.0.1 - frank [09/Aug/2014:06:25:02 +0400] "GET
/apache_pb.gif HTTP/1.0" 200 2326 "http://www.exam-
ple.com/start.html" "Mozilla/4.08 [en] (Win98; I ;Nav)"
31.184.219.88 - - [09/Aug/2014:06:25:03 +0400] "GET
/groups/19541 HTTP/1.0" 200 4334 "-" "Mozilla/5.0 (Mac-
intosh; Intel Mac OS X 10_9_2) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/33.0.1750.152 Safari/537.36"
5.255.253.92 - - [09/Aug/2014:06:25:05 +0400] "GET
/friends/18332 HTTP/1.1" 200 4057 "-" "Mozilla/5.0 (com-
patible; YandexBot/3.0; +http://yandex.com/bots)"
```

Таким образом, выполнив следующие команды, мы получим семь файлов: 2014-06-20.count, 2014-06-21.count и т.д.

```
cd /var/archive/
php activity-time.php 2014-06-20 2014-06-21
php activity-time.php 2014-06-21 2014-06-22
…
php activity-time.php 2014-06-26 2014-06-27
```

Эти файлы будут содержать таблицы с почасовой статистикой посещений сайта за соответствующий день. По этим файлам можно построить графики, изображённые на рисунке 4.5.

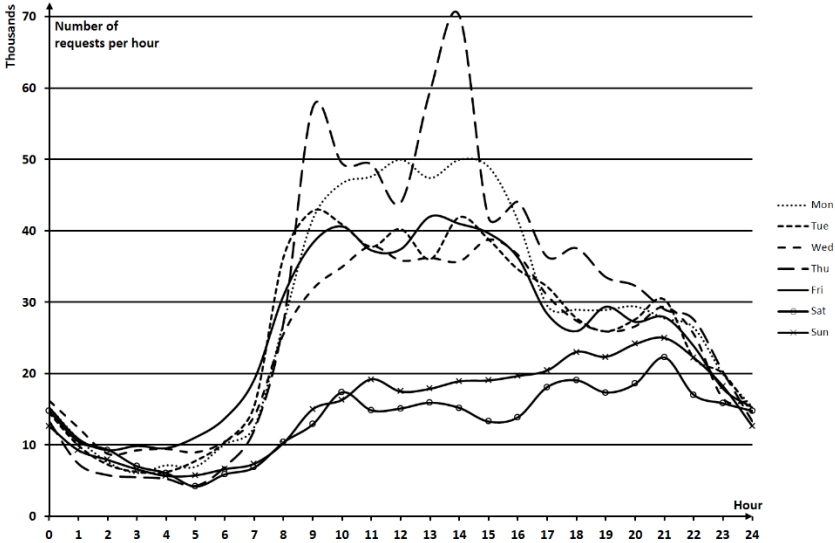

Рисунок 4.5 – Почасовая статистика посещений сайта за неделю

Как видно из рисунка, статистика сильно различается по дням недели, но при этом в любой день наименьшее число посещений находится в районе 4 часов утра. Именно это время мы возьмём для начала отсчёта новых суток и соответственно с него должен начинаться каждый файл журнала.

Следовательно, необходим скрипт, который преобразует имеющиеся в распоряжении файлы журналов веб-сервера к единой точке отсчёта времени – 4 часам утра. Алгоритм прост – проходим по всем записям файла журнала и распределяем эти записи в новые файлы в соответствии со временем каждой записи. Код скрипта с пояснениями приведён ниже:

```
<?php // time-sync.php 2014-06-20 //Пример использования скрипта

if ($argc!=2) exit("Неверное число параметров!\n"); //Проверяется передано ли нужное количество параметров

$month=array('Jan' => '01', 'Feb' => '02', 'Mar' => '03', 'Apr' => '04', 'May' => '05', 'Jun' => '06', 'Jul' => '07', 'Aug' => '08', 'Sep' => '09', 'Oct' => 10, 'Nov' => 11, 'Dec' => 12); //Массив соответствия буквенного значения месяца числовому
```

```
$hin = @fopen($argv[1], 'r'); //Открываем файл журнала
if ($hin) { //Если файл открыт продолжаем обработку
    //Регулярное выражение для разбора даты записи в
файле журнала
    $rgx                    =                    '/^.+
\[(\d\d)\/(...)\/(\d\d\d\d):(\d\d):\d\d:\d\d \+0400\]/';
    while (($data = fgets($hin, 16384)) !== false) {
//Считываем строку максимальной длиной 16384 символа из
файла
        preg_match($rgx, $data, $res); //Проводим поиск
по регулярному выражению $rgx в строке $data и выводим
все совпадения в массив $res
        if (isset($day)) { //Если переменная $day задана
            if ($day!=$res[1] && $res[4]>=4) { //Создаём
новый файл, если день текущей записи журнала не совпадает
с днём первой записи и время больше или равно 4 часам
ночи
                fclose($hout); //Закрываем ранее откры-
тый для записи файл
                //Открываем (создаём новый, если не су-
ществует) файл в директории new, и задаём имя исходя из
даты в записи журнала
                $hout=fopen('new/'.$res[3].'-
'.$month[$res[2]].'-'.$res[1],'a');
                $day=$res[1]; //Задаём день из записи
журнала
            }
        }else{ //Если переменная $day не задана (это пер-
вая запись в файле)
            //Открываем (создаём новый, если не суще-
ствует) файл в директории new, и задаём имя исходя из
даты в записи журнала
            $hout=fopen('new/'.$res[3].'-
'.$month[$res[2]].'-'.$res[1],'a');
            $day=$res[1]; //Задаём день из записи журнала
        }
        fwrite($hout,$data); //Переносим запись журнала
из файла $hin в файл $hout
    }
    if (!feof($hin)) { //Если предыдущий цикл закончился,
а конец файла всё ещё не достигнут, значит произошла
ошибка чтения из файла.
        echo   "Error:   unexpected   fgets()   fail\n";
//Выводим ошибку
    }
    if  (isset($hout))  fclose($hout);  //Закрываем  файл
$hout, если открыт
```

```
    fclose($hin); //Закрываем файл $hin
}else exit("Файл не найден!\n"); //Если файл $hin открыть
не удалось – выводим ошибку
```

Теперь необходимо заранее создать директорию new, в которую будут помещаться приведённые к временному соответствию файлы журналов. Запускать скрипт необходимо строго последовательно в соответствии с датой для каждого файла журнала. Пример использования приведён ниже:

```
cd /var/archive/
mkdir new
php time-sync.php 2014-06-20
php time-sync.php 2014-06-21
php time-sync.php 2014-06-22
…
```

Теперь сутками мы будем называть промежуток времени с 04:00:00 до 03:59:59. Файлы журналов, приведённые к этому промежутку, находятся в директории new.

Пришло время получить статистику по каждому из дней анализа. Следующие значения уже можно вычислить:

1. IP-адреса запросов к веб-серверу за сутки

```
cd /var/archive/new/ # Переходим в каталог с файлами жур-
налов
cat 2014-06-20 | grep -o "^[0-9\.]*" > 2014-06-20.ip #
IP-адреса запросов сохраняются в файл 2014-06-20.ip
```

2. Уникальные IP-адреса за сутки

```
cat 2014-06-20.ip | sort –u > 2014-06-20.sort # Сохраняет
в файл 2014-06-20.sort  уникальные IP-адреса клиентов за
сутки
```

3. Уникальные IP-адреса со времени начала анализа на каждые сутки

```
cat 2014-06-19.all 2014-06-20.sort | sort –u > 2014-06-
20.all # Добавляем к базе данных уникальные IP-адреса за
20 июля 2014 года отбрасывая повторения
# Если файл 2014-06-19.all ещё не существует просто не
указывайте его
```

4. Новые (не встречавшиеся ранее) IP-адреса за сутки

```
cat 2014-06-19.all 2014-06-20.all | sort | uniq –u > 2014-
06-20.new # Вычисляем построчную разницу между текущим
состоянием базы данных IP-адресов и прошлым
```

Вышеописанные расчёты могут быть автоматизированы следующим образом:

```
OLD='2014-05-31' # Дата прошлого состояния базы данных
NEW='2014-06-01' # Дата, с которой начнётся анализ
END='2014-07-01' # Дата окончания анализа
while [ "$NEW" != "$END" ]
do
        #echo 'old '$OLD' new '$NEW
        OLD=`date -d $OLD' 1 days' +%Y-%m-%d` # +1 день
к $OLD
        NEW=`date -d $NEW' 1 days' +%Y-%m-%d` # +1 день
к $NEW
        cat $NEW | grep -o "^[0-9\.]*" > $NEW.ip
        cat $NEW.ip | sort -u > $NEW.sort
        cat $OLD.all $NEW.sort | sort –u > $NEW.all
        cat $OLD.all $NEW.all | sort | uniq –u > $NEW.new
        #Выведем количество строк (IP-адресов) в полу-
чившихся файлах
        IP=`cat $NEW.ip | wc -l`
        SORT=`cat $NEW.sort | wc -l`
        ALL=`cat $NEW.all | wc -l`
        NEW=`cat $NEW.new | wc -l`
        echo $NEW $IP $SORT $ALL $NEW
done
```

Для запуска скрипта необходимо сохранить его в файл и выполнить:

```
bash script.sh
```

1. Количество обращений с каждого IP-адреса за сутки (таблицей)

```
cat 2014-06-20.ip | sort | uniq –c > 2014-06-20.count
```

2. Сколько суток задействован IP-адрес

```
START='2014-06-01' # Дата, с которой начнётся анализ
END='2014-07-01' # Дата окончания анализа (не включается)
CUR=$START
while [ "$CUR" != "$END" ] # Цикл от начальной даты до
конечной
do
        ALL=$ALL$CUR'.sort ' # В переменную $ALL через
пробел записываем все файлы для анализа
        CUR=`date -d $CUR' 1 days' +%Y-%m-%d` # +1 день
к $CUR
done

cat $ALL | sort | uniq –c > $START_$END.assigned
```

Вышеописанные скрипты выводят таблицы, которые могут быть открыты табличным процессором (например, Microsoft Excel) и по ним построены графики для наглядного анализа.

4.4. Обработка статистики NetFlow

Ранее в главе был рассмотрен процесс настройки мониторинга сетевого трафика с использованием протокола NetFlow на базе операционной системы Debian GNU/Linux версии 7.6.0. Теперь каждую минуту в директории /var/cache/nfdump на вашем компьютере создаётся новый бинарный файл NetFlow с записями о сетевых потоках за прошлую минуту. При ротации файла запускается скрипт /usr/local/sbin/nfdump.pl, который должен производить следующие операции:

1. Преобразовывать бинарные файлы в текстовые таблицы, пригодные для статистической обработки.
2. Выводить статистику по количеству завершившихся потоков.
3. Обнаруживать атаку на веб-сервер по количеству запросов с одного IP-адреса.
4. Заносить IP-адрес атакующего в файл журнала.
5. Блокировать IP-адрес атакующего компьютера.
6. Разблокировать IP-адрес через определённое время.

Так как текстовые файлы nfdump представляют собой таблицы большого объёма, то для их разбора лучше всего подойдёт специально для этих целей созданный скриптовой язык Perl. Именно на нём будет выполнен обрабатывающий скрипт. Интерпретатор этого языка уже предустановлен в базовой версии Debian сервера. Для разблокировки IP-адреса при блокировании он должен быть куда-то записан, а также сохранено время блокировки. Удобней всего проводить запись в базу данных. Кроме того, в БД можно сохранять ежеминутную статистику по количеству потоков. Для этого установим, если этого не было сделано раньше, базу данных MySQL:

```
apt-get install mysql-server
```

Помните, что большинство команд администрирования доступны только привилегированным пользователям! Для получения привилегий воспользуйтесь командами su или sudo.

В процессе установки будет запрошен пароль для учётной записи root базы данных, как это показано на рисунке 4.6. Затем на следующем экране нужно повторить этот пароль, чтобы не было ошибки. Не путайте пароль администратора MySQL с паролем операционной системы!

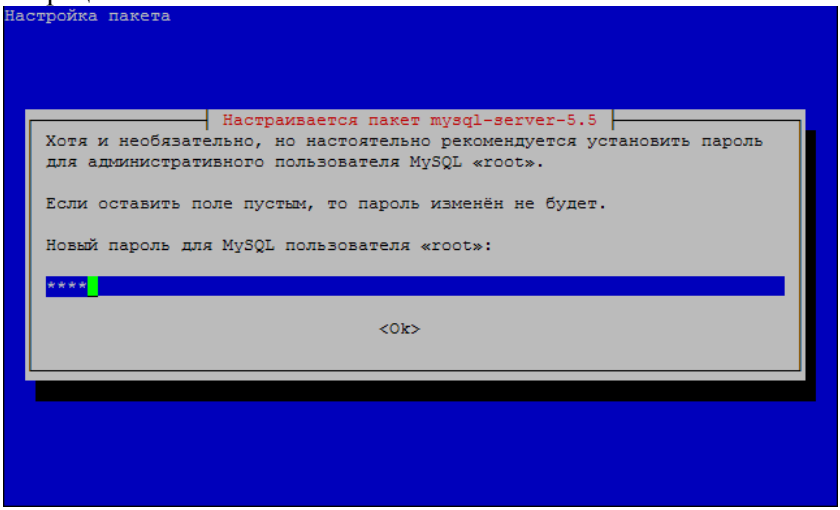

Рисунок 4.6 – Установка MySQL: запрос root пароля

Обратите внимание, что если MySQL не работает должным образом её можно полностью переустановить, выполнив следующие команды:

```
apt-get remove --purge ^mysql-server-* mysql-common
apt-get install mysql-server
```

Теперь пришло время создать базу данных, таблицы и пользователя для скрипта. Это можно сделать, подключившись к БД следующей командой:

```
mysql -user=root -p
```

Либо установить на веб-сервер phpMyAdmin [6]. Затем выполните следующие SQL команды:

```
# Создание БД с именем nfdump_pl
CREATE DATABASE `nfdump_pl` DEFAULT CHARACTER SET utf8
COLLATE utf8_general_ci;

# Создание пользователя nfdump_pl, который имеет только
локальный доступ на выборку, вставку, обновление и уда-
ление записей в базе данных nfdump_pl с паролем password
```

```
GRANT SELECT, INSERT, UPDATE, DELETE PRIVILEGES ON
`nfdump_pl` TO nfdump_pl@localhost IDENTIFIED BY 'pass-
word';

# Переход в контекст БД nfdump_pl
USE `nfdump_pl`;

# Таблица для хранения забаненных IP-адресов
# Создание таблицы banlist с полями id (автоматический
идентификатор записи), ip (забаненный IP-адрес), time
(автоматически проставляемое время создания записи)
CREATE TABLE IF NOT EXISTS `banlist` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `ip` int(10) unsigned NOT NULL,
  `time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  UNIQUE KEY `ip` (`ip`),
  KEY `time` (`time`)
) ENGINE=MyISAM  DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;

# Таблица для сохранения количества потоков в определён-
ное время
# Создание таблицы flow_cnt с полями ts (автоматически
проставляемое время создания записи), flows (количество
потоков)
CREATE TABLE IF NOT EXISTS `flow_cnt` (
  `ts` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `flows` int(10) unsigned NOT NULL,
  PRIMARY KEY (`ts`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

Для занесения IP-адреса в таблицу необходимо выполнить по крайней мере два запроса к БД:

1. Проверить, не заблокирован ли IP-адрес уже.
2. Если заблокирован, то обновить время блокировки, в противном случае добавить IP-адрес в таблицу.

Для уменьшения количества запросов к БД ниже приведено создание специальной функция ban. Она будет использоваться в скрипте. Код необходимо выполнить в SQL окне:

```
# Создание функции ban, принимающей в качестве аргумента
IP адрес, а возвращающей состояние: 0 – IP-адрес уже су-
ществует в таблице, 1 – IP-адрес внесён в таблицу.
CREATE FUNCTION `ban`(parip VARCHAR(15)) RETURNS ti-
nyint(1)
BEGIN
```

```
  # Переменная для идентификатора записи
  DECLARE vid INT(10) UNSIGNED DEFAULT 0;
  # Переменная для преобразованного в число IP адреса
  DECLARE iip INT(10) UNSIGNED DEFAULT INET_ATON(parip);
  # Поиск IP-адреса в таблице banlist
  SELECT id INTO vid FROM banlist WHERE ip=iip LIMIT 1;
  # Если IP-адрес не найден
  IF vid = 0 THEN
    # Вставляется новая запись
    INSERT INTO banlist (ip) VALUES (iip);
  ELSE # Если адрес найден
    # Обновляется время в существующей записи
    UPDATE banlist SET time = CURRENT_TIMESTAMP WHERE id
= vid;
  END IF;
  # Возвращается 0 – IP-адрес уже существует в таблице,
1 – IP-адрес внесён в таблицу
  RETURN vid=0;
END
```

Далее приведён код скрипта на языке Perl с комментариями:

```
#!/usr/bin/perl
use 5.8.8; use strict; use warnings; use DBI; # подклю-
чение Perl библиотек

# данные для подключения к БД
my $hostname = "localhost";
my $database = "nfdump_pl";
my $user = "nfdump_pl";
my $password = "password";

my $dbh = 0; # указатель на подключение к БД
my $sth = 0; # запрос к БД

my $path_nfcapd = "/var/cache/nfdump/"; # путь к бинарным
файлам nfcapd
my $path_nfdump = "/var/nfdump/";       # путь к тексто-
вым файлам nfdump

# поиск самого нового файла файла в архиве
chdir $path;                        # задаёт текущей дирек-
торию $path
my (@file_list) = glob "nfcapd.20*";    # сохраняем спи-
сок файлов по маске в массив @file_list
my $flowfile = $file_list[$#file_list]; # в переменную
$flowfile сохраняем значение последнего элемента массива
@file_list
```

```
undef @file_list;                          # удаляем из па-
мяти массив @file_list
##############################

# преобразование бинарного формата nfdump в текстовый в
той же директории
system   "nfdump   -r   ".$path_nfcapd.$flowfile."   >
".$path_nfcapd.$flowfile.".txt";
# удаление бинарного файла (если конечно мы не хотим со-
хранить его в архиве)
system "rm -f ".$path_nfcapd.$flowfile;

open(FileData,$path_nfdump.$flowfile.".txt"); # открытие
текстового файла nfdump
my @lines = <FileData>; # считываем содержимое файла в
массив строк
close(FileData); # закрываем файл

# копирование текстового файла nfdump в архивную дирек-
торию (если нам нужен архив)
system        "cp        ".$path_nfcapd.$flowfile.".txt
".$path_nfdump.$flowfile.".txt";
# удаление текстового файла nfdump из директории с би-
нарными файлами
system "rm -f ".$path_capd.$flowfile.".txt";

my $fcnt = 0; # общее количество потоков на весь файл (за
минуту)

my @IPs = ();       # массив [IP-адрес, Дата] для потоков

for ($i = 1; $i < @lines-4; $i++) { # заполнение массивов
исходных данных
  if    (    $lines[$i]    =~    m/^(\d{4}-\d{2}-\d{2}
\d{2}:\d{2}:\d{2}\.\d{3})  {1,5}\d{1,5}\.\d{3}  \w{1,6}
+(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}):[\d        ]{6}->
+(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}):\d{1,5} +\d+ +\d+
+(\d+)$/ ) {
# каждая строка файла nfdump обрабатывается регулярным
выражением и на каждой итерации переменным присваиваются
следующие значения:
# $1 - Дата и время фиксации потока в файле
# $2 - IP-адрес источника
# $3 - IP-адрес назначения
# $4 - зафиксированное количество потоков
    if ($3 eq "IP нашего сервера") { # для всех потоков
идущих на наш сервер
```

```
      if (($2 ne "IP DNS 1") && ($2 ne "IP DNS 2")) { #
исключаем ДНС сервера
        $fcnt += $4; # прибавляем к переменной $fcnt ко-
личество зафиксированных потоков
        push(@IPs , [$2,$1]); # добавляем в массив @IPs
запись о потоке
      }
    }
  }
}

# сортировка массива @IPs по IP-адресу источника
@IPs = sort { ( $a->[0] cmp $b->[0] ) } @IPs;

open ( LogFile, ">> /var/log/nfdump.pl.log" ); # откры-
ваем файл журнала для записи
$dbh   =   DBI->connect("DBI:mysql:$database:$hostname",
$user, $password) || print LogFile "Got error". $dbh-
>errstr ."\n"; # получение доступа к БД, в случае неудачи
ошибка сохраняется в журнал

# заносим общее число потоков в таблицу flow_cnt БД
$sth = $dbh->prepare("INSERT INTO flow_cnt (flows) VALUES
(".$fcnt.")"); # подготовка запроса
$sth->execute; # выполнение запроса

my $ref = 0; # служебная переменная для результатов за-
просов к БД

my $count = 1; # счетчик числа потоков
my $i = 0; # служебная переменная для цикла
for ($i = 1; $i < $#IPs; $i++) { # проход по всем элементам
массива IPs
  if ($IPs[$i-1][0] eq $IPs[$i][0]) { # если текущий IP
адрес в элементе массива IPs равен предыдущему, то число
потоков на этот IP увеличивается на единицу
    ++($count);
  } else { # в противном случае
    if ($count > 300) { # проверяется не превысило ли
количество потоков 300 и если превысило, то
      my $ipT = $IPs[$i-1][0]; # в переменную $IPt со-
храняется IP-адрес атакующего
      print     LogFile     $IPs[$i-1][1],"     ",$ipT,"
",$count,"\n"; # IP-адрес атакующего заносится в журнал

#>>>>> Следующие строки используются только в данной ра-
боте для непосредственной блокировки IP-адреса атакующего
```

```
и в дальнейшем в работе будет предложен универсальный
способ блокировки при помощи fail2ban, который работает
с файлами журналов.

      # IP-адрес атакующего заносится в таблицу забанен-
ных в БД
      $sth = $dbh->prepare("select ban('".$ipT."');"); #
подготовка запроса к БД
      $sth->execute; # выполнение запроса к БД
      if    (($ref    =    $sth->fetchrow_arrayref)    &&
($$ref[0]==1)) { # проверяется выполнился ли запрос к БД
и какой статус вернула функция ban (0 уже в таблице за-
баненных, 1 успешно помещён в список)
        system "iptables -I fail2ban-galcev -s ".$ipT."
-j DROP"; # блокировка IP-адреса на сервере с помощью
iptables
      }
#<<<<<
    }
    $count = 1; # сброс счётчика количества потоков
  }
}

#>>>>> Далее в работе будет предложен универсальный спо-
соб блокировки при помощи ipset, который работает значи-
тельно быстрее.

# функция разбана IP-адресов по прошествии 10 минут
# из таблицы banlist выбираются все IP-адреса, занесённые
более 10 минут назад
$sth = $dbh->prepare("SELECT id, INET_NTOA(ip) FROM ban-
list  WHERE  time<CURRENT_TIMESTAMP  -  INTERVAL  10  MI-
NUTE;"); # подготовка запроса к БД
$sth->execute; # выполнение запроса к БД
while ($ref = $sth->fetchrow_arrayref) { # цикл по всем
выбранным IP-адресам
  system "iptables -D fail2ban-galcev -s ".$$ref[1]." -j
DROP"; # разблокировка IP-адреса на сервере с помощью
iptables
  # удаление IP-адреса из БД
  my $sth2 = $dbh->prepare("DELETE FROM banlist WHERE
id='".$$ref[0]."';"); # подготовка запроса
  $sth2->execute; # выполнение запроса
  $sth2->finish; # очистка памяти
}
#<<<<<
```

```
$sth->finish; # очистка памяти
$dbh->disconnect; # закрытие подключения к БД
close(LogFile); # закрытие файла журнала

exit( 0 ); # успешное завершение скрипта
```

Не забудьте создать директорию /var/nfdump/ для архива файлов nfdump в текстовом виде. Обратите внимание, что эта директория будет быстро заполняться новыми файлами и вам придётся периодически её очищать. Если вы не хотите содержать полный архив – закомментируйте копирование файлов в эту директорию в начале скрипта.

Проверить работу скрипта можно выполнив следующие команды:

```
# Подключение к MySQL
mysql –user=nfdump_pl –p

# Вход в контекст БД nfdump_pl
USE `nfdump_pl`

# Запрос 10 последних записей из таблицы flow_cnt
SELECT * FROM `flow_cnt` ORDER BY `ts` DESC LIMIT 10;
```

Перед вами должна появиться таблица следующего вида:

```
+---------------------+-------+
| ts                  | flows |
+---------------------+-------+
| 2014-06-04 23:49:01 |   143 |
| 2014-06-04 23:48:01 |   164 |
| 2014-06-04 23:47:00 |   138 |
| 2014-06-04 23:46:00 |   198 |
| 2014-06-04 23:45:00 |   163 |
| 2014-06-04 23:44:00 |   256 |
| 2014-06-04 23:43:00 |   183 |
| 2014-06-04 23:42:00 |   175 |
| 2014-06-04 23:41:01 |   232 |
| 2014-06-04 23:40:01 |   203 |
+---------------------+-------+
10 rows in set (0.00 sec)
```

В ней отображены последние 10 записей, которые скрипт сделал в БД. Первая запись должна быть сделана меньше минуты назад.

Для того чтобы иметь возможность отлаживать скрипт и видеть ошибки, происходящие в нём рекомендуется вышеуказанный скрипт назвать /usr/local/sbin/nfdump2.pl, а в качестве файла

/usr/local/sbin/nfdump.pl использовать нижеследующий:

```
#!/usr/bin/perl
use 5.8.8; use strict; use warnings; use DBI; # подклю-
чение Perl библиотек

# Выполнение скрипта nfdump2.pl и добавление стандартного
вывода и вывода ошибок в файл журнала.
system          "/usr/local/sbin/nfdump2.pl          >>
/var/log/nfdump.pl.errors.log 2>&1";

exit( 0 ); # успешное завершение скрипта
```

Этот скрипт избавит нас от проблем с программой nfdump при поломке главного скрипта, а так сохранит информацию для отладки в файл журнала.

В связи с тем, что в директории /var/cache/nfdump/ постоянно создаются и удаляются файлы (она участвует в конвертации из бинарного формата nfdump в текстовый) создаётся большая нагрузка на систему хранения данных компьютера. Все эти файлы временные, поэтому в этой директории правильно будет использовать файловую систему tmpfs. Её смысл в том, чтобы не записывать временные файлы на диск, а отвести часть оперативной памяти для их хранения. Таким образом, производительность доступа к этим файлам значительно повышается, а износ системы хранения данных уменьшается. Для этого необходимо добавить в файл /etc/fstab следующую информацию:

```
# <file system> <mount point>     <type>    <options>
<dump>  <pass>
nfdump       /var/cache/nfdump  tmpfs   size=24M       0
0
```

Что означает, что в директории /var/cache/nfdump/ будет использоваться файловая система tmpfs размеров в 24Мб. Далее приведён список команд для внесения изменений:

```
# Добавляет необходимую строку в файл fstab
echo "nfdump /var/cache/nfdump tmpfs size=24M 0 0" >>
/etc/fstab

# Монтирование новой файловой системы
mount nfdump

# Проверка
df -h /var/cache/nfdump
```

```
# Размонтирование файловой системы при необходимости
umount nfdump
```

Когда вы убедитесь, что скрипт работает, и БД наполняется можно перейти к следующему этапу – создание страницы для отображения общей статистики по потокам. Ниже приведёт PHP скрипт flows.php, который делает выборку из базы данных и выводит на страницу среднее значение потоков в минуту за прошлые 1, 5, 10, 30 и 60 минут. Страница автоматически обновляется каждые 10 секунд.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="Refresh" content="10" />
  <title>Flows Statistics</title>
</head>
<body>
<?

// Параметры подключения к БД
$hostname = "localhost";
$database = "nfdump_pl";
$user = "nfdump_pl";
$password = "password";

// Подключение к MySQL
$link = mysql_connect($hostname, $user, $password);
if (!$link) {
    die('Ошибка соединения: ' . mysql_error());
}

// Переход в контекст БД
$db_selected = mysql_select_db($database, $link);
if (!$db_selected) {
    die ('Не удалось выбрать базу foo: ' . mysql_er-
ror());
}

// Выборка последних 60 записей из таблицы flow_cnt
$result = mysql_query("SELECT flows FROM flow_cnt ORDER
BY ts DESC LIMIT 60");
if (!$result) {
    die ('Ошибка запроса: ' . mysql_error());
}
```

```
$cnt=0; // количество элементов (минут в данном случае)
в выборке
$sum=0; // сумма потоков во всех полученных записях

while($row=mysql_fetch_array($result)) { // цикл по всем
полученным записям
    $sum+=$row[0]; // прибавляем к сумме количество по-
токов в текущей записи
    $cnt++; // количество элементов (минут) в выборке +1
    if($cnt==1)echo  $sum."  flow/min  (last  minute)<br
/>\n";
    // если было просуммировано пять элементов (минут)
выборки, то делим сумму на 5 и выводим округлённый ре-
зультат на страницу
    else if($cnt==5)echo round($sum/5)." flow/min (last
5 minute)<br />\n";
    else   if($cnt==10)echo   round($sum/10)."   flow/min
(last 10 minute)<br />\n";
    else   if($cnt==30)echo   round($sum/30)."   flow/min
(last 30 minute)<br />\n";
    else   if($cnt==60)echo   round($sum/60)."   flow/min
(last hour)<br />\n";
}

mysql_close($link); // закрываем подключение к MySQL

?>
</body>
</html>
```

Указанный скрипт необходимо разместить на веб-сервере, в директории, в который исполняются PHP скрипты. Чтобы увидеть результат обратитесь к скрипту при помощи браузера, например http://localhost/flows.php.

4.5. Мониторинг трафика на уровне пакетов

Ранее в данной главе были рассмотрены способы мониторинга активности пользователей на уровне запросов к веб-серверу и потоков NetFlow. В этом пункте главы приводятся способы мониторинга сети на уровне пакетов. Зачем это нужно? Дело в том, что существует такой тип атак UDP-flood, при котором на адрес компьютера-жертвы отправляется большое количество UDP пакетов, которые забивают канал и блокируют прохождение нормальных запросов от пользователей, возникает отказ в обслуживании. Так

как веб-сервер работает с TCP подключениями, то в журналах доступа такой трафик никак не отображается. В частом случае, когда порты источника и получателя у всех пакетов одинаковы в течении длительного времени, анализ на основе NetFlow потоков тоже не покажет ничего необычного, ведь это один большой поток данных и не более того. В свою очередь с помощью мониторинга трафика на уровне пакетов можно мгновенно отреагировать на превышение лимита, коим может выступать, как количество информации, так её скорость передачи или другие статистические особенности.

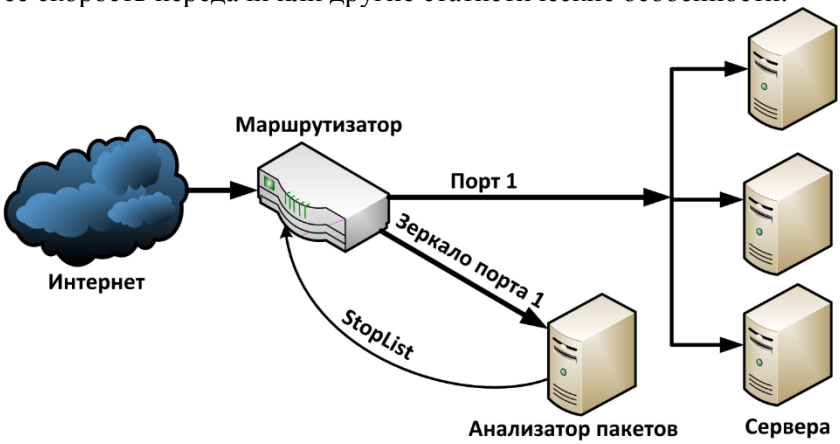

Рисунок 4.7 - Принципиальная схема установки анализатора пакетов

Схема установки анализатора пакетов в сети приведена на рисунке 4.7. Она похожа на схему на рисунке 4.1, с той лишь разницей, что теперь с маршрутизатора на анализатор поступает не статистическая информация NetFlow, а дублируются все пакеты с порта, к которому подключены сервера. Зеркалирование порта это стандартная функция для маршрутизаторов и коммутаторов уровня предприятия. В случае если на маршрутизаторе нет свободных портов, функция зеркалирования может быть возложена на промежуточный коммутатор. Анализатор пакетов в свою очередь должен отправлять на маршрутизатор информацию по блокируемым IP-адресам. В рамках работы будет рассмотрен минимальный вариант с установкой анализатора пакетов непосредственно на рабочем компьютере, который необходимо защитить.

В этой работе будет рассмотрена программа с минимальным набором функций. Это шаблон или «болванка», которая нуждается в доработке с точки зрения статистического анализа и алгоритмики обнаружения атак, но она небольшая в объёме и достаточно проста, чтобы проводить обучение на её основе.

Программа использует свободную библиотеку Pcap [7] (Packet Capture) для получения информации о приходящих по сети пакетах. На основе этой библиотеки работают такие проекты, как tcpdump, Wireshark, Snort, fprobe и многие другие. Ниже приведён текст программы, написанной на языке Си, с комментариями.

```
#define APP_NAME           "netmon" // название программы
#define APP_DESC           "Network monitor" // описание
программы
// информация об авторстве
#define APP_COPYRIGHT      "Copyright   (c)   2005   The
Tcpdump Group, 2014 Evgeny Sagatov"
// отказ от ответственности
#define APP_DISCLAIMER     "THERE   IS   ABSOLUTELY   NO
WARRANTY FOR THIS PROGRAM."

// Подключение дополнительных заголовочных файлов
#include <pcap.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdbool.h>
#include <signal.h>
#include <syslog.h>

/* default snap length (maximum bytes per packet to
capture) */
#define SNAP_LEN 1518

/* размер заголовка канального уровня. Поддерживаются
заголовок Ethernet, размер которого всегда 14 байт, и
Linux cooked-mode capture (SLL), размер которого 16 байт.
Таким образом программа может функционировать с
большинством устройств. */
```

```
#define SIZE_ETHERNET 14
#define SIZE_LINUX_SLL 16

int header_size; // реальный размер заголовка после
определения типа сети

/* IP header */
struct sniff_ip {
        u_char  ip_vhl;                 /* version << 4
| header length >> 2 */
        u_char  ip_tos;               /* type of service
*/
        u_short ip_len;                 /* total length
*/
        u_short ip_id;                  /* identification
*/
        u_short ip_off;               /* fragment offset
field */
        #define IP_RF 0x8000                /* reserved
fragment flag */
        #define IP_DF 0x4000            /* dont fragment
flag */
        #define IP_MF 0x2000           /* more fragments
flag */
        #define IP_OFFMASK 0x1fff          /* mask for
fragmenting bits */
        u_char  ip_ttl;                 /* time to live
*/
        u_char  ip_p;                   /* protocol */
        u_short ip_sum;                 /* checksum */
        struct   in_addr ip_src,ip_dst;   /* source and
dest address */
};
// макрос для извлечения размера IP заголовка
#define IP_HL(ip)               (((ip)->ip_vhl) & 0x0f)
// макрос для извлечения версии IP протокола
#define IP_V(ip)                (((ip)->ip_vhl) >> 4)

/* Далее идёт блок кода с описанием списка, для хранения
IP-адресов, пакеты с которых пришли на компьютер, и
различных статистических параметров, например количество
пакетов разного типа и объём полученной информации.
Список выполнен в виде массива ссылок на структуры типа
stat_ip, каждая из которых хранит IP-адрес и
статистическую информацию по нему. Все элементы списка
упорядочиваются при добавлении по IP-адресу. Такой подход
даёт возможность использовать очень быстрый алгоритм
```

```
бинарного поиска с незначительно большими затратами
вычислительной мощности и памяти при добавлении
элементов. Дублирование IP-адресов запрещено. */

struct stat_ip { // ячейка списка
      struct    in_addr  ip;  //  IP-адрес  удалённого
компьютера
      unsigned long tcp, tcp_size, udp, udp_size, other,
other_size; /* количество пакетов различного типа и общий
размер полученной информации в этих пакетах. */
      unsigned long udp_attack; /* количество полученных
UDP пакетов. Используется для определения атаки UDP-
flood. Когда лимит на количество пакетов превышен IP-
адрес блокируется, а данная переменная обнуляется. */
};

struct { /* структура без имени описывающая только одну
глобальную переменную stat_list. Является описанием
массива ссылок stat_ip. */
       struct  stat_ip  **list;  //  массив  ссылок  на
структуры stat_ip
       unsigned  long  size;  //  количество  элементов
stat_ip в массиве list
} stat_list = { .size=0 }; // при создании stat_list поле
size инициализируется нулём

/* Функция реализует алгоритм бинарного поиска. Если IP-
адрес в списке найден, то функция возвращает порядковый
номер элемента. Если не найден, то возвращается позиция
вставки нового IP-адреса. В качестве аргумента функция
принимает IP-адрес клиента в виде структуры in_addr. */
unsigned long stat_about(struct in_addr ip_src){
       long   min=0,   max=stat_list.size-1,   cur;   /*
описание классических для бинарного поиска переменных
min, max и cur */
       while(min<=max){ // поиск выполняется пока min
меньше или равно max
              cur=min+(max-min)/2;    //    выбирается
элемент посередине между min и max
              /* если IP-адрес выбранного элемента в
списке меньше переданного */
              /* в функцию IP-адреса, то значит искомый
элемент больше cur */
              if(stat_list.list[cur]->ip.s_addr      <
ip_src.s_addr){
                     /*      задаём      минимальный
исследуемый элемент, как cur+1 */
```

```
                        min=cur+1;
                /* если IP-адрес выбранного элемента в
списке больше переданного */
                /* в функцию IP-адреса, то значит искомый
элемент меньше cur */
                }else  if(stat_list.list[cur]->ip.s_addr
> ip_src.s_addr){
                        /*      задаём      максимальный
исследуемый элемент, как cur-1 */
                        max=cur-1;
                /* если IP-адреса равны, значит искомый
элемент найден */
                }else{
                        /*     выходим     из     функции,
возвращаем номер искомого элемента */
                        return cur;
                }
        }
        /* если IP-адрес в списке найден не был, то
возвращается позиция */
        /* вставки нового элемента с тем IP-адресом, что
был передан в функцию */
        return min;
}

/* Функция stat_set является основным обработчиком
списка. Информация (исходящий IP-адрес, протокол, размер
пакета) о всех пришедших пакетах должна быть передана в
эту функцию. Первым делом в функции производится поиск
переданного IP-адреса в списке stat_list. Если он
находится, то в статистику по нему добавляются переданные
значения. Если IP-адреса ещё нет в списке, то он
вставляется в позицию, при которой список будет
оставаться упорядоченным по IP-адресам. При каждой
вставке происходит создание нового массива адресов
new_list, который на один элемент больше старого
old_list. Если новый элемент должен быть добавлен в
позицию index, то из старого массива копируются все
элементы до index, в их текущих позициях, а начиная с
index в новый массив элементы переносятся со здвигом +1
элемент. Затем создаётся новый элемент в позиции index.
*/
void stat_set(struct in_addr ip_src, u_char ip_p,
u_short ip_len){
      /* Вызывается функция stat_about для поиска
переданного IP-адреса в списке stat_list. index будет
содержать порядковый номер элемента в списке, либо
```

```
место в списке, куда необходимо вставить новый элемент.
*/
       unsigned long index = stat_about(ip_src);
      /* Если index выходит за размер списка или IP-
адрес по указанному индексу не равен искомому IP-
адресу, то */
       if(stat_list.size==index ||
          stat_list.list[index]-
>ip.s_addr!=ip_src.s_addr){
            /* сохраняется ссылка на массив list во
временную переменную */
             struct stat_ip **old_list=stat_list.list;
            /* создаётся новый массив, на один элемент
больше старого */
             struct stat_ip
**new_list=malloc(sizeof(struct stat_ip*)*
(stat_list.size+1));
            /* копируются ссылки на элементы из старого
массива */
            /* в новый до элемента index */

       memcpy(new_list,old_list,index*sizeof(struct
stat_ip*));
            /* копируются после index со сдвигом вправо
на один элемент*/

       memcpy(&new_list[index+1],&old_list[index],
                sizeof(struct stat_ip*)*(stat_list.size-
index));
            /* создаётся новый элемент массива с
порядковым номером index */
             new_list[index]=malloc(sizeof(struct
stat_ip));
            /* обнуляется содержимое элемента с номером
index */
             memset(new_list[index],0,sizeof(struct
stat_ip));
            /* новому элементу присваивается IP-адрес
*/
             new_list[index]->ip=ip_src;
            /* задаётся новый массив list и
увеличивается размер списка */
             stat_list.list=new_list;
             stat_list.size++;
            /* удаление старого массива ссылок */
             free(old_list);
           /* Примечание: Такой порядок действий, когда
```

```
старый массив удаляется уже после обновления списка и
использование временных переменных необходимы, так как
в программе присутствует функция signal_handler. Она
вызывается в момент срабатывания события USR1 и
прерывает выполнение программы в её текущем состоянии
для вывода статистики по IP-адресам. При этом
необходимо быть уверенным, что все переменные списка
stat_list доступны и верны. */
       }

      /* выбор действий в зависимости от типа пакета */
      switch(ip_p) {
        case IPPROTO_TCP: // для TCP пакета
            /* прибавляется к статистике по IP-адресу
*/
            /* один TCP пакет и его размер */
            stat_list.list[index]->tcp++;
            stat_list.list[index]->tcp_size+=ip_len;
            break;
        case IPPROTO_UDP: // для TCP пакета
            /* прибавляется к статистике по IP-адресу
*/
            /* один UDP пакет и его размер */
            stat_list.list[index]->udp++;
            stat_list.list[index]->udp_size+=ip_len;
            /* определяется атака UDP-flood */
            /* прибавляем один пакет к статистической
переменной, */
            /* определяющей количество пришедших UPD
пакетов, */
            /* со времени прошлого блокирования IP-
адреса */
             stat_list.list[index]->udp_attack++;
            /* если количество таких пакетов больше 20,
*/
            /* то вероятно это атака */
             if(stat_list.list[index]->udp_attack > 20){
                  /* в системный журнал записывается
оповещение */
                  /* о блокировке IP-адреса */
                   syslog(LOG_WARNING,"Ban (UDP-flood
attack) %s", inet_ntoa(stat_list.list[index]->ip));
                  /* переменная udp_attack обнуляется
*/
                   stat_list.list[index]->udp_attack=0;
             }
            break;
```

```
        default: // для других типов пакетов
            /* прибавляется к статистике по IP-адресу
*/
            /* один пакет отличный от TCP и UDP и его
размер */
            stat_list.list[index]->other++;
            stat_list.list[index]->other_size+=ip_len;
      }
}

/* Завершено описание списка. */

/* определение основных функций программы */
void got_packet(u_char *args, const struct pcap_pkthdr
*header, const u_char *packet);

void print_app_banner(void);

void print_app_usage(void);

/* реализация основных функций программы */
/* app name/banner. Информация о программе выводится в
консоль при запуске. */
void print_app_banner(void)
{
      printf("%s - %s\n", APP_NAME, APP_DESC);
      printf("%s\n", APP_COPYRIGHT);
      printf("%s\n", APP_DISCLAIMER);
      printf("\n");
return;
}

/* Функция выводит на консоль информацию по использованию
*/
/* программы и передаваемых аргументах. */
void print_app_usage(void)
{
      printf("Usage: %s [interface]\n", APP_NAME);
      printf("\n");
      printf("Options:\n");
      printf("    interface    Listen on <interface> for
packets.\n");
      printf("\n");
return;
}

/* Функция, в которую поступает информация о каждом
```

```
пришедшем пакете. */
void got_packet(u_char *args, const struct pcap_pkthdr
*header, const u_char *packet)
{
       /* declare pointers to packet headers */
       const struct sniff_ip *ip;              /* The IP
header */

       int size_ip; // размер IP заголовка пакета

       /* define/compute ip header offset */
      ip = (struct sniff_ip*)(packet + header_size);
      size_ip = IP_HL(ip)*4;
      /* если размер заголовка пакета меньше 20 байт, */
      /* то пакет с ошибкой и не будет обработан */
      if (size_ip < 20)return;

      /* добавляем информацию о пакете в список stat_list
*/
      stat_set(ip->ip_src,ip->ip_p,ip->ip_len);
return;
}

/* обработчик сигналов */
/* Функция реализует один из вариантов межпроцессорного
взаимодействия, путём отправки программе сигнала. Это
может быть сигнал закрытия или отключения от консоли, но
в данном случае это сигнал USR1, который мы будем
отправлять самостоятельно, чтобы получить статистику по
IP-адресам в любое время, не останавливая сбор
статистики. */
void signal_handler(int sig)
{
      /* проверяется какой сигнал пришёл программе */
       switch(sig){
             /* если пришёл сигнал USR1, то */
              case SIGUSR1:{
                   /* описываются временные переменные
*/
                    unsigned long i, tcp=0, tcp_size=0,
udp=0, udp_size=0,
                        other=0, other_size=0;
                   /* цикл по всем элементам списка
stat_list */
                    for(i=0;i<stat_list.size;i++){
                     /* в консоль выводится вся
собранная статистика по */
```

```
                    /* пакетам с одного IP-адреса в
каждую итерацию цикла */
                        printf("%s %u %u %u %u
%u\n",

      inet_ntoa(stat_list.list[i]->ip),
                            stat_list.list[i]->tcp,
                            stat_list.list[i]-
>tcp_size,
                            stat_list.list[i]->udp,
                            stat_list.list[i]-
>udp_size,
                            stat_list.list[i]-
>other,
                            stat_list.list[i]-
>other_size
                        );
                    /* суммируется статистика по
всем IP-адресам */
                        tcp+=stat_list.list[i]->tcp;;
                        tcp_size+=stat_list.list[i]-
>tcp_size;
                        udp+=stat_list.list[i]->udp;
                        udp_size+=stat_list.list[i]-
>udp_size;
                        other+=stat_list.list[i]-
>other;

      other_size+=stat_list.list[i]->other_size;
                }
            /* выводится общая статистика по пакетам
для всех IP-адресов*/
                printf("%s %u %u %u %u %u %u\n",
"TOTAL", tcp, tcp_size, udp, udp_size, other,
other_size);
                break;
            }
      }
}

/* Функция main выполняется при запуске программы. В неё
передаются аргументы запуска программы. */
int main(int argc, char **argv)
{
      /* программа запрашивает доступ к отправке */
      /* своих сообщений в системный журнал */
      openlog(APP_NAME,     LOG_PID|LOG_CONS|LOG_NDELAY,
```

```
LOG_USER);

       /* функция signal_handler устанавливается */
       /* в качестве обработчика сигнала USR1 */
       struct sigaction act;
       memset(&act, 0, sizeof(act));
       act.sa_handler = signal_handler;
       sigset_t set;
       sigemptyset(&set);
       sigaddset(&set, SIGUSR1);
       act.sa_mask = set;
       sigaction(SIGUSR1, &act, 0);

       char *dev = NULL;                /* capture device
name */
       char errbuf[PCAP_ERRBUF_SIZE];   /*  error  buffer
*/
       pcap_t *handle;                  /* packet capture
handle */

       char filter_exp[256];            /*         filter
expression [3] */
       struct bpf_program fp;           /*       compiled
filter program (expression) */
       bpf_u_int32 mask;                /* subnet mask */
       bpf_u_int32 net;                 /* ip */

       print_app_banner();   //   в   консоль   выводится
информация о программе

       /* check for capture device name on command-line
*/
      if (argc == 2) { // если переданы два аргумента
             /* если в качестве аргумента программе
переданы -h или -help, */
             /* то выводится информация об
использовании программы */
             if(!strcmp(argv[1],"-h") ||
!strcmp(argv[1],"--help")){
                    print_app_usage();
                    exit(0);
             }
             dev = argv[1];
       }else if (argc > 2) { // если передано более двух
аргументов,
             // то выход с выводом ошибки
             fprintf(stderr, "error: unrecognized
```

```
command-line options\n\n");
              print_app_usage();
              exit(EXIT_FAILURE);
       }else {
              /* find a capture device if not specified
on command-line */
              dev = pcap_lookupdev(errbuf);
              if (dev == NULL) {
                     fprintf(stderr, "Couldn't find
default device: %s\n",
                         errbuf);
                     exit(EXIT_FAILURE);
              }
       }

       /* get network number and mask associated with
capture device */
       if (pcap_lookupnet(dev, &net, &mask, errbuf) == -
1) {
              fprintf(stderr, "Couldn't get netmask for
device %s: %s\n",
                  dev, errbuf);
              net = 0;
              mask = 0;
       }

       /* составляется фильтр для libpcap с учётом IP-
адреса компьютера*/
       int sock = socket(AF_INET, SOCK_DGRAM, 0); //
создаётся сокет
       /* заполняется структура для получения IP-адреса
*/
       struct ifreq ifr={.ifr_addr.sa_family = AF_INET};
       strncpy(ifr.ifr_name, dev, IFNAMSIZ-1); //
копирование имени интерфейса

       /* попытка получить IP-адрес на выбранном
интерфейсе */
       if(!ioctl(sock, SIOCGIFADDR, &ifr)){
              strcpy(filter_exp,"ip dst host ");
              strcat(filter_exp,
                     inet_ntoa(((struct sockaddr_in
*)&ifr.ifr_addr)->sin_addr));
       }else strcpy(filter_exp,"ip");

       close(sock); // закрытие сокета
```

```
        /* print capture info */
        printf("Device: %s\n", dev);
        printf("Filter expression: %s\n", filter_exp);

        /* open capture device */
        handle = pcap_open_live(dev, SNAP_LEN, 1, 1000,
errbuf);
        if (handle == NULL) {
                fprintf(stderr, "Couldn't open device %s:
%s\n", dev, errbuf);
                exit(EXIT_FAILURE);
        }

        /* make sure we're capturing on an Ethernet or
Linux SLL device */
        int dl=pcap_datalink(handle);
        /* в зависимости от оборудования устанаваливается
*/
        /* значение размера заголовка пакетов */
        if(dl == DLT_EN10MB){
                header_size=SIZE_ETHERNET;
        }else if (dl == DLT_LINUX_SLL){
                header_size=SIZE_LINUX_SLL;
        }else{
                fprintf(stderr, "%s is not an Ethernet or
Linux    SLL.    This    is    %d    type.\n",    dev,
pcap_datalink(handle));
                exit(EXIT_FAILURE);
        }

        /* compile the filter expression */
        if (pcap_compile(handle, &fp, filter_exp, 0, net)
== -1) {
                fprintf(stderr, "Couldn't parse filter %s:
%s\n",
                    filter_exp, pcap_geterr(handle));
                exit(EXIT_FAILURE);
        }

        /* apply the compiled filter */
        if (pcap_setfilter(handle, &fp) == -1) {
                fprintf(stderr,  "Couldn't  install  filter
%s: %s\n",
                    filter_exp, pcap_geterr(handle));
                exit(EXIT_FAILURE);
        }
```

```
        /* now we can set our callback function */
        pcap_loop(handle, -1, got_packet, NULL);

        /* cleanup */
        pcap_freecode(&fp);
        pcap_close(handle);
       unsigned                    int                    i;
for(i=0;i<stat_list.size;i++)free(stat_list.list[i]);
        free(stat_list.list);
        closelog();
return 0;
}
```

Необходимо сохранить текст программы в файл с именем ~/netmon/netmon.c и провести компиляцию исходного кода в Linux программу. Для этого установите дополнительный пакет с исходными текстами библиотеки Pcap:

```
apt-get install libpcap-dev
```

Все остальные необходимые библиотеки и компиляторы ставятся по умолчанию при установке базовой системы Debian. Перейдите в директорию с файлом netmon.c и выполните компиляцию программы:

```
gcc -march=native -O2 -lpcap netmon.c -o netmon
```

gcc – это название компилятора,
-O2 – опция, позволяющая оптимизировать программу для производительности,
-march=native – компиляция производится с учётом особенностей архитектуры компьютера на котором производится,
-lpcap – подключение библиотеки Pcap,
-o netmon – задаёт имя программы после компиляции.

Компиляция должна завершиться без ошибок и предупреждений. Столько небольшая программа компилируется всего несколько секунд, после чего командный интерфейс консоли Linux снова ждёт ввода следующих команд.

Проверим, что программа действительно скомпилировалась, и файл netmon был создан:

```
find netmon
```

В случае если программа на своём месте, то вы увидите её имя, в противном случае ошибку. Для запуска программы используется следующая команда:

```
./netmon
```

По умолчанию программа автоматически определит первый доступный сетевой интерфейс и запустит его мониторинг. Если на вашем компьютере несколько активных интерфейсов, то укажите тот, который нужно мониторить:

```
./netmon eth1
```

После запуска программы на консоли вы увидите следующую информацию:

```
netmon - Network monitor
Copyright (c) 2005 The Tcpdump Group, 2014 Evgeny Sagatov
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.

Device: eth0
Filter expression: ip dst host 192.168.2.1
```

Сначала информацию о программе, затем выбранный сетевой интерфейс и значение фильтра Pcap – в данном случае выбран мониторинг только IP пакетов, адресованных именно на наш интерфейс. На вашем компьютере вероятно IP-адрес будет другим. Программа перешла в режим работы, приглашения для ввода команд нет, ввод с клавиатуры возможен, но игнорируется. Завершить программу можно сочетанием клавиш Ctrl+C.

Оставим программу собирать статистику, а сами попробуем запустить другую консоль и сделать запрос статистики. Если у вас запущена графическая среда, можно просто открыть ещё одно окно консоли. Либо выполнить переход из режима рабочего стола в режим одной из виртуальных консолей с помощью сочетания клавиш Ctrl + Alt + F1..F6, где F1-F6 соответствует номеру консоли. Обратно в графический режим можно переключиться с помощью сочетания клавиш Ctrl + F7. Переход между виртуальными консолями выполняется клавишами Ctrl + F1-F6.

В итоге в другой консоли мы можем выяснить идентификатор процесса нашей программы:

```
ps ax | grep "[n]etmon" | egrep -o "^[ 0-9]+"
```

Зная идентификатор процесса можно отправлять программе различные команды, например, для закрытия или в нашем случае для вывода статистики. Для этого необходимо использовать команду kill:

```
kill 12345 # закрывает процесс с идентификатором 12345
kill –USR1 12345 # отправляет сигнал USR1 процессу с
идентификатором 12345
# следующая команда автоматически находит идентификатор
```

```
# процесса netmon и отправляет ему сигнал USR1
kill -s USR1 `ps ax | grep "[n]etmon" | egrep -o "^[ 0-
9]+"`
```

При выполнении последней команды на экран будет выведена статистическая таблица:

```
~/netmon# ./netmon eth1
netmon - Network monitor
Copyright (c) 2005 The Tcpdump Group, 2014 Evgeny Sagatov
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.

Device: eth1
Filter expression: ip dst host 10.7.163.13
192.168.0.1 16 283648 0 0 258282 1968611032
81.28.160.1 0 0 230 7434038 0 0
10.4.203.13 0 0 1 37120 0 0
10.15.255.33 556 10201995 0 0 0 0
10.1.197.46 8 118530 3 36864 0 0
10.5.32.88 0 0 6 73728 0 0
81.28.160.111 0 0 184 5885803 0 0
10.2.135.145 0 0 1 37120 0 0
10.6.128.164 0 0 18 170514 0 0
10.245.193.201 0 0 2 29696 0 0
10.1.66.203 0 0 1 11777 0 0
10.13.72.204 0 0 6 244992 0 0
10.0.128.250 26 376832 6 73728 0 0
TOTAL 606 10981005 458 14035380 258282 1968611032
```

В таблице приведён список IP-адресов, с которых были получены пакеты на адрес компьютера. Через пробел к каждому адресу указано количество пакетов разного типа и количество полученной информации:

1. IP-адрес
2. Количество принятых TCP пакетов
3. Количество принятых байт информации в TCP пакетах
4. Количество принятых UDP пакетов
5. Количество принятых байт информации в UDP пакетах
6. Количество принятых пакетов не UDP или TCP
7. Количество принятых байт информации в пакетах не UDP или TCP

Последней строкой указана статистическая сумма для всех IP-адресов.

Если с какого-то из IP-адресов пришло более 20 UDP пакетов,

то информация об этом должна была попасть в журнал user.log. Убедиться в этом можно следующей командой:

```
cat /var/log/user.log | grep netmon
```

Вывод должен быть примерно таким:

```
Sep 14 02:17:38 debian netmon[30341]: Ban (UDP-flood attack) 81.28.160.65
Sep 14 02:19:15 debian netmon[30341]: Ban (UDP-flood attack) 81.28.160.2
Sep 14 02:21:32 debian netmon[30341]: Ban (UDP-flood attack) 81.28.160.5
Sep 14 02:22:59 debian netmon[30341]: Ban (UDP-flood attack) 81.28.167.128
Sep 14 02:25:03 debian netmon[30341]: Ban (UDP-flood attack) 10.13.72.204
```

В настоящий момент программа только выводит предупреждения в журнал, процесс блокировки будет подробно рассмотрен в следующем пункте работы.

Внимательный студент, вспомнив таблицы NetFlow, заметит, что они так же содержат в себе информацию по типам пакетов и объёму переданных данных для завершившихся потоков. Приведём скрипт, который из архива NetFlow формирует статистическую таблицу по IP-адресам, аналогичную программе netmon.

```
<?php // php netflow-to-netmon.php /var/nfdump/ //Пример использования скрипта

if ($argc!=2) exit("Неверное число параметров!\n"); //Проверяется передано ли нужное количество параметров

$from=$argv[1]; //Путь к архиву NetFlow заносится в переменную $from

$files=scandir($from);

//Инициализация массива IP-адресов
$ips=array();

//Инициализация переменных суммарной статистики
$tcp=0; $tcp_size=0; $udp=0; $udp_size=0; $other=0; $other_size=0;

foreach ($files as $file){ //Цикл по всем файлам в директории с архивом
  //Проверка соответствует ли имя файла регулярному выражению $rgx
```

```
  $rgx='/^nfcapd\.201\d[01]\d[0-3]\d[0-2]\d[0-
5]\d\.txt$/';
  if(preg_match($rgx,$file)){
    $file_handle=@fopen($from.$file, "r"); //Открывается
файл для чтения
    if($file_handle){ //Проверяется удалось ли открыть
файл
      //Цикл по всем строкам в файле
      while (($buf=fgets($file_handle,1024))!==false) {
        $rgx='/^201\d-[01]\d-[0-3]\d [0-2]\d:[0-5]\d:[0-
5]\d\.\d\d\d              +\d+\.\d\d\d              (\w+)
+(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}):\d{1,5}       +-\>
+(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}):\d{1,5}     +(\d+)
+(\d+) +(\d+)$/';
        //Разбор строки файла по регулярному выражению
$rgx
        //$res[1] - протокол
        //$res[2] - IP-адрес источника
        //$res[3] - IP-адрес назначения
        //$res[4] - количество пакетов в потоке
        //$res[5] - количество переданных байт в потоке
        if(preg_match($rgx,$buf,$res)){
          //Фильтр потоков адресованных на сервер
          if($res[3]=='IP-АДРЕС-СЕРВЕРА'){
            //Если IP-адрес источника уже существует в
массиве $ips, то
            if(isset($ips[$res[2]])){
              if($res[1]=='TCP'){//Если протокол потока
TCP, то
                //количество пакетов и байт информации
добавляются в массив
                $ips[$res[2]][0]+=$res[4];
                $ips[$res[2]][1]+=$res[5];
              }else if($res[1]=='UDP'){ //Если протокол
потока UDP, то
                $ips[$res[2]][2]+=$res[4];
                $ips[$res[2]][3]+=$res[5];
              }else{ //Если протокол потока не TCP и не
UDP, то
                $ips[$res[2]][4]+=$res[4];
                $ips[$res[2]][5]+=$res[5];
              }
            }else{//Если IP-адреса источника ещё нет в
массиве
              if($res[1]=='TCP'){ //Если протокол потока
TCP, то
                //количество пакетов и байт информации
```

```
                //добавляются в новый элемент массива
                $ips[$res[2]]=ar-
ray($res[4],$res[5],0,0,0,0);
              }else if($res[1]=='UDP'){ //Если протокол
потока UDP, то
                $ips[$res[2]]=ar-
ray(0,0,$res[4],$res[5],0,0);
              }else{ //Если протокол потока не TCP и не
UDP, то
                $ips[$res[2]]=ar-
ray(0,0,0,0,$res[4],$res[5]);
              }
            }
            //Обновление общей статистики пакетов
            //и количества информации по протоколам
            if($res[1]=='TCP'){
              $tcp+=$res[4];
              $tcp_size+=$res[5];
            }else if($res[1]=='UDP'){
              $udp+=$res[4];
              $udp_size+=$res[5];
            }else{
              $other+=$res[4];
              $other_size+=$res[5];
            }
          }
        }
      }
      //Если цикл чтения прерван, а конец файла ещё не
достигнут
      if (!feof($file_handle)) {
        //Выводится ошибка чтения файла
        echo "Error: unexpected fgets() fail\n";
      }
      fclose($file_handle); //Закрывается ранее открытый
файл
    }
  }
}

//Вывод статистики
//Проход по всем элементам массива $ips
foreach ($ips as $ip => $v) {
       echo "$ip $v[0] $v[1] $v[2] $v[3] $v[4] $v[5]\n";
}

//Вывод итоговой суммы по всем IP-адресам
```

```
echo "TOTAL $tcp $tcp_size $udp $udp_size $other
$other_size\n";
```

Теперь есть возможность сравнить показания обеих программ на одинаковом отрезке времени. Помните, что показания NetFlow запаздывают минимум на минуту и расчётный период необходимо сдвинуть соответствующим образом.

4.6. Формирование пользовательского ядра и использование списков доступа

В прошлых пунктах главы были рассмотрены способы построения базы данных IP-адресов пользовательского ядра веб-сайта на основе запросов к веб-серверу. Была создана архитектура для обнаружения IP-адресов атакующих компьютеров.

Пользовательское ядро сайта состоит из IP-адресов, с которых заходили посетители, за какой-то продолжительный промежуток времени. За это время количество новых IP-адресов должно стабилизироваться между 30% и 40% от общего количества уникальных IP-адресов за сутки. Для проведённых авторами исследований в конкретном случае стабилизация наступала через шесть недель (см. рисунок 4.8).

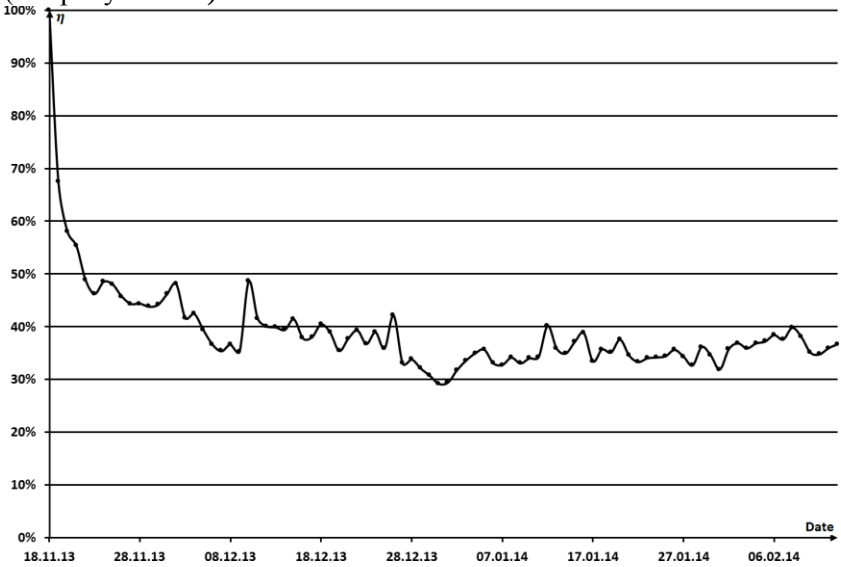

Рисунок 4.8 - График зависимости доли новых IP адресов за сутки от времени, пошедшего с начала измерений

Провайдеры последней мили чаще всего выдают пользователям IP-адреса в момент их входа в сеть из динамического пула своих адресов. Как правило, такие пользователи при каждом входе получают разные адреса, но из чётко ограниченного множества, принадлежащего провайдеру. Такие адреса называются динамическими.

Если более 30% такого динамического пула попали в базу данных пользовательского ядра, то необходимо все адреса пула добавить в БД, так как с них, вероятно, будут заходить благонадёжные клиенты. То есть необходимо произвести дополнение пользовательского ядра.

Адреса, с которых за всё время исследования были посещения в течение только одних суток, и не состоящие ни в одном пуле динамических IP-адресов, будем называть случайными. Эти адреса необходимо исключить из БД.

Обобщённый алгоритм дополнения БД выглядит следующим образом. Все IP-адреса БД разбиваются на подсети класса C (маска 255.255.255.0 или /24). Проверяется, заполнена ли эта подсеть адресами из БД хотя бы на 30%. Если заполнена, то вся подсеть вносится в БД. Если не заполнена, то эта подсеть разбивается на две подсети по маске /25 и по тому же алгоритму проверяется каждая отдельно. Разбивка и проверка продолжаются до сетевой маски /29 включительно (это сеть из 8 IP-адресов).

На практике в скрипте составляется двумерный массив, в котором в качестве первого ключа используются первые три октета IP-адреса. Второй ключ может быть от 0 до 31 (всего 32 значения) и представляет собой подсети /29. Каждая ячейка содержит количество IP-адресов в БД, которые попадают в эту подсеть. Просуммировав ячейки с 0 по 31 получим количество IP-адресов входящих в подсеть /24, с 0 по 14 – в первую подсеть /25, с 15 по 31 – во вторую подсеть /25 и т.д.

Для каждой подсети класса C вызывается рекурсивная функция proc, которая проверяет есть ли в подсети нужное количество IP-адресов. Если есть, то возвращает сеть и её маску, а если нет, то вызывает сама себя, в качестве аргументов передавая сначала одну подсеть /25, затем вторую. Для подсети /25 проводится аналогичная операция вплоть до /29. Затем объединяются и возвращаются

все получившиеся подсети. Ниже приведён скрипт, проводящий дополнение БД IP-адресами подсетей.

```
<?php // php bd-refill.php 2014-06-20 //Пример использо-
вания скрипта

if($argc!=2)   exit("Неверное   число   параметров!\n");
//Проверяется передано ли нужное количество параметров

$nets=array(); //Массив сетей класса C, составленный из
всех IP-адресов посетителей

$file_handle = fopen($argv[1].".all", "r"); //Открыва-
ется БД IP-адресов для чтения
if($file_handle) { //Проверяется открыта ли БД
    //Цикл по всем IP-адресам в БД
    while(($data=fgets($file_handle, 20))!==false) {
      //IP-адрес  (192.128.5.16)  разбивается  на  сеть
класса C (192.168.5) и номер хоста (16)
      if(preg_match
('/^(\d{1,3}\.\d{1,3}\.\d{1,3})\.(\d{1,3})/',     $data,
$res)){
            $ind=$res[1]; //Сеть класса C берётся в ка-
честве индекса в массиве $nets
            $val=$res[2]; //Номер хоста извлекается в
переменную $val
            //Вычисляется, к какой подсети /29 в массиве
принадлежит хост
            $net29=ceil(($val+1)/8)-1;
            if(!isset($nets[$ind])){   //Если   подсеть
класса C не существует в массиве
                  //Инициализируется   нулями   массив
подсетей класса /29

      $nets[$ind]=array(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0);
            }
            $nets[$ind][$net29]++; //В подсеть /29 до-
бавляется один хост
      }
    }
    if(!feof($file_handle)) { //Если предыдущий цикл за-
кончился, а конец файла всё ещё не достигнут, значит
произошла ошибка чтения из файла.
       echo   "Error:   unexpected   fgets()   fail\n";
//Выводится ошибка
    }
```

```
    fclose($file_handle); //Закрывается файл БД
}else exit("Не удалось открыть файл!\n"); //Выводится
ошибка, если не удалось открыть файл БД

// Рекурсивная функция, которая проверяет, есть ли в под-
сети 30% и более IP-адресов.
// $net – одна сеть класса C из массива $nets, передаётся
по ссылке
// $from – с какого элемента массива $net начинать подсчёт
// $to – на каком элементе массива $net подсчёт заверша-
ется
// $net_c – в строковом виде передаётся подсеть класса C
function proc(&$net,$from,$to,$net_c){
      $hosts=0; //Инициализация количества хостов в под-
сети
      // Цикл, который суммирует все хосты в подсетях
/29, которые входят в исследуемую подсеть
      for($i=$from;$i<=$to;$i++){
            $hosts+=$net[$i];
      }
      // Вычисление минимального количества хостов для
внесения подсети в БД.
      // Это 30% размера подсети.
      $need_hosts=floor(((float)(($to+1)-
$from))*8.0*0.3);
      // Если реальная заполненность подсети больше 30%,
то
      if($hosts>=$need_hosts){
            // Возвращается подсеть с маской, например
192.168.5.128/25.
            $conv=ar-
ray(31=>24,15=>25,7=>26,3=>27,1=>28,0=>29);
            return  $net_c.'.'.($from*8).'/'.$conv[$to-
$from]."\n";
      }else{
            // Выход из функции, если достигнута мини-
мальная подсеть /29
            if($from==$to)return '';
            // Ищется серединный элемент подсети, т.е.
разбиение на две подсети
            $middle=$from+floor(($to-$from)/2);
            // Функция proc вызывается сначала для одной
меньшей подсети, затем
            // для второй. Результат объединяется и воз-
вращается.
            $ret=proc($net,$from,$middle,$net_c);
```

```
            return                  $ret.proc($net,$mid-
dle+1,$to,$net_c);
        }
}

// Создаётся новый файл с расширением *.nets и открыва-
ется для записи
$file_handle = fopen($argv[1].".nets", "w");

// Цикл по всем сетям класса C массива $nets
// $net_c – сеть класса C в виде строки
// $nets29 – массив количества хостов в сети класса C с
разбивкой на подсети /29
foreach ($nets as $net_c => $nets29) {
        // Вызывается функция proc для подсети класса C
        $ret=proc($v,0,31,$nets29);
        // Если функция нашла заполненные на 30% и более
подсети, то сохраняем их в открытый файл *.nets
        if($ret!='')fwrite($file_handle,$ret);
}

fclose($file_handle); //Закрывается открытый файл с се-
тями
```

После выполнения прошлых пунктов работы у вас на компьютере должна была сформироваться посуточная база данных IP-адресов. Это файлы с расширениями *.ip, *.sort, *.all, *.new, *.count и *.assigned. Для чего нужен тот или иной файл вы сможете понять в следующих пунктах данной главы. Для скрипта необходимо выбрать самую старшую дату и выполнить:

```
php bd-refill.php 2014-06-20
```

При этом скрипт создаст ещё один файл 2014-06-20.nets, в котором будут перечислены все найденные по вышеописанному алгоритму подсети. Он выглядит следующим образом:

```
100.43.81.0/28
100.43.83.136/29
101.170.213.56/29
101.171.213.64/29
101.199.108.48/28
101.199.112.48/28
101.226.166.128/25
101.226.167.128/25
101.226.168.128/25
...
```

Этот список, согласно проведённым авторами исследованиям,

включает в себя ~70% всех IP-адресов, попавших в БД. Именно он является пользовательским ядром, а соответственно списком доступа к веб-серверу в момент DDoS-атаки. Но кроме подсетей в БД остались ещё индивидуальные IP-адреса, которые тоже должны быть включены в пользовательское ядро. Для того чтобы их вычислить, необходимо развернуть файл сетей *.nets до IP-адресов, а затем найти IP-адреса из файла БД *.all, которые не встречаются в этом списке.

Сконвертировать список подсетей в IP-адреса поможет следующий скрипт:

```
<?php //php nets-to-ips.php 2014-06-20 //Пример исполь-
зования скрипта

if($argc!=2)   exit("Неверное   число   параметров!\n");
//Проверяется передано ли нужное количество параметров

// Массив соответствия сети количеству хостов в ней
$net_to_ip_cnt=ar-
ray(24=>256,25=>128,26=>64,27=>32,28=>16,29=>8);

// Открывается для чтения файл сетей
$input_file = fopen($argv[1].'.nets', "r");
if($h===NULL)exit("Не      удалось      открыть      файл
".($argv[1]).".nets!\n");

// Создаётся и открывается для записи файл IP-адресов,
входящих в сети
$output_file = fopen($argv[1].'.ipnets', "w");
if($h===NULL)exit("Не      удалось      открыть      файл
".($argv[1]).".ipnets!\n");

// Цикл по всем подсетям в файле подсетей
while(($data=fgets($input_file, 50))!==false) {
      // Строка из файла разбивается на два значения:
сеть и маску сети
      if                  (                  preg_match
('/^(\d{1,3}\.\d{1,3}\.\d{1,3}\.)(\d{1,3})\/(\d\d)/',
$data, $res) ) {
      // В файл IP-адресов записываются все адреса, вхо-
дящие в выбранную подсеть
            for($i=0;$i<$net_to_ip_cnt[$res[3]];$i++){

      fwrite($ouput_file,$res[1].($res[2]+$i)."\n");
            }
```

```
        }
}
if(!feof($input_file)) { //Если предыдущий цикл закончился, а конец файла всё ещё не достигнут, значит произошла ошибка чтения из файла.
        echo "Error: unexpected fgets() fail\n";
//Выводится ошибка
}

fclose($input_file); // Закрывается файл подсетей
fclose($output_file); // Закрывается файл IP-адресов
```

После выполнения скрипта:

```
php nets-to-ips.php 2014-06-20
```

будет создан новый файл 2014-06-20.ipnets, который будет содержать все IP-адреса, входящие в подсети из файла 2014-06-20.nets. Получить все IP-адреса из БД, которые не вошли в найденные подсети теперь можно одной bash командой:

```
cat 2014-06-20.all 2014-06-20.ipnets 2014-06-20.ipnets | sort | uniq –u > 2014-06-20.ipwonets
```

Команда uniq –u на вход должна получить отсортированный список строк, а на выходе будут только те строки, которые не повторяются. Так список подсетей – это, по сути, дополненный список всех адресов БД, то в нём встречаются IP-адреса, которых нет в основном списке БД. Чтобы удалить их файл *.ipnets на вход подаётся дважды. Таким образом, в файл *.ipwonets будут сохранены только IP-адреса, которые есть в файле *.all, но отсутствуют в *.ipnets.

Теперь из этого “остатка” IP-адресов необходимо исключить случайные адреса. Это адреса, которые были замечены во время исследования в течение только одних суток. В прошлой части работы приводится способ формирования списка IP-адресов с указанием количества дней, в течение которых с него были обращения к веб-серверу. В результате создаётся файл с расширением *.assigned. Удалим случайные IP-адреса с помощью командной строки bash:

```
# Выбрать только IP-адреса, которые встречались одни сутки
cat 2014-05-01_2014-06-20.assigned | egrep "(^1 | 1 )" | grep -P -o "[\.\d]+$" > 2014-05-01_2014-06-20.casual

# Удалить случайные адреса из пользовательского ядра
```

```
cat   2014-06-20.ipwonets   2014-05-01_2014-06-20.casual
2014-05-01_2014-06-20.casual | sort | uniq –u > 2014-06-
20.ips
```

В получившимся файле 2014-06-20.ips будет список IP-адресов, которые не вошли в список сетей 2014-06-20.nets, но с них были запросы более чем за одни сутки. Именно эти два файла являются списками доступа, а также пользовательским ядром вебсайта.

Теперь разберёмся с тем, как именно нужно использовать списки доступа и блокировок. На рисунке 4.9 изображена принципиальная схема соединения сетей. Стрелочками показано направление трафика при DDoS-атаке, а ширина линий обозначает ширину Интернет-канала между двумя устройствами.

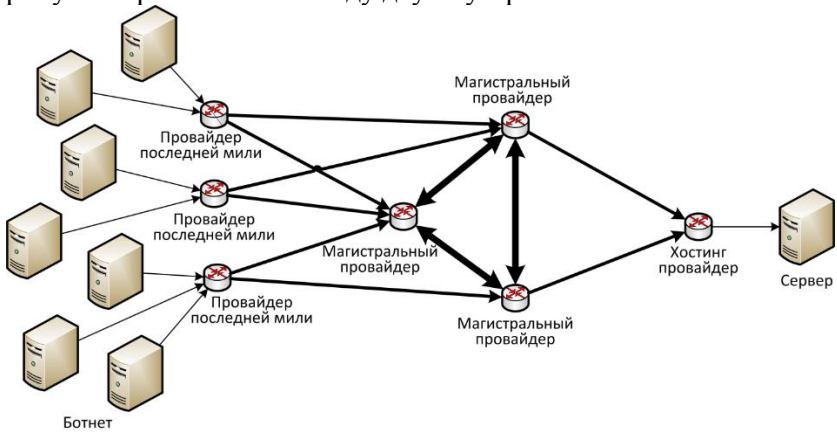

Рисунок 4.9 - Принципиальная схема DDoS-атаки

Из схемы можно понять, что провайдеры между собой соединены несколькими высокоскоростными каналами связи. Компьютеры из ботнет-сети, как правило, имеют подключение последней мили к вышестоящему провайдеру 1-100Мбит/с. За счёт своего количества эти боты создают много UDP-трафика, идущего к атакуемому серверу. Этот трафик забивает Интернет-канал сервера. Соответственно для отражения такой атаки, необходима блокировка IP-адресов у провайдеров, как можно ближе к самим ботам. В идеальном случае блокировка должна проводиться у провайдеров последней мили, тогда ненужный трафик вообще не будет попадать

в Интернет, и доходить до сервера. Но на практике чем административно дальше от сервера находится атакующий компьютер, тем сложнее его ограничивать. В настоящий момент необходимо иметь договорённость с хостинг провайдером, а лучше с вышестоящим магистральным провайдером о применении ваших списков доступа.

В рамках работы будет рассмотрена возможность применения списков доступа на рабочем компьютере в операционной системе Debian GNU/Linux 7.6.0. Таким способом можно отразить только сравнительно небольшую атаку.

Основным средством управления сетевым трафиком, а точнее межсетевым экраном NetFilter[8] в ОС Linux является утилита iptables. С её помощью системный администратор задаёт правила, исходя из которых, каким-то пользователям будет предоставлен доступ к веб-серверу, а каким-то будет отказано. В нашем случае основным критерием будет выступать IP-адрес.

Сразу необходимо отметить, что для того, чтобы пропустить или заблокировать каждый пакет от пользователя, ядру Linux придётся сравнить IP-адрес источника этого пакета с каждым IP-адресом в правилах, которое задал администратор. Например, в исследованиях, которые проводили авторы работы, получилась ~21 тысяча сетей и ~34 тысячи индивидуальных IP-адресов. Что составляет ~55 тысяч правил. Если учесть, что каждую секунду на канал в 100Мбит/с может приходить в среднем 15 тысяч запросов, то ядру Linux необходимо будет обработать 825 миллионов правил. Это слишком много и любой сервер уровня предприятия зависнет, обрабатывая сетевой поток.

Для решения вышеописанной проблемы был разработан модуль для NetFilter, который обеспечивает очень быстрое сравнение списка IP-адресов с заданным правилом. Это обеспечивается за счёт того, что модуль хранит не правила, которые необходимо последовательно проверять, а именно упорядоченные хэш списки адресов или сетей, поиск по которым происходит очень быстро. Для всех IP-адресов будет всего одно правило в iptables, которое будет сравнивать адрес в пришедшем пакете сразу со списком в ipset.
Для начала необходимо установить ipset:

```
apt-get install ipset
```

Теперь можно создавать списки. Утилита ipset поддерживает две нотации написания команд: как в iptables и в виде слов. Рассмотрим несколько команд более подробно:

```
# Создание списка хэшированных IP-адресов с именем ban-
list
ipset -N banlist iphash

# Другая запись предыдущей команды
ipset create banlist hash:ip

# В самое начала таблицы INPUT (входящие пакеты) добав-
ляется запись о том, что
# если адрес источника пакета содержится в списке ban-
list, то пакет отбрасывается
iptables -I INPUT -m set --set banlist src -j DROP

# В список вносятся два IP-адреса разными нотациями
ipset -A banlist 101.17.225.177
ipset add banlist 101.1.215.37

# Получить информацию о списке и IP-адреса входящие в
него
ipset -L banlist
ipset list banlist

# IP-адреса удаляются из списка
ipset -D banlist 101.17.225.177
ipset del banlist 101.1.215.37

# Очистка списка
ipset -F banlist
# или по другому
ipset flush banlist

# Удаление списка
ipset -X banlist
# или по другому
ipset destroy banlist

# Создание списка хэшированных IP-сетей с именем netban-
list
ipset -N netbanlist nethash
ipset create netbanlist hash:net

# В список вносятся две IP-сети разными нотациями
ipset -A netbanlist 101.17.225.0/24
```

```
ipset add netbanlist 101.1.215.0/26

# Сохраняет список сетей или IP-адресов в файл
ipset save netbanlist > netbanlist

# Считывает список сетей или IP-адресов из файла
ipset restore < netbanlist

# Пример списка, элементы в котором автоматически удаля-
ются через 5 минут (300 сек.)
ipset create banlist hash:ip timeout 300
# В список добавляется IP-адрес, но автоудаление этого
адреса произойдёт через 1 минуту
ipset add banlist 101.1.215.37 timeout 60
```

Если вы попробуете создать список адресов, внести в него несколько адресов, а затем сохранить этот список, то увидите файл следующего формата:

```
add banlist 101.17.225.47
add banlist 101.17.225.149
add banlist 101.17.225.98
add banlist 101.17.225.67
```

То есть сохранение происходит в том же формате, что вводит администратор для пополнения списка. Команда restore внесёт список IP-адресов значительно быстрее, так как ipset будет запущена всего один раз. Необходимо файлы ядра пользователей *.nets и *.ips привести к нужному формату. Для этого используем простые bash команды:

```
cat 2014-06-20.nets | sed -e 's/^/add access_nets /g' >
access.list
cat 2014-06-20.ips | sed -e 's/^/add access_ips /g' >>
access.list

# Добавьте так же используемые на компьютере DNS сервера
и другие необходимые
# для нормального функционирования IP-адреса, иначе они
будут заблокированы
# при активации списков доступа
echo 'add access_ips 8.8.8.8' >> access.list
...
```

Файл access.list будет содержать все IP-адреса пользовательского ядра, адаптированные для быстрой загрузки в ядро Linux. Таким образом, для включения списка доступа и блокировки остальных IP-адресов необходимо выполнить следующие команды:

```
# Создаются списки для сетей и IP-адресов
ipset create access_nets hash:net
ipset create access_ips hash:ip

# Списки считываются из файла
ipset restore < access.list

# Создаётся новая цепочка правил iptables
iptables -N access_list

# 1 выпустить из цепочки пакет, если адрес источника есть
в списке сетей access_nets
iptables -A access_list -m set --match-set access_nets
src -j RETURN

# 2 выпустить из цепочки пакет, если адрес источника есть
# в списке IP-адресов access_ips
iptables -A access_list -m set --match-set access_ips src
-j RETURN

# 3 все пакеты, что не дошли до этого шага отбрасываются
iptables -A access_list -j DROP

# Цепочка access_list ставится в самое начало цепочки
INPUT, в которую попадают
# все входящие пакеты
iptables -I INPUT -j access_list
```

Быстро отключить список можно командой:

```
iptables -D INPUT -j access_list
```

При этом списки и цепочка правил access_list всё ещё будут загружены в память, только пакеты из INPUT туда попадать не будут. Чтобы снова активировать списки нужно просто выполнить снова следующую команду:

```
iptables -I INPUT -j access_list
```

Для полного удаления списков и цепочек из памяти выполните следующий скрипт:

```
iptables -D INPUT -j access_list
iptables -F access_list
iptables -X access_list
ipset destroy access_ips
ipset destroy access_nets
```

Теперь вы знаете как создать список доступа на веб-сервере. Попробуйте по аналогии самостоятельно организовать стоплист

для скрипта и программы на C++, которые приведены ранее в главе для обнаружения сетевых атак.

Литература

[1] Adiscon GmbH. RSYSLOG. The rocket-fast system for log processing [Электронный ресурс]. URL: http://www.rsyslog.com/doc/master/index.html (дата обращения: 10.02.2017).
[2] Troan E., Brown P., Kolomeets V. Проект OpenNet: MAN logrotate (8) Команды системного администрирования (FreeBSD и Linux) [Электронный ресурс]. URL: http://www.opennet.ru/man.shtml?topic=logrotate&category=8&russian=0 (дата обращения: 10.02.2017).
[3] The Apache Software Foundation. Log Files - Apache HTTP Server Version 2.5: Common Log Format [Электронный ресурс]. URL: https://httpd.apache.org/docs/trunk/logs.html#common (дата обращения: 10.02.2017).
[4] The Apache Software Foundation. Log Files - Apache HTTP Server Version 2.5: Combined Log Format [Электронный ресурс]. URL: https://httpd.apache.org/docs/trunk/logs.html#combined (дата обращения: 10.02.2017).
[5] Haag P. NFDUMP [Электронный ресурс]. URL: http://nfdump.sourceforge.net/ (дата обращения: 10.02.2017).
[6] Čihař M. et al. phpMyAdmin: Bringing MySQL to the web [Электронный ресурс]. URL: http://www.phpmyadmin.net (дата обращения: 10.02.2017).
[7] Tcpdump/Libpcap. TCPDUMP/LIBPCAP public repository [Электронный ресурс]. URL: http://www.tcpdump.org/ (дата обращения: 10.02.2017).
[8] Welte H., Ayuso P.N. netfilter/iptables project homepage - The netfilter.org project [Электронный ресурс]. URL: http://www.netfilter.org/ (дата обращения: 10.02.2017).
[9] Kadlecsik J. et al. IP sets [Электронный ресурс]. URL: http://ipset.netfilter.org/ (дата обращения: 10.02.2017).

ОГЛАВЛЕНИЕ

Учебное издание

Гальцев Алексей Анатольевич,
Сагатов Евгений Собирович,
Сухов Андрей Михайлович,

Учебное пособие
Выявление аномальных сетевых состояний и причин их возникновения

Самарский национальный исследовательский университет.
443086 Самара, Московское шоссе, 34.