ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

В. А. НЕСТЕРЕНКО

МЕТОДЫ ОБНАРУЖЕНИЯ НАРУШЕНИЙ В СЕТИ

Ростов-на-Дону

2010

Оглавление

Введение

Предлагаемое учебное пособие предназначено для студентов факультета математики, механики и компьютерных наук изучающих курс «Обнаружение нарушений в сети». Представленный материал пособия может быть использован как для самостоятельного изучения, так и в качестве основного материала лекционного курса. При изучении учебного материала настоятельно рекомендуется выполнение приведённых в курсе лекций практических примеров, реализация и исполнение программ приведённых в материалах данного учебного пособия. Программы, приведённые в пособии, написаны на языке C и в основном не привязаны к конкретной версии компилятора языка, среде разработки или операционной системе. При проверке работоспособности программ был использован компилятор gcc, возможно использование другого удобного компилятора языка C.

Область научной деятельности, связанная с защитой информации и обнаружением нарушений информационной безопасности, является молодой и появилась относительно недавно (~ 20-25 лет). Эта область бурно развивается и сильно меняется в течение небольшого промежутка времени (один-два года). По этой причине хорошим дополнением к представленным материалам лекционного курса будет изучение материалов зарубежных и отечественных научных конференций посвящённых данной тематике. Разработка и использование новых методов обнаружения нарушений безопасности часто связана с привлечением и использованием материала других областей знаний, прежде всего математики и дисциплин связанных с программированием. Поэтому часть времени, предназначенного для самостоятельной работы по этому курсу, следует отвести на повторение и углублённое изучение материала сопутствующих дисциплин связанных с современ-

ными методами аналитической обработки данных (Data Mining) и поиска новых знаний в базах данных (Knowledge Discovery in Databases). Некоторые из этих методов будут использованы в данном курсе.

В общем случае решение задачи обнаружения нарушений в системе можно разбить на следующий (относительно независимые) этапы:

1. **Мониторинг и сбор информации в сети.** Выбор набора характеристик событий. Масштабирование (шкалирование) данных. Снижение размерности характеристик событий.
2. **Предварительная обработка результатов мониторинга**
3. **Выбор метода обнаружения нарушений и построение модели состояния системы.** Обработка данных в рамках используемой модели. Оценка качества работы системы обнаружения вторжений.
4. **Анализ полученных результатов.**

В соответствии с этими этапами расположен фактический материал учебника.

Мониторинг и сбор информации в сети

Введение в TCPdump

Пакет TCPdump - это основное инструментальное средство, используемое для сбора данных в сети, расшифровки собранной информации и вывод результатов в удобном для чтения виде. В большинстве случаев TCPdump служит основой для других пакетов сбора и анализа потока данных в сети. Пакет TCPdump включен в большинство дистрибутивов Linux, широко распространены и доступны версии TCPdump для Windows.

Программа TCPdump осуществляет перехват и вывод заголовков пакетов для указанного сетевого интерфейса в соответствии с заданными логическим выражением. Перечень доступных сетевых интерфейсов можно получить при использовании команды с ключом **-D**:

```
tcpdump -D
```

Результат исполнения команды может иметь вид:

```
1.eth0
2.any (Pseudo-device that captures on all interfaces)
3.lo
```

Последующий вызов **tcpdump** с указанием существующего интерфейса

```
tcpdump -i eth0
```

приведёт к захвату всех пакетов проходящих через заданный сетевой интерфейс. Отметим, что чтение пакетов из сетевого интерфейса может по-

требовать (в зависимости от используемой операционной системы) специальных привилегий. Так, например, в операционной системе Linux требуются полномочия **root** или установка флага **SUID** для программы **tcpdump**.

Возможно использование программы **tcpdump** с флагом **-w** для записи перехваченных пакетов в файл, который впоследствии можно использовать для анализа:

```
tcpdump -w File.log
```

Возможен просмотр заголовков пакетов из таких файлов:

```
tcpdump -r File.log
```

Во всех случаях использования (перехват из сетевого интерфейса или чтение из файла) **tcpdump** имеет дело только с пакетами соответствующими заданному логическому выражению - фильтру. Если фильтр не задан, то собираются все пакеты, проходящие через указанный сетевой интерфейс. Для указания признаков интересующих нас пакетов используются значения элементов в IP-пакете. Таким элементом может быть часть заголовка IP (например адрес источника), заголовка TCP (например флаги TCP), заголовка UDP (например порт приёмника), заголовка ICMP (например тип ICMP сообщения) или заголовки пакетов других типов (ARP, RARP, ...). Для обозначения заголовка каждого типа используются соответствующие имена. Так, например, фильтрацию пакетов по типу протокола можно выполнить следующим образом:

```
tcpdump <in> ip
tcpdump <in> tcp
tcpdump <in> udp
tcpdump <in> icmp
```

Эти команды позволяют отбирать IP-, TCP-, UDP- или ICMP-пакеты соответственно. Здесь и далее через **<in>** мы будем обозначать объект применения программы **tcpdump** (сетевой интерфейс или предварительно созданный файл с результатом работы **tcpdump**). Для создания фильтров по значениям отдельных полей заголовков пакетов заданного типа можно использовать выражение следующего формата:

```
<заголовок> [смещение:длина] <отношение> <значение>
```

Так, например, для отбора ICMP-пакетов фильтр задаётся следующим образом:

```
tcpdump <in> ip[9:1]=1
```

или

```
tcpdump <in> ip[9]=1
```

Здесь мы воспользовались тем фактом, что 9-й байт в заголовке IP-пакета задаёт тип инкапсулированного протокола, для протокола ICMP значение этого байта должно быть равно 1 (для TCP - 6, для UDP - 17). Для выделения широковещательных пакетов можно воспользоваться комбинацией двух выражений:

```
tcpdump <in> ip[19]=0xff or ip[19]=0x00
```

Исчерпывающая информация о структуре заголовков пакетов различных протоколов содержится в соответствующей справочной или учебной литературе (например У. Стивенсон [1]).

Для наиболее часто используемых полей существует большой набор

мнемонических обозначений. Например: **port** - для указания номера порта источника или приёмника пакета, **host** - указания IP-адреса или имени узла и т.д.. Фильтрацию пакетов по номерам портов источника и получателя можно выполнить следующим образом:

```
tcpdump <in> dst port <port>
tcpdump <in> src port <port>
```

Фильтрация пакетов по адресам источника и получателя:

```
tcpdump <in> dst <ip-address>
tcpdump <in> src <ip-address>
```

Выделение SYN-пакетов в TCP соединениях:

```
tcpdump <in> tcp[tcpflags]=tcp-syn
```

Допустимые обозначения флагов TCP-соединения:

```
tcp-fin, tcp-syn, tcp-rst, tcp-push, tcp-ack, tcp-urg
```

Возможны использования комбинаций флагов:

```
tcpdump <in> 'tcp[tcpflags]=(tcp-syn|tcp-ack)'
```

(при использовании в фильтре комбинаций различных условий сложные выражения следует заключать в одинарные или двойные кавычки). Также можно воспользоваться числовыми значениями для соответствующих констант:

```
tcp-fin = 0x01
tcp-syn = 0x02
```

```
tcp-rst = 0x04
tcp-push = 0x08
tcp-ack = 0x10
tcp-urg = 0x20
```

или комбинацией флагов:

```
tcp-syn|tcp-ack = 0x02|0x10 = 0x12 = 18
```

(любое из этих выражений может быть использовано для выделения TCP пакетов с одновременно установленными флагами SYN и ACK.)

Установлению TCP соединения, в соответствии с правилами "тройного рукопожатия", должна соответствовать упорядоченная по времени последовательность пакетов для двух фиксированных узлов и номеров портов:

```
tcpdump <in> 'tcp[tcpflags]=tcp-syn
              && src <ip1> && src port <ip1>
              && dst <ip2> && dst port <ip2>'
tcpdump <in> 'tcp[tcpflags]=(tcp-syn|tcp-ack)'
              && src <ip2> && src port <ip2>
              && dst <ip2> && dst port <ip2>'
tcpdump <in> 'tcp[tcpflags]&tcp-ack!=0
              && src <ip1> && src port <ip1>
              && dst <ip2> && dst port <ip2>'
```

Проверку заданной временной последовательности невозможно провести только средствами пакета **TCPdump**, для этих целей необходимо привлекать другие средства, рассмотрению такой возможности мы посвятим одну из следующих лекций.

Более подробно правила построения фильтров и возможные значения ключей для программы **tcpdump** изложены в соответствующих справочных руководствах (**man tcpdump** например).

Если программа **tcpdump** запущена без флага **-c** (этот флаг позволяет задавать количество пакетов, после получения указанного числа пакетов программа завершит свою работу) то она будет собирать пакеты до тех пор, пока её работа не будет прервана внешним вмешательством - нажатием **ctrl-C** например. После завершения захвата пакетов **tcpdump** выводит значения счётчиков для

- числа пакетов захваченных и обработанных программой
- число пакетов соответствующих условием фильтрации
- число пакетов отброшенных ядром системы по причине фильтрации или нехватки системных ресурсов.

Обнаружение аномалий при анализе заголовков пакетов

При изучении материала этой и последующих лекций мы будем использовать данные о потоке пакетов через один из серверов факультета механики, математики и компьютерных наук ЮФУ. Информация о всех захваченных в течении суток пакетах была предварительно сохранена в файле под именем **file.log**:

```
tcpdump -i all -w file.log
```

Общее число пакетов, сохранённых в файле, можно определить при помо-

щи команды **wc**:

```
   tcpdump -r file.log | wc -l
-> 1798554
```

Статистическая информация собранная на шлюзах, соединяющих внешнюю и локальную сети, показывает, что большинство пакетов приходящих из внешней сети являются IP пакетами; большинство из них являются пакетами TCP типа, доля UDP пакетов составляет около 2% всего трафика. По этой причине мы будем рассмотрим только IP- и TCP-пакеты в наблюдаемом трафике. Действительно, в нашем случае число IP-пакетов составляет:

```
   tcpdump -r file.log ip | wc -l
-> 1347893
```

или 75% от общего числа пакетов.

IP-пакеты распределены по типам следующим образом:

```
TCP:    tcpdump -r file.log tcp | wc -l
     -> 1271110 (94.3%)
UDP:    tcpdump -r file.log udp | wc -l
     -> 57771 (4.3%)
ICMP:   tcpdump -r file.log icmp | wc -l
     -> 1525 (0.1%)
прочее:
     -> 17487 (1.3%)
```

Теперь рассмотрим заголовки IP-пакетов и выделим заведомо неправильные пакеты:

Размер IP-заголовка должен быть больше или равен 20 октетам. Определим число пакетов с неправильным размером заголовка:

```
   tcpdump -r file.log 'ip[0]&0x0f < 5' | wc -l
-> 0
```

таких пакетов нет.

Выявим пакеты с неправильными IP адресами:

Совпадающие адреса источника и приёмника

```
   tcpdump -r file.log 'ip[12:4]=ip[16:4]' | wc -l
-> 0
```

Передача пакетов между "внешней" и "внутренней" сетями:

```
   tcpdump -r file.log 'src net 10 and not dst net 10'
           | wc -l
-> 4
   tcpdump -r file.log 'not src net 10 and dst net 10'
           | wc -l
-> 2212
   tcpdump -r file.log 'src net 192.168
                        and not dst net 192.168' | wc -l
-> 37
   tcpdump -r file.log 'not src net 192.168
                        and dst net 192.168' | wc -l
-> 36
```

Аналогичным образом попытаемся выявить пакеты с заведомо не-

правильными TCP заголовками.

Размер заголовка TCP пакета:

```
   tcpdump -r file.log 'tcp[12]&0xf0 < 0x50' | wc -l
-> 0
```

Номера портов приёмника и источника не должны совпадать:

```
   tcpdump -r file.log 'tcp[0:2]=tcp[2:2]' | wc -l
-> 0
```

Номера портов приёмника и источника не должны быть равны 0:

```
   tcpdump -r file.log 'src port 0 or dst port 0' | wc
-l
-> 2
```

Недопустимые комбинации TCP флагов:

```
   tcpdump -r file.log 'tcp[tcpflags]&tcp-syn!=0
                   and tcp[tcpflags]&tcp-urg!=0' | wc
-l
-> 0

   tcpdump -r file.log 'tcp[tcpflags]&tcp-syn!=0
                   and tcp[tcpflags]&tcp-push!=0' | wc
-l
-> 0

   tcpdump -r file.log 'tcp[tcpflags]&tcp-syn!=0
                   and tcp[tcpflags]&tcp-rst!=0' | wc
-l
-> 0
```

```
   tcpdump -r file.log 'tcp[tcpflags]&tcp-syn!=0
                   and tcp[tcpflags]&tcp-fin!=0' | wc
-l
-> 0
```

Разумеется, в реально работающей системе, выявление подобных аномальных пакетов должно осуществляться "на лету", в режиме реального времени, с последующим анализом причин появления подобных пакетов и соответствующей реакцией системы защиты на них.

Интересно будет посмотреть на разницу между числом TCP-пакетов с установленным SYN-флагом (запрос на TCP-соединение) и пакетов с установленными SYN-ACK-флагами (первое подтверждение о принятии TCP-соединения):

```
   tcpdump -r file.log 'tcp[tcpflags]=tcp-syn | wc -l
-> 14909

   tcpdump -r file.log 'tcp[tcpflags]=(tcp-syn|tcp-
ack) | wc -l
-> 14493
```

Разница составляет 416 пакетов - значительная величина. Попробуем проследить, как изменяется эта разница с увеличением числа захваченных пакетов. Для первых 100000 пакетов

```
   tcpdump -r file.log -w - -c 100000 |
   tcpdump -r - 'tcp[tcpflags]=tcp-syn | wc -l
-> 1941
```

```
   tcpdump -r file.log -w - -c 100000 |
   tcpdump -r - 'tcp[tcpflags]=(tcp-syn|tcp-ack) | wc
-l
-> 1963
```

эта разница невелика. Проведём подобные измерения с интервалом в 100000 пакетов и полученные результаты соберём в таблицу, для корректного сравнения результатов различных измерений мы будем использовать относительную разницу:

Общее число пакетов	Число SYN-пакетов	Число SYN-ACK -пакетов	Разница относительно SYN-пкт.
100000	1941	1963	-0.01
200000	2146	2168	-0.01
300000	2149	2171	-0.01
400000	2154	2176	-0.01
500000	2159	2181	-0.01
600000	2180	2202	-0.01
700000	2204	2226	-0.01
800000	3243	3211	0.01
900000	4989	4802	0.04
1000000	6922	6730	0.03
1100000	8650	8352	0.03
1200000	8650	8352	0.03
1300000	8742	8436	0.04
1400000	9714	9381	0.03
1500000	11301	10882	0.04
1600000	13834	13407	0.03
1700000	14626	14219	0.03
все	14909	14493	0.03

Приведённые результаты показывают, что относительная разница невелика (≈ 0.03) и, следовательно, является результатом каких-то случайных процессов в сети.

Анализ ещё одного log-файла, полученного на сервере другой сети:

Общее число пакетов - 603178

Число TCP-пакетов - 223469

Общее число пакетов	Число SYN-пакетов	Число SYN-ACK -пакетов	Разница относительно SYN-пкт.
2000	61	59	0.03
4000	138	136	0.01
6000	183	181	0.01
8000	271	262	0.03
10000	289	279	0.03
12000	308	296	0.04
14000	328	314	0.04
16000	432	338	0.22
18000	487	345	0.29
20000	551	390	0.29
40000	921	731	0.21
60000	1343	1118	0.17
80000	2290	2004	0.12
100000	2695	2388	0.11
140000	2754	2446	0.13
180000	2854	2533	0.11
220000	3668	3341	.0.09

Здесь мы видим резкое увеличение числа SYN-пакетов в районе 16000-го пакета (или 5000-го TCP-пакета как показано на графике поведения разницы между числом SYN- и SYN_ACK-пакетов в зависимости от числа TCP-пакетов).

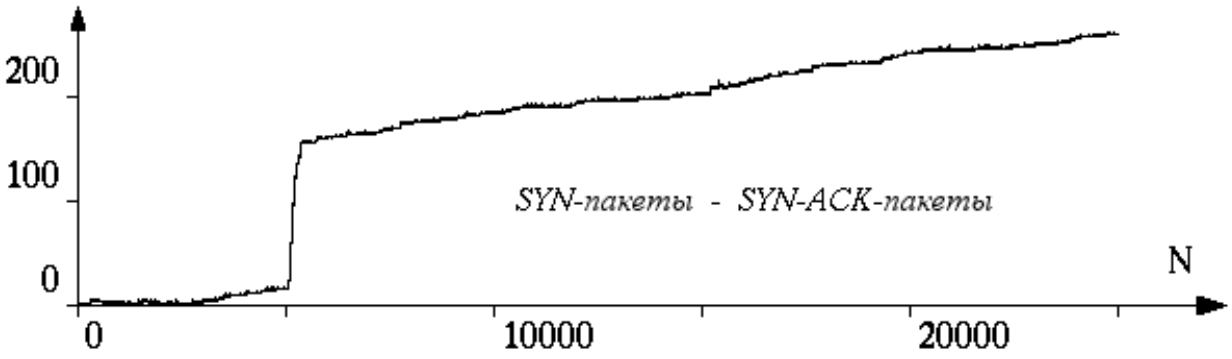


Рис 1

Такое поведение не является типичным для сети и нуждается в дополнительном изучении. Действительно, анализ пакетов в этой области показывает, что имеет место SYN-сканирование сети.

Предварительная обработка результатов мониторинга

Структура log-файлов программы TCPdump

Программа **TCPdump** обладает развитыми средствами фильтрации пакетов по комбинации большого числа различных признаков. Однако все эти средства относятся к отдельно взятому пакету, без учёта связи и корреляции его свойств с другими пакетами. Для решения задач связанных с учётом корреляционных свойств пакетов рассмотрим структуру log-файла пакета **TCPdump** и возможность использования результатов работы **TCPdump** другими программами.

Двоичный log файл начинается с заголовка файла, формат заголовка файла можно представить в виде структуры данных:

```
typedef struct tagPCAPFILEHDR {
    uint32_t   magic;
```

- 4-х байтовое поле содержащее сигнатуру 0xa1b2c3d4

```
    uint16_t   version_major;
    uint16_t   version_minor;
```

- 2-х байтовые поля содержащие номер версии

```
    uint32_t   thiszone;
```

- 4-х байтовое поле задающее отличие локального времени от UTC в секундах

```
    uint32_t   sigfigs;
```

- значение всегда равно 0

```
    uint32_t   snap_len;
```

- содержит размер той части захватываемого пакетов, которая будет записана в файл; обычно значение этого поля равно 68; если необходимо записать

весь пакет, то значение этого поля должно быть равно MTU или достаточно большой величине (больше возможного размера пакета)

```
uint32_t   linktype;
```

- содержит идентификатор протокола канального уровня для сетевого интерфейса с которого производился захват пакетов

```
} PCAPFILEHDR;
```

Некоторые из возможных значений для поля **linktype** в структуре заголовка файла:

- "null" 0
- 10Mb Ethernet 1
- 3Mb Ethernet 2
- IEEE 802.x 6
- ARCNET 7
- SLIP 8
- PPP 9
- FDDI 10
- "raw IP" 12

Все поля структуры заголовка файла записаны с использованием порядка байтов задаваемым архитектурой компьютера.

Непосредственно за заголовком файла следует последовательность записей соответствующих захваченным пакетам. Каждая запись состоит из заголовка пакета и необработанных данных самого пакета. Структура заголовка пакета имеет вид:

```
typedef struct tagPCAPPKTHDR {
   struct timeval ts;
```

- содержит время захвата пакета, формат этого поля зависит от типа используемой операционной системы

```
uint32_tcaplen;
```

- значение этого поля указывает сколько байт данных (без учёта заголовка пакета) было записано в файл

```
uint32_tlen;
```

- указывает размер захваченного пакета

```
} PCAPPKTHDR;
```

Вслед за заголовком пакета располагаются данные самого пакета, эти данные начинаются с заголовка канального уровня. Таким образом пакет записывается в файл в том виде в каком он поступил на сетевой интерфейс (возможно пакет обрезается по длине до размера указанного в поле **snap_len** заголовка файла). Структура заголовка канального уровня определяется типом физического сетевого устройства и протоколом канального уровня. В этом заголовке последние 2 байта задают тип инкапсулированного пакета: 0x0800 - ip, 0x0806 - arp, 0x0835 - rarp и так далее.

В соответствии с идеологией стека протоколов пакет протокола канального уровня содержит в себе пакет протокола другого уровня: например ip-пакет, пакет сетевого уровня. Структура этого пакета соответствует структуре данных:

```
typedef struct tagEVENT {
   IPHDR ip;
   union {
      ICMPHDR    icmp;
      TCPHDR     tcp;
      UDPHDR     udp;
```

```
    };
    uint8_t       rezerved[1024];
} EVENT, *PEVENT;
```

Структура заголовка ip-пакета:

```
typedef struct tagIPHDR {
    uint8_t    ihl:4,
               version:4;
    uint8_t    tos;
    uint16_t   tot_len;
    uint16_t   id;
    uint16_t   frag_off;
    uint8_t    ttl;
    uint8_t    protocol;
    uint16_t   check;
    uint32_t   saddr;
    uint32_t   daddr;
} IPHDR, *PIPHDR;
```

Пакет ip-протокола может содержать в себе пакеты следующих уровней стека протоколов: **IP, UDP, ICMP**.

Заголовки этих пакетов имеют следующую структуру:

```
typedef struct tagICMPHDR {
    uint8_t    type;
    uint8_t    code;
    uint16_tchecksum;
    union {
       struct {
          uint16_tid;
```

```
            uint16_tsequence;
         }  echo;
         uint32_tgateway;
         struct {
            uint16_treserved;
            uint16_tmtu;
         }  frag;
      } un;
} ICMPHDR, *PICMPHDR;

typedef struct tagTCPHDR {
   uint16_tsource;
   uint16_tdest;
   uint32_tseq;
   uint32_tack_seq;
   uint8_t    res;
   uint8_t    flags;
   uint16_twindow;
   uint16_tcheck;
   uint16_turg_ptr;
} TCPHDR, *PTCPHDR;
```

Более подробно о стеке протоколов и структуре сетевых пакетов различных уровней можно узнать из книги У. Стивенса [1]. Этот материал понадобится нам при изучении материала следующих лекций нашего курса.

Программная обработка результатов TCPdump

Зная принципы организации и структуру log-файла пакета **TCPdump** мы можем организовать взаимодействие **TCPdump** с какой-либо программой через файл: открыть этот файл в программе и провести анализ его содержимого программным путём. Если требуется обработка данных перехвата пакетов "на лету", в режиме реального времени, то следует использовать средства операционной системы. Можно создать именованный канал (ОС Linux):

```
mkfifo fifo.log
```

и перенаправлять вывод **TCPdump** в этот канал:

```
tcpdump ... > fifo.log
```

В этом случае в именованный канал направляется информация в текстовом формате: в том виде, в каком она выводится на экран монитора. В программе обработки перехваченных пакетов можно считывать информацию из именованного канала как из текстового файла и обрабатывать её по одной записи:

```
FILE* pF = fopen ("fifo.log", "r");
for (i=0; i<32; i++) {
   fgets (str, 1000, pF);
   . . .
   <обработка записи log-файла>
   . . .
}
fclose (pF);
```

При использование двоичного формата log-файла результат работы **TCPdump** следует записывать в именованный канал как в файл:

```
tcpdump ... -w fifo.log
```

а одновременно работающая программа будет считывать информацию из **fifo.log** и обрабатывать её в соответствии со структурой записей в log-файле (см. материал предыдущей лекции).

В качестве примера рассмотрим работу простой программы. Эта программа считывает поток данных из именованного канала, выделяет данные соответствующие одному ip-пакету и вызывает внешнюю функцию **treatdump( )** для обработки этих данных.

```
// Пример программы для обработки данных
// из log-файла TCPdump

uint32_t SIZE_LINK_HEADER;
```

- размер заголовка пакета канального уровня

```
void* thread_function (void* Param)
```

- эта функция м.б. Использована в качестве потоковой функции, поэтому у неё такой тип аргумента и возвращаемого значения

```
{
   uint8_t buffer[128];
```

- массив байтов для временного хранения информации

```
   PCAPFILEHDR *pfilehdr;
```

- указатель на структуру заголовка файла

```
   PCAPPKTHDR pkthdr;
```

- структура заголовка пакета

```
EVENT event;
```

- для одного перехваченного пакета из log-файла

```
FILE* pFile = fopen ("fifo.log", "r");
```

- открываем как файл именованный канал для чтения

```
fread (buffer, sizeof (PCAPFILEHDR), 1, pFile);
```

- читаем в буфер данные соответствующие заголовку файла

```
pfilehdr = (PCAPFILEHDR*)buffer;
```

- отображаем эти данные на структуру заголовка файла

Проверяем сигнатуру файла:

```
if (pfilehdr->magic != 0xa1b2c3d4) {
```

- если сигнатура не соответствует принятой для TCPdump, то закрываем файл и завершаем работу функции с соответствующим кодом возврата:.

```
    fclose (pFile);
    return 1;
}
```

Далее, в соответствии с типом протокола канального уровня

```
switch (pfilehdr->linktype) {
```

задаём размер заголовка этого протокола

```
    case 1:                 /* Ethernet (10Mb) */
        SIZE_LINK_HEADER = 14;
        break;
    case 113:               /* Linux cooked sockets
*/
        SIZE_LINK_HEADER = 16;
        break;
```

```
      default:
```

Для протоколов неуказанного типа закрываем файл и завершаем работу функции с соответствующим кодом возврата:

```
         fclose (pFile);
         return 2;
   }
```

```
   while (1) {
```

- запускаем цикл и на каждом шаге этого цикла

```
      fread (&pkthdr, sizeof (PCAPPKTHDR), 1, pFile);
```

- читаем из файла заголовок захваченного пакета

```
      if (feof (pFile))
         break;
```

- в случае достижения конца файла выходим из цикла

```
      fread (&buffer, SIZE_LINK_HEADER, 1, pFile);
```

- читаем заголовок пакета канального уровня

```
      fread (&event, pkthdr.caplen - SIZE_LINK_HEADER,
1, pFile);
```

- читаем сам пакет и помещаем его в поля структура **event**

Проверяем тип пакета сетевого уровня

```
      if (ntohs (*(uint16_t*)(buffer+SIZE_LINK_HEADER-
2)) == 0x0800)     // ip
```

- если это пакет протокола **ip**, топередаём его функции **treatdump** для дальнейшей обработки

```
         treatdump (&event);
   }
```

После выхода из цикла закрываем файл и завершаем работу функции

```
    fclose (pFile);
    return NULL;
}
```

Анализ главных компонент

Principal Component Analysis (PCA) - анализ главных компонент; метод анализа данных основанный на преобразовании взаимозависимых (коррелированных) компонент вектора характеристик событий в набор независимых (некоррелированных) переменных, называемых главными компонентами. Основная цель метода главных компонент заключается в выделении существенных (ответственных за основные изменения) в рассматриваемой системы характеристик и снижении размерности соответствующего пространства характеристик.

Для простоты начнём с рассмотрения случая двумерного пространства характеристик $X : X_i = (x_i, y_i), i = 1,..., N$. Наблюдаемые величины x и y центрированы и соответствующим образом нормированы:

$$\bar{x} = \frac{1}{N} \cdot \sum_{i=1}^{N} x_i = 0, \quad \sigma_x^2 = \frac{1}{N} \cdot \sum_{i=1}^{N} x_i \cdot x_i = 1$$

$$\bar{y} = \frac{1}{N} \cdot \sum_{i=1}^{N} y_i = 0, \quad \sigma_y^2 = \frac{1}{N} \cdot \sum_{i=1}^{N} y_i \cdot y_i = 1$$

Будем считать, что величины x и y не являются независимыми и соответствующая ковариация составляет:

$$\sigma_{xy} = \frac{1}{N} \cdot \sum_{i=1}^{N} x_i \cdot y_i = \zeta$$

В этом случае ковариационная матрица имеет вид:

$$\Sigma = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{pmatrix} = \begin{pmatrix} 1 & \zeta \\ \zeta & 1 \end{pmatrix}$$

В дальнейшем нам понадобится обратная матрица Σ^{-1} :

$$\Sigma \cdot \Sigma^{-1} = \Sigma^{-1} \cdot \Sigma = 1, \quad \Sigma^{-1} = \frac{1}{1-\zeta^2} \cdot \begin{pmatrix} 1 & -\zeta \\ -\zeta & 1 \end{pmatrix}$$

найдём собственные значения и собственные вектора ковариационной матрицы:

$$\Sigma \cdot e = \lambda \cdot e$$

здесь λ - собственное значение, $e = (e_x, e_y)$ - собственный вектор. Искомые собственные значения и собственные вектора имеют вид:

$$\lambda_1 = 1 + \zeta, \quad e^{(1)} = \frac{1}{\sqrt{2}} \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$\lambda_2 = 1 - \zeta, \quad e^{(2)} = \frac{1}{\sqrt{2}} \cdot \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

Обратную матрицу Σ^{-1} можно представить в виде:

$$\Sigma^{-1} = \frac{1}{\lambda_1} \cdot e^{(1)} \times e^{(1)} + \frac{1}{\lambda_2} \cdot e^{(2)} \times e^{(2)}$$

В пространстве характеристик X перейдём к новому базису, основанному на собственных векторах $e^{(1)}$ и $e^{(2)}$ ковариационной матрицы Σ. В этом случае мы переходим от наблюдаемых характеристик x и y к новым характеристикам u_1 и u_2 :

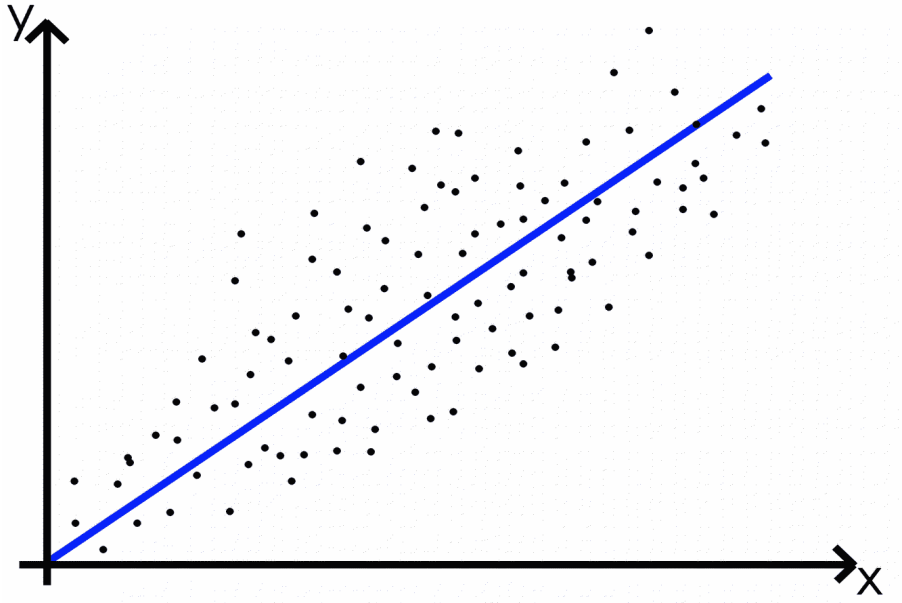
$$u_1 = \left(e^{(1)} \cdot X\right) = \left(e_x^{(1)}\ e_y^{(1)}\right) \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{\sqrt{2}} \cdot (x + y)$$

$$u_2 = \left(e^{(2)} \cdot X\right) = \left(e_x^{(2)}\ e_y^{(2)}\right) \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{\sqrt{2}} \cdot (x - y)$$

Представим (смоделируем зависимость) величину y в виде:

$$y = \zeta x + \sqrt{1-\zeta^2}\, z$$

величины x и z - независимы и кроме того: $\bar{z} = 0$, $\sigma_z^2 = 1$.



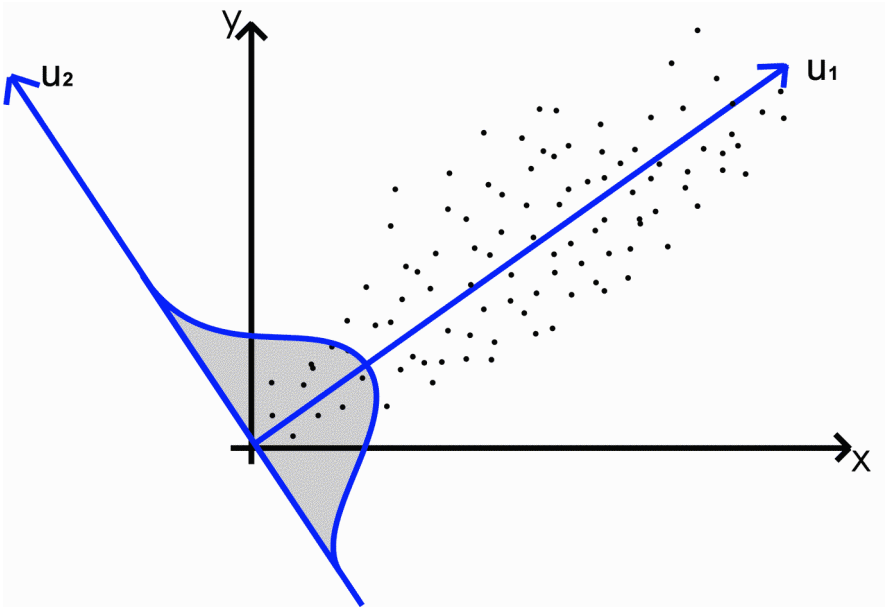
В этом случае выражения для величин u_1 и u_2 примут вид:

$$u_1 = \sqrt{\frac{\lambda_1}{2}} \cdot \left(\sqrt{1+\zeta}\, x + \sqrt{1-\zeta}\, z\right)$$

$$u_2 = \sqrt{\frac{\lambda_2}{2}} \cdot \left(\sqrt{1-\zeta}\, x - \sqrt{1+\zeta}\, z\right)$$

В том случае, если компоненты x и y независимы $\zeta = 0$, то переход к переменным u_1 и u_2 сводится к повороту координатных осей:

$$u_1 = \sqrt{\frac{1}{2}} \cdot (x+y),\ u_2 = \sqrt{\frac{1}{2}} \cdot (x-y)$$



Если компоненты x и y существенно скоррелированы $|\zeta| \to 1$, то одно из собственных значений стремится к нулю и в силу соотношений

$u_1 \approx \sqrt{\lambda_1}x$, $u_2 \approx \sqrt{\lambda_2}z$

соответствующая компонента тоже стремится к нулю $u_k \to 0$ и задача фактически становится одномерной.

Новые компоненты u_1 и u_2 вектора характеристик X обладают следующими свойствами:

1. u_1 и u_2 центрированы: $\overline{u}_1 = \overline{u}_2 = 0$
2. u_1 и u_2 независимы: $\sigma_{12} = \frac{1}{N} \cdot \sum_{i=1}^{N} u_{1i} \cdot u_{2i} = 0$
3. Дисперсии $\sigma_{u1}^2 = \lambda_1$ и $\sigma_{u2}^2 = \lambda_2$ равны соответствующим собственным значениям корреляционной матрицы.
4. Направление координатной оси, соответствующей компоненте u_1 с максимальным значением собственного значения λ_1 ковариационной матрицы Σ, обозначает направление наибольшего разброса значений

в пространстве характеристик.

С учётом приведённых рассуждений для метода главных компонент обратимся к метрике в пространстве характеристик X. Используемая метрика $d(X_1, X_2)$ определяет расстояние между точками X_1 и X_2 в пространстве характеристик. Для простоты мы будем рассматривать расстояние от точки $X = (x, y)$ до начала координат $d(X, 0)$. Наиболее часто в качестве метрики используется евклидово расстояние:

$$d_E^2(X,0) = x^2 + y^2 = u_1^2 + u_2^2$$

Эта формула достаточно проста и имеет одинаковый вид как для исходных компонент вектора характеристик так и для главных компонент. Однако евклидова метрика имеет существенный недостаток: так как компоненты u_1 и u_2 имеют разную степень разброса σ_{u1}^2 и σ_{u2}^2 от своих средних значений $\bar{u}_1$ и $\bar{u}_2$. Это обстоятельство соответствует тому, что разные компоненты имеют разный удельный вклад при вычислении расстояния и возможна потеря информации при использовании евклидова расстояния в задачах анализа данных. В подобных случаях вместо евклидова расстояния уместно использовать расстояние Махаланобиса:

$$d_M(X,0) = X^T \cdot \Sigma^{-1} \cdot X = \frac{1}{1-\zeta^2} \cdot \begin{pmatrix} x & y \end{pmatrix} \begin{pmatrix} 1 & -\zeta \\ -\zeta & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \frac{x^2 + y^2 - 2\zeta xy}{1-\zeta^2}$$

При использовании независимых компонент $x, z : y = \zeta x + \sqrt{1-\zeta^2}\, z$ расстояние Махаланобиса имеет вид:

$$d_M(X,0) = x^2 + z^2$$

и в случае главных компонент получаем:

$$d_M(X,0)=\frac{u_1^2}{\lambda_1}+\frac{u_2^2}{\lambda_2}$$

Таким образом, использование метрики Махаланобиса позволяет выравнять вклад отдельных компонент при вычислении расстояния между точками в пространстве характеристик.

Обобщим проведённое рассмотрение на случай пространства произвольной размерности m. Пусть $X: X_i=\left(x_1^i,...,x_m^i\right), i=1,...,N$ вектор в m-мерном пространстве характеристик, описывающий некоторое событие D в системе. Компоненты x_k^i представляют характеристики отдельного события, нижний индекс $k: 1\le k\le m$ обозначает номер компоненты вектора характеристик, верхний индекс $i: 1\le i\le N$-номер события. Как и прежде, компоненты x_k центрированы:

$$\bar{x}_k=\frac{1}{N}\cdot\sum_{i=1}^{N}x_k^i=0$$

и соответствующим образом нормированы:

$$\sigma_k^2=\frac{1}{N}\cdot\sum_{i=1}^{N}x_k^i x_k^i=1$$

Корреляционная $m\times m$-матрица Σ имеет вид:

$$\Sigma=\begin{pmatrix}1 & \dots & \sigma_{1k} & \dots & \sigma_{1m}\\ \dots & \dots & \dots & \dots & \dots\\ \sigma_{kl} & \dots & 1 & \dots & \sigma_{1k}\\ \dots & \dots & \dots & \dots & \dots\\ \sigma_{ml} & \dots & \sigma_{mk} & \dots & 1\end{pmatrix}$$

недиагональные элементы матрицы Σ

$$\sigma_{kl}=\frac{1}{N}\cdot\sum_{i=1}^{N}x_k^i x_l^i,\ k\ne l$$

определяет меру зависимости компонент вектора характеристик X. Теперь

находим собственные значения $\lambda_k : 1 \le k \le m$ и собственные вектора $e^{(k)} : 1 \le k \le m$ вариационной матрицы: $\Sigma \cdot e^{(k)} = \lambda_k \cdot e^{(k)}$ (заметим, что $\lambda_1 + \lambda_2 + ... + \lambda_m = m$). Будем считать, что собственные значения упорядочены в порядке убывания: $\lambda_1 \ge \lambda_2 \ge ... \ge \lambda_m$. Если некоторые из собственных значений существенно меньше других: $\lambda_1, ..., \lambda_p << \lambda_{p+1}, ..., \lambda_k$, то это означает, что среди m компонент $x_k : 1 \le k \le m$ вектора характеристик X только p реально независимых. Поэтому при переходе к новым, независимым компонентам u_k:

$$u_k = \sum_{l=1}^{m} e_l^{(k)} x_k : \quad 1 \le k \le p$$

имеет смысл использовать только первые $p < m$ из них. В этом случае размерность данных снижается, что может существенно упростить анализ состояния системы.

(Рассуждения о повороте в m-мерном пространстве и проекции на двумерную плоскость).

Выбор метода обнаружения нарушений и построение модели состояния системы

Основные понятия безопасности в сети

С ростом использования компьютерных сетей становится необходимым использование систем безопасности. Вторжение в сеть злоумышленников или неавторизованных пользователей может привести к нарушению работоспособности сети. Поэтому развитие адекватных и надёжных систем обнаружения вторжений в сеть (IDS – Intrusion Detection System) чрезвычайно важная задача.

Вторжение (intrusion) можно определить следующим образом: любые действия направленные на нарушение целостности, конфиденциальности или доступности ресурсов. В качестве первой линии защиты информации компьютерных систем используется техника предотвращения вторжений, такая как аутентификация пользователей (использование паролей, биометрия ...), исключение программных ошибок, защита информации (шифрование, стеганография ...). Использование только методов предотвращения вторжений недостаточно, так как сложность систем постоянно возрастает и возрастает их уязвимость вследствие программистских ошибок или использования “социальной инженерии” в качестве техники проникновения. По этой причине компьютерные системы нуждаются ещё в одной линии защиты – системах обнаружения вторжений.

Обнаружение вторжений – определение атаки на систему путём проверки различных данных аудита системы (системные логи, заголовки пакетов ...) и отделения нормальных пользователей от нарушителей. Цель обнаружения вторжений – построение системы, которая могла бы автоматиче-

ски следить за сетевой активностью и определять такие попытки вторжения (атаки). В том случае, если обнаружено вторжение в систему, администратор системы информируется об этом и должен предпринять соответствующие действия.

Центральными элементами системы определения вторжений являются:

•*ресурсы,* подлежащие защите в атакуемых системах (учётные записи, файловые системы, ядро системы и т.д.).

•*модели,* которые характеризуют “нормальное” или “легитимное” поведение этих ресурсов.

•*техники (алгоритмы)* сравнивающие текущую активность системы с выбранной моделью и идентифицирующие поведение системы как “аномальное”.

Традиционно для решения задачи определения вторжений применяются методы, основанные на использовании сигнатур. Когда атака на сеть обнаружена и проанализирована, соответствующие образцы трафика передаются экспертам в виде сигнатур и впоследствии используются для построения базы данных сигнатур и выявления опасного трафика. Сигнатурные методы обнаружения вторжений могут выявить только заранее известную атаку с соответствующей сигнатурой. Ограничения сигнатурных методов увеличивают интерес к методам обнаружения вторжений основанным на аналитической обработке данных (data mining).

Техника аналитической обработки данных предназначена для обнаружения в больших массивах данных ранее неизвестной, нетривиальной и

практически полезной информации и широко используется при решении следующих задач:

- **Классификация** – отнесение объекта к одному из заранее известных классов.
- **Кластеризация** – разделение множества объектов на группы (кластеры) по степени «похожести» друг на друга.
- **Ассоциация** – поиск повторяющихся паттернов. Например, поиск устойчивых связей в корзине покупателя (market basket analysis) – вместе с пивом покупают орешки.
- **Анализ аномалий** – выявление наиболее нехарактерных паттернов. Например, выявление нетипичной сетевой активности позволяет обнаружить вредоносные программы.
- В области аналитической обработки данных используется различные алгоритмы: методы распознавания образов, методы машинного обучения, статистические методы и т.п.

Основанная на аналитической обработке данных техника обнаружения вторжений в общем случае распадается на две категории: выявление нарушений (misuse detection) и выявление аномалий (anomaly detection). При использовании методов выявления нарушений каждый элемент тренировочного набора данных помечается как “нормальный” или “опасный” и обучающий алгоритм использует помеченные данные для обучения систем обнаружения вторжений. Подходы, основанные на выявлении аномалий, строят модели нормальных данных и фиксируют отклонения от нормальных моделей в наблюдаемых данных.

Модель обнаружения вторжений

Модель экспертной системы реального времени обнаружения вторжений была предложена Дороти Деннинг в 1987 году [2] и до настоящего момента времени является отправным пунктом при построении практически любой системы обнаружения вторжений. Модель независима от любой специфической системы, прикладной среды окружения, уязвимости системы или типа вторжения. Эта модель описывает структуру экспертной системы обнаружения вторжения общего назначения.

Модель основана на гипотезе о том, что **использование уязвимостей системы проявляется в аномальном (нетипичном) использовании системы**. Поэтому, нарушения безопасности могут быть обнаружены при появлении признаков неправильного использование системы. Следующие примеры иллюстрируют это:

- *Отказ в обслуживании (Denial-of-Service)* – поведение, проявляющееся в монополизации ресурса (например, сети), может иметь аномально высокую активность в отношении этого ресурса.
- *Попытка взлома (Attempted break-in)* –попытка взлома системы может сопровождаться большим количеством отказов в принятии пароля для отдельной учетной записи.
- *Троянский конь (Trojan horse)* - поведение программы с установленным троянским конём может отличиться от обычной программы по загруженности CPU или интенсивности операций ввода/вывода.
- *Вирус (Virus)* - вирус, установленный в системе, мог бы привести к увеличению частоты перезаписи исполнимых файлов используемых как вирусные распространители.

Очевидно, что отмеченные выше аномалии также могут быть связаны с обычной деятельностью, не связанной с нарушениями безопасности: пользователь может перейти к решению новой задачи, использовать новые методы или просто ошибку в программе, обновлять программное обеспечение или изменить стиль работы.

Предлагаемая модель обнаружения вторжений не зависит от особенностей системы, прикладного окружения, уязвимостей системы или типа нарушений. Эта модель описывает экспертную систему обнаружения вторжений общего назначения (IDES - intrusion-detection expert system).

Рассматриваемая модель состоит из следующих компонент:

1. ***Субъекты*** – источники активности в системе: процессы, сама система, пользователи или процессы, инициализированные пользователями.
2. ***Объекты*** – точка приложения активности субъектов: программы, файлы, устройства, отчёты, ...
3. ***Записи аудита системы (логи)*** – записи генерируемые системой при воздействии субъекта на объект: вход в систему, доступ к файлу, выполнение команды, ...
4. ***Профиль активности*** описывает поведение данного субъекта по отношению к объекту.
5. ***Записи об аномалиях*** – создаются при обнаружении аномального поведения системы.
6. ***Правила активности*** – специфицируют действие, исполняемое при выполнении некоторых условий, этими условиями могут быть: появление записи аудита, генерация записи об аномальном событии или завершение временного интервала.

Рассмотрим более подробно некоторые компоненты модели обнаружения вторжений.

Записи аудита характеризуют воздействие субъекта на объект. Информация, содержащаяся в записях аудита должна соответствовать структуре:

<Subject, Action, Object, Exception-Condition, Resource-Usage, Time-stamp>

Поля этой структуры имеют следующий смысл:

- Subject – субъект;
- Action – воздействие субъекта на объект;
- Object – объект;
- Exception-Condition – передаваемый системе код завершения. Этот код завершения должен соответствовать предусмотренным в системе случаям обработки исключений.
- Resource-Usage – список, каждый элемент списка характеризует использование некоторого ресурса: время использования процессора, количество чтений из файла, количество записей в файл и т.п. ;
- Time-stamp – уникальное значение времени события;

Так, например команда копирования пользователем **User** файла **file1** в **file2**

```
COPY file1 file2
```

приведёт к появлению следующих записей аудита:

```
(User, execute, COPY.EXE, O, CPU=00002,
11058521678)
```

- системный вызов функции копирования;

(User, read, file1, O, RECORDS=0, 011058521679)

- чтение файла **file1;**

(User, write, file2, write_viol, RECORDS=O, 11058521680)

- запись в файл **file2** (операция привела к нарушению защиты).

Профиль активности описывает поведение данного субъекта по отношению к объекту. Наблюдаемое поведение описывается в терминах метрики и модели активности. Метрика это некоторая величина X, представляющая количественную меру, аккумулированную за некоторый интервал времени (секунду, минуту, час, время работы программы, время сеанса пользователя ...). Конкретные значения X_i величины X, полученные при аудите системы, используются совместно с моделью активности для нахождения аномальных событий.

В общем виде структура профиля может быть представлена как

<Variable-Name, Action-Pattern, Exception-Pattern, Resource-Usage-Pattern, Period, Variable-Type, Threshold, Subject-Pattern, Object-Pattern, Value>

с десятью полями:

- Variable-Name – имя переменной, соответствующей наблюдаемой величине;
- Action-Pattern – операция (‘login’, ‘read’, ‘execute’, …);
- Exception-Pattern – код завершения;
- Resource-Usage-Pattern – используемый ресурс;
- Period – интервал измерения;

- Variable-Type – абстрактный тип данных, определяет тип используемой метрики и модели активности;
- Threshold – параметр модели, этот параметр задаёт критерий аномалии и его смысл зависит от используемой модели (пороговое значение, количество стандартных отклонений и т.д.);
- Subject-Pattern – субъект;
- Object-Pattern – объект;
- Value – текущие значения наблюдаемой величины и параметров используемых в статистической модели (например количество предыдущих событий определённого типа);

Профиль активности должен однозначно идентифицироваться именем переменной, субъектом и объектом. Пример одной записи профиля:

```
Variable-Name:           SessionOutput
Action-Pattern:          'logout'
Exception-Pattern:       0
Resource-Usage-Pattern:  amount
Period:
Variable-Type:           ResourceByActivity
Threshold:               4
Subject-Pattern:         User
Object-Pattern:          *
Value:
```

Структура профиля может быть создана администратором системы безопасности или автоматически создаваться при первом использовании объекта субъектом.

Записи об аномалиях создаются при обнаружении аномального по-

ведения защищаемой системы. В соответствии с правилами активности система обнаружения вторжений создаёт запись профиля активности и производит проверку относительно аномального поведения. Если зафиксировано аномальное событие, то генерируется соответствующая запись. Запись об аномальном событии может состоять из трёх полей:

<Event, Time-stamp, Profile>

- Event – событие, предшествовавшее выявлению аномалии;
- Time-stamp – время выявления аномалии;
- Profile – профиль активности, соответствующий выявленной аномалии.

При необходимости система обнаружения вторжений может добавить дополнительные поля к профилю активности.

Правила активности специфицируют действие, выполняемое при удовлетворении некоторых условий. Этими условиями могут быть запись аудита, генерации записи об аномальном событии или завершение временного интервала. Правила активности состоят из двух частей: условия и тела (действия). В модели используется четыре типа правил:

1. Правило записи аудита.
2. Правило периодической проверки активности
3. Правило аномальных записей
4. Правило периодической проверки аномалий

Метрика и модель активности. Центральным пунктом системы обнаружения вторжений являются используемые метрика модель активности. Метрика позволяет получить количественную меру состояния системы в

соответствии с теми её характеристиками, которые имеют отношение к задаче выявления вторжений. Модель активности системы предназначена для получения значений метрики соответствующих нормальному (легитимному, типичному) состоянию системы. В модели активности могут быть использованы следующие типы метрики:

1. *Счётчик событий* – величина X характеризует число записей аудита удовлетворяющих некоторым условиям в течение заданного интервала времени. Например, количество пакетов поступивших в течение одной секунды.

2. *Временной интервал* – X это промежуток времени между двумя событиями. Например, время между двумя входами пользователя в систему.

3. *Мера (величина) ресурса* – X это "количество ресурса" измеренное в соответствующих единицах. Например: объём используемой оперативной памяти, количество одновременно открытых файлов, объём отправленной информации.

Когда происходит очередное событие (установление сетевого соединения, получение пакета, системный вызов и т.д.) в соответствии с введённой мерой вычисляются числовые характеристики события и генерируется соответствующая запись аудита. В соответствующий момент времени, определяемый правилами активности, сравниваются реальные характеристики системы с характеристиками предлагаемыми моделью активности. Модель активности позволяет определить, когда очередное значение X_{n+1} величины X является аномальным по отношению к предыдущим наблюдениям $X_1, ..., X_n$. В системе обнаружения вторжений могут быть использованы модели:

3. *Операционная модель.* Эта модель базируется на допущении о том, что

аномалия может быть выявлена при сравнении нового наблюдения величины X с фиксированным пределом. Эта модель применима в тех случаях, когда эксперименты показывают, что некоторые значения метрики обычно соответствуют вторжениям. Например, пароль неверно был неверно введён более трёх раз подряд.

4. *Модель среднего значения и стандартного отклонения.* Новое наблюдение X_{n+1} будет аномальным, если выходит за пределы доверительного интервала $\overline{X} \pm d \cdot \sigma_X$, где $\overline{X}$ - среднее значение, σ_X - стандартное отклонение, d - некоторый параметр: $\overline{X} = \frac{1}{n} \cdot \sum_{i=1}^{n} X_i$, $\sigma_X = \sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} \left(X_i - \overline{X}\right)^2}$. В соответствии с неравенством Чебышева: $\mathrm{P}\left\{\left|X_{n+1} - \overline{X}\right| \geq d * \sigma_X\right\} \leq \frac{1}{d^2}$. Это означает, что для любого вида распределения величины X и при значении параметра $d = 4$ вероятность выйти за пределы доверительного интервала не превышает 0.0625.

5. *Многовариантная модель.* Эта модель базируется на корреляциях между несколькими метриками. Например, загрузка процессора и частота обращения к устройствам ввода/вывода, количество входов в систему и длительность сеанса и т.д.

6. *Модель Марковского процесса.* Эта модель применима только к счётчикам событий, которые рассматриваются как характеристика состояния системы, в модели вводится матрица переходов, описывающая частоту переходов между состояниями. Событие определяется как аномальное, если вероятность его появления (в соответствии с предыдущим состоянием и матрицей переходов) мала. Эта модель может быть полезна при анализе событий в том случае, когда важна их последовательность.

7. *Модель временных рядов (серий).* Эта модель использует временной ин-

тервал совместно с мерой ресурса или счётчиком событий, учитывая величину и время появления событий $X_1, \ldots, X_n$. Событие считается аномальным, если вероятность его появления в данный момент времени мала.

Статистические методы выявления аномалий

Одним из способов построения модели нормальной активности системы является использование статистических методов. Статистические методы обнаружения попыток нарушения безопасности в сети основаны на том обстоятельстве, что в случае нарушения безопасности могут изменяться некоторые статистические характеристики потока пакетов. Так, например, в случае Flood-атаки резко возрастает трафик, при сканировании сети увеличивается доля пакетов определённого типа (SYN, ACK, FIN, …) и т.п. В этом случае методы обнаружения нарушений основываются на сравнении текущих, локальных характеристик потока пакетов с усреднёнными за продолжительный промежуток времени, глобальными характеристиками. В качестве статистических характеристик обычно используется энтропия, критерий согласия χ^2 и т.п.. Если локальные характеристики сильно отличается от соответствующих глобальных характеристик, то это свидетельствует об аномальном поведении потока пакетов и вполне вероятна попытка сканирования сети или сетевой атаки. В этой лекции мы рассмотрим некоторые методы вычисления статистических характеристик системы и применения их для выявления аномалий в сети.

Будем считать, что числовая величина X_i, $x_{\min} < X_i \le x_{\max}$ представляет

некоторое событие из потока событий произошедшее в момент времени t_i, $1 \le i \le N$. Весь набор событий характеризуется средним значением $\bar{x}$ и дисперсией σ_x^2 величины X. В качестве статистической характеристики потока событий будем использовать среднее арифметическое функции $f(X)$ от величины X:

$$w(N) = \frac{1}{N} \cdot \sum_{i=1}^{N} f(X_i)$$

Общее количество событий N определяется интервалом времени, в течении которого ведётся наблюдение за потоком. При увеличении числа событий N среднее арифметическое $w(N)$ стремится к $\mathrm{M}[f(X)]$ - математическому ожиданию величины $f(X)$ и может быть использовано в качестве глобальной, долговременной характеристики потока. Для определения локальных характеристик среднее значение будем вычислять не для всего потока N событий, а только для n последних событий. С этой целью введём весовую функцию $F(z)$ и значения локальных характеристик $W(N)$ будем вычислять по формуле:

$$W(N) = \sum_{i=1}^{N} F(t_N - t_i) \cdot f(X_i) \quad (1)$$

Для нахождения локальных статистических характеристик потока событий мы будем использовать весовую функцию следующего вида:

$$F(z) = \frac{1}{k} \cdot \exp(-z / \tau) \quad (2)$$

Функция $F(z)$ локализована вблизи нуля и довольно быстро (экспоненциально) убывает с ростом аргумента z. Параметр τ, присутствующий в определении весовой функции, задаёт временной интервал, на котором эффективно вычисляются локальные характеристики $W(N)$ (1). Коэффициент k в формуле (2) введён для обеспечения правильной нормировки функции

$F(z)$:

$$\sum_{i=1}^{N} F(t_N - t_i) = 1$$

Предлагаемый в данной работе выбор весовой функции обусловлен тем обстоятельством, что формула (2) позволяет получить простые рекуррентные соотношения для вычисления $W(N)$. Введём обозначения

$$W(N) = A(N) / K(N) \quad (3)$$

где

$$A(N) = \sum_{i=1}^{N} f(X_i) \cdot \exp(-(t_N - t_i)/\tau)$$

$$K(N) = \sum_{i=1}^{N} \exp(-(t_N - t_i)/\tau)$$

и выделяя вклад последнего события, получаем рекуррентные соотношения для вычисления величин $A(N)$ и $K(N)$:

$$A(1) = f(X_1)$$

$$A(N) = f(X_N) + e^{-\Delta/\tau} \cdot A(N-1) \qquad (4)$$

$$K(1) = 1$$

$$K(N) = 1 + e^{-\Delta/\tau} \cdot K(N-1) \quad (5)$$

где $\Delta_N = t_N - t_{N-1}$ - временной интервал между последним и предпоследним событиями в потоке. Учитывая тот факт, что величины $X_1, \ldots, X_N$ характеризуют события, происходящие в последовательные моменты времени $t_1, \ldots, t_N$, формулы (3)-(5) позволяют реализовать вычисления локальных характеристик $W(N)$, $W(N+1)$, ... в режиме реального времени, по мере поступления новых пакетов и получения числовых характеристик X_N, X_{N+1} ... потока сети.

В предположении $N >> n$ и $\Delta / \tau << 1$ результат вычисления τ хорошо аппроксимируется выражением

$$\tau = \frac{\Delta \cdot n}{2} \qquad (6)$$

Таким образом, использование весовой функции (2) при вычислении усреднённых значений W эквивалентно нахождению среднего значения функции $f(X)$ от n последних элементов в последовательности $X_1,...,X_N$. Величина n и параметр τ связаны соотношением (6).

Здесь должен быть график разности SYN и SYN-ACK пакетов и анализ скачка этой разности вблизи ~5000 пакета (class_tcpdump.log)

```
//
// Обработка результатов от tcpdump
//
//
#include "main.h"
#include <math.h>
#define dN (0.5*1000)

double sA = 0.0;
double saA = 0.0;
double K = 0.0;
void treatdump (PEVENT pEvent)
{
   char str[1000];
   static uint32_t N=0, synN=0, synackN=0;
   if (pEvent->ip.protocol == 6) {
```

```
            // tcp
      N++;
      K = 1.0 + K * exp (-1.0/dN);
      sA = sA * exp (-1.0/dN);
      saA = saA * exp (-1.0/dN);
// syn - флаг
      if (pEvent->tcp.flags == 0x02) {
         synN++;
         sA = 1.0 + sA;
         usleep (10000);
      }
// syn+ack - флаги
      if (pEvent->tcp.flags == 0x12) {
         synackN++;
         saA = 1.0 + saA;
         usleep (10000);
      }
      sprintf (str, "N=%lu syn=%lf synack=%lf
%lf", N, sA/K*1000, saA/K*1000, (sA-saA)/K*1000);
      myTextOut (hPm, hGC, 20, 20, str);
      myRedrawRect (hWnd, 20, 20-9, 6*strlen (str),
11);
   }
}
```

Результат применения программы для подсчёта числа TCP-протокола с установленными SYN- и SYN-ACK-флагами.

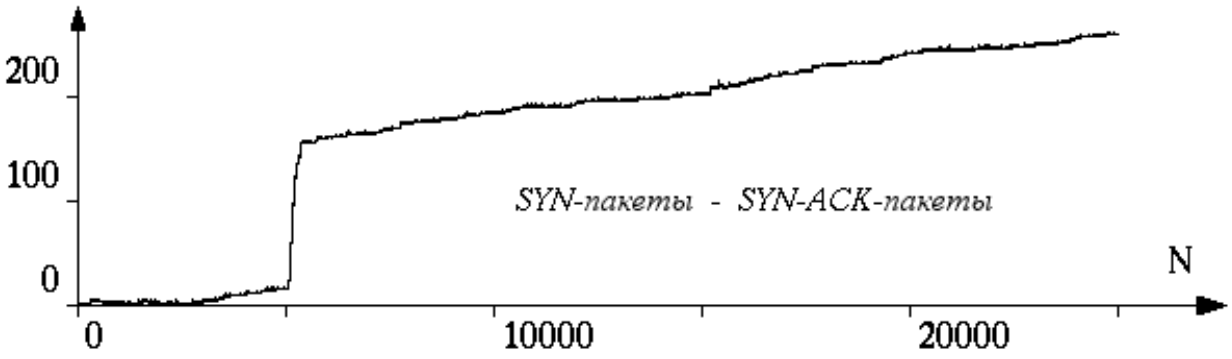


Рис 1

На рисунке 1 приведён график зависимости разности числа пакетов с установленными SYN- и SYN-ACK-флагами в зависимости от числа перехваченных TCP-пакетов. Данные представленные на графике относятся к глобальным характеристикам потока пакетов в сети, они подсчитываются в течении всего времени мониторинга сети. За исключением скачка в районе 5000-го пакета график разности числа SYN- и SYN-ACK-пакетов медленно растёт, что соответствует некоторому количеству отвергнутых TCP-соединений и, по-видимому, является типичным, так как наблюдается при анализе данных других сетей (см. предыдущую лекцию об обнаружении аномалий при анализе заголовков пакетов).

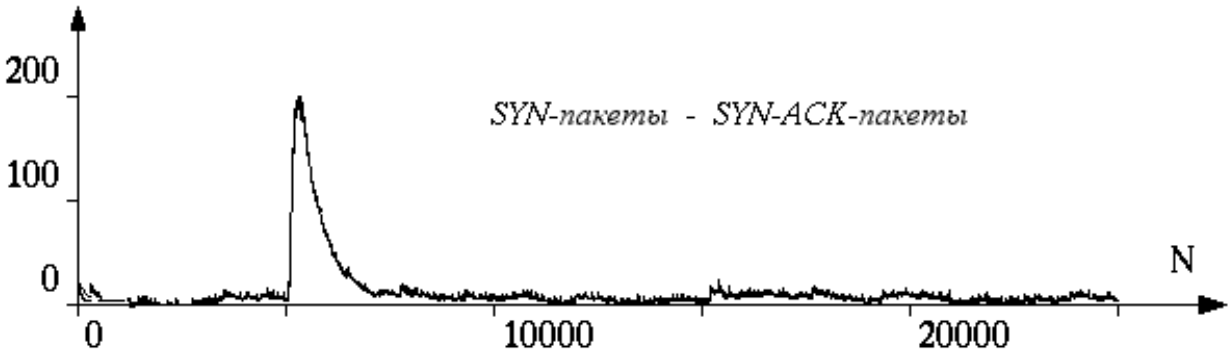


Рис 2

На рисунке 2 приведены результаты обработки тех же данных, только раз-

ница вычисляется не для всего потока, а для последних 1000 пакетов — локальная характеристика.

Кластерный анализ

Задачей кластерного анализа является разбиение данных (кластеризации) на группы в соответствии с некоторыми признаками или свойствами данных. Кластеризация данных является хорошо известной и изученной задачей. Среди методов кластеризации можно выделить следующие методы: разделяющие методы; методы, основанные на плотности; решёточные методы и другие.

1.*Разделяющие методы.* Разделяющие методы разбивают набор данных на группы, каждая группа данных представляет собой кластер. Наиболее интересен алгоритм кластеризации с фиксированным радиусом. Достоинство этого метода заключается в том, что его затраты линейно зависят от числа объектов в наборе данных и числа атрибутов у объекта.

2.*Методы основанные на плотности.* Эти методы основаны на простом допущении: кластеры являются областями высокой плотности в пространстве характеристик данных и разделены областями низкой плотности. Основная идея заключается в том, что область принадлежит кластеру, если плотность в этой области превышает некоторую пороговую величину. Эти методы хороши для фильтрации выбросов и для создания кластеров произвольной формы. Примером таких методов может служить алгоритм ближайших соседей.

3.*Решёточные методы.* Решёточные методы разделяют пространство объектов на конечное число ячеек, эти ячейки формируют решёточную структуру. Все кластерные операции производятся на решёточной структуре. Основное достоинство этого приближения заключается в высокой производительности, время вычислений существенно зависит от числа ячеек по каждой размерности квантованного пространства.

В этой лекции мы рассмотрим использование кластеров для описания модели активности в системе обнаружения вторжений. Методы обнаружения вторжений, основанные на выявлении аномальных событий, построены по следующему принципу: вначале создаётся модель нормального поведения, характеризующая поведение системы в отсутствие нарушений, а затем выявляются отклонения наблюдаемых данных от нормального поведения. Одним из методов построения модели нормального поведения системы является кластеризация данных. В основе этого метода лежит допущение о том, что в пространстве числовых характеристик событий множество данных кластеризуется - собирается в отдельные группы. При подходящем выборе набора характеристик нормальные и аномальные события различимы - они не могут попасть в один кластер. Кроме этого мы предполагаем что число нормальных событий заметно превышает число аномальных событий, т.е. нормальные события формируют большие кластеры. Следует понимать ограничения такого подхода: Например, система не обнаружит вторжений, если злоумышленник будет маскироваться под авторизованного пользователя, так как в этом случае может не быть признаков отличия нормальных и аномальных событий. Другой пример – трудности с определением syn-flood DoS атаки, в этом случае число неправильных пакетов велико и даже может сравниться с числом нормальных.

Одно из основных допущений кластерного метода состоит в том, что данные с близкими значениями метрики будут стремиться собраться вместе в отличие от данных с разными значениями метрики. Следовательно, нахождение или построение соответствующей метрики является критичным для эффективности метода. Первым шагом в выборе метрики является отображение входных данных из наблюдаемого потока в пространство признаков. Пространство признаков это векторное пространство обычно большой размерности. Размерности этого пространства представляют (соответствующим образом нормированные) числовые характеристики элемента данных.

Обозначим через D_1 , D_2 , ..., D_i , ... элементы входного набора данных и через D пространство всех возможных элементов данных или пространство событий. Конкретный вид пространства D зависит от вида анализируемых данных (приходящие пакеты, сетевые соединений, последовательность системных вызовов и т.п.). Для численного анализа данных мы отображаем элементы пространства событий в пространство характеристик X. Выбор способа отображения пространства событий в пространство признаков зависит от конкретной задачи. Этот выбор влияет на эффективность используемых алгоритмов, так как при обнаружении вторжений обычно обрабатываются очень большие наборы данных. В рассматриваемом подходе пространство признаков это пространство заданной размерности. В дальнейшем мы будем рассматривать двумерное пространство характеристик: каждому событию D_i соответствуют две числовые соответствующим образом нормированные характеристики x_i и y_i. В качестве метрики и для оценки степени близости событий D_i и D_j будем использовать евклидово расстояние в пространстве характеристик:

$$d(D_i, D_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Введённая метрика может быть использована как мера близости элементов данных и делает возможным использование методов кластеризации. Рассматриваемый метод кластеризации может быть легко обобщён для случая пространства характеристик другой размерности.

Мы надеемся, что используемая нами метрика позволяет собирать вместе объекты с близкими характеристиками и разделить объекты с отличающимися свойствами. В этом случае после кластеризации мы получим набор кластеров с объектами одного типа в каждом их них. Это соответствует исходному предположению о том, что нормальные и аномальные события различимы. После кластеризации входного набора данных возникает проблема классификации кластеров на содержащие нормальные события и события аномальные (предположительно соответствующие вторжениям). Ранее мы предположили, что в обрабатываемом наборе данных нормальных объектов намного больше чем аномальных. В этом случае разумно предположить, что кластеры, соответствующие нормальным объектам содержат больше вхождений. Эти кластеры пометим как нормальные, остальные кластеры – аномальные. Проблема может появиться в том случае, если набор нормальных событий распадается на множество подтипов нормальной сетевой активности, например при использовании разных протоколов (ftp, telnet, http, …). Каждый из этих подтипов может образовать собственный кластер, это приведёт к появлению большого числа кластеров с малым числом нормальных объектов. Очевидное решение проблемы заключается в изменении пространства признаков, изменении метрики или (если это невозможно) использовать набор данных с достаточно большим числом нормальных событий, так чтобы при разбиении на большое число кластеров число объектов в нормальных кластерах должно получаться большим.

Простой метод кластеризации данных

Будем считать, что каждый кластер характеризуется совокупностью входящих в него объектов, положением центра $X=(x,y)$ и радиусом R:

$$C_k=\left\{X_i, i=1,2,... \;\middle|\; d\left(X^{(k)},X_i\right)\le R^{(k)}\right\}$$

Алгоритм кластеризации представим в виде последовательности шагов:

1. Инициализируем пустой набор кластеров Ω и зададим «затравочный» радиус R_0.

2. Для каждого новой точки $X_i=(x_i,y_i)$ ищем ближайший кластер C_K соответствующего радиуса: $\min d\left(X_i,X^{(k)}\right)\wedge d\left(X_i,X^{(k)}\right)<R^{(k)}$

3. Если такой кластер найден, то включаем X_i в состав этого кластера C_K и пересчитываем положение центра $X^{(k)}$ и радиус $R^{(k)}$ кластера с учётом новой точки.

4. Если подходящий кластер не найден, то создаём новый кластер с центром в точке X_i и «затравочным» радиусом R_0: $C_{new}=\left(X_i,R_0\right)$ и включаем его в набор кластеров Ω.

5. Если центр какого либо кластера оказывается внутри другого, то эти два кластера объединяем в один кластер.

6. Возвращаемся к шагу 2 для обработки следующей точки.

Так как кластер задаётся как совокупность N точек, то положение

центра кластера $X^{(k)} = \left(x^{(k)}, y^{(k)}\right)$ можно определить как среднее арифметическое положений принадлежащих ему точек:

$$x = \frac{1}{N} \cdot \sum_{i=1}^{N} x_i \text{ и } y = \frac{1}{N} \cdot \sum_{i=1}^{N} y_i$$

Радиус кластера, характеризующий степень разброса точек относительно центра кластера, определим как среднеквадратичное отклонение:

$$R^2 = \sigma_x^2 + \sigma_y^2,\ \sigma_x^2 = \frac{1}{N} \sum_{i=1}^{N} \left(x - x_i\right)^2,\ \sigma_y^2 = \frac{1}{N} \sum_{i=1}^{N} \left(y - y_i\right)^2$$

Используя простые преобразования можно показать что:

$$\sigma_x^2 = \frac{1}{N} \sum_{i=1}^{N} x_i^2 - x^2 \text{ и } \sigma_y^2 = \frac{1}{N} \sum_{i=1}^{N} y_i^2 - y^2$$

Таким образом, для нахождения центра и радиуса кластера нет необходимости использовать информацию о положении каждой точки, достаточно знать значения N - числа элементов в кластере и четырёх накапливающих параметров $\sum_{i=1}^{N} x_i$, $\sum_{i=1}^{N} y_i$, $\sum_{i=1}^{N} x_i^2$ и $\sum_{i=1}^{N} y_i^2$

При программной реализации алгоритма кластеризации для описания кластера определим структуру данных:

```
typedef struct tagCLUST {
   float X;      // x - координата центра кластера
   float Y;      // y - координата центра кластера
   float R2;     // R*R - квадрат радиуса кластера
   long int N;   // количество элементов в кластере
   float sx1;    // sum (x)
   float sy1;    // sum (y)
   float sx2;    // sum (x*x)
```

```
   float sy2;    // sum (y*y)
} CLUST;
```

Опишем совокупность кластеров в виде массива:

```
CLUST Clust[100];
```

Введём «затравочный» радиус и текущее число кластеров:

```
float R0 = 150.0;  // "затравочный" радиус кластера
int QntClust = 0;  // число кластеров
```

Теперь определим основные операции работы с кластерами. Операция создания нового кластера с центром в точке $X=(x,y)$ и радиусом R:

```
void NewCluster (float X, float Y, float R) {
   Clust[QntClust].X = X;
   Clust[QntClust].Y = Y;
   Clust[QntClust].R2 = R*R;
   Clust[QntClust].N = 1;
   Clust[QntClust].sx1 = X;
   Clust[QntClust].sy1 = Y;
   Clust[QntClust].sx2 = X*X;
   Clust[QntClust].sy2 = Y*Y;
   ++QntClust;
}
```

Добавление в кластер C_n новой точки $X=(x,y)$:

```
void ModCluster (int n, float X, float Y) {
   Clust[n].N += 1;
   Clust[n].sx1 += X;
   Clust[n].sy1 += Y;
```

```
   Clust[n].sx2 += X*X;
   Clust[n].sy2 += Y*Y;

   Clust[n].X = Clust[n].sx1 / Clust[n].N;
   Clust[n].Y = Clust[n].sy1 / Clust[n].N;
   Clust[n].R2 = Clust[n].sx2/Clust[n].N
                 - Clust[n].X*Clust[n].X
                 + Clust[n].sy2/Clust[n].N
                 - Clust[n].Y*Clust[n].Y;
}
```

Включение кластера C_m в кластер C_n :

```
void ConsCluster (int n, int m) {
   int i;

   Clust[n].N += Clust[m].N;
   Clust[n].sx1 += Clust[m].sx1;
   Clust[n].sy1 += Clust[m].sy1;
   Clust[n].sx2 += Clust[m].sx2;
   Clust[n].sy2 += Clust[m].sy2;

   Clust[n].X = Clust[n].sx1 / Clust[n].N;
   Clust[n].Y = Clust[n].sy1 / Clust[n].N;
   Clust[n].R2 = Clust[n].sx2/Clust[n].N
              - Clust[n].X*Clust[n].X
              + Clust[n].sy2/Clust[n].N
              - Clust[n].Y*Clust[n].Y);

   --QntClust;
```

```
   for (i=m; i<QntClust; ++i)
      memcpy (&Clust[i], &Clust[i+1], sizeof (CLUST));
}
```

Основная функция, реализующая простой алгоритм кластеризации:

```
void TreatEvent (float x, float y) {
   int i, j;
   float d, D;
   int n;

// определяем принадлежность точки соотв. кластеру
   n = -1;
   d = 1.0e+20;
   for (i=0; i<QntClust; ++i) {
      D = (Clust[i].X-x)*(Clust[i].X-x)+(Clust[i].Y-
y)*(Clust[i].Y-y);
      if (D < Clust[i].R2 && D < d) {
         d = D;
         n = i;
      }
   }
// Если нужный кластер не найден
   if (n == -1)
// то создаём новый кластер
      NewCluster (x, y, R0);
   else
// или включаем точку в состав найденного кластера
      ModCluster (n, x, y);

// просматриваем пары кластеров и объединяем их если
необходимо
```

```
   for (i=0; i<QntClust; ++i)
      for (j=0; j<i; ++j) {
         D = (Clust[i].X-Clust[j].X)*(Clust[i].X-
Clust[j].X)+(Clust[i].Y-Clust[j].Y)*(Clust[i].Y-
Clust[j].Y);
         if (D < Clust[i].R2 || D < Clust[j].R2) {
            ConsCluster (j, i);
            i = QntClust;
            j = QntClust;
         }
      }
}
```

На приведённых ниже картинках представлен результат применения рассматриваемого алгоритма к набору данных образующих три, частично перекрывающихся, кластера. В первом случае алгоритм кластеризации распознал три кластера и определил их параметры (центр и радиус). Во втором случае, при большом перекрытии кластеров, два кластера были распознаны как один.

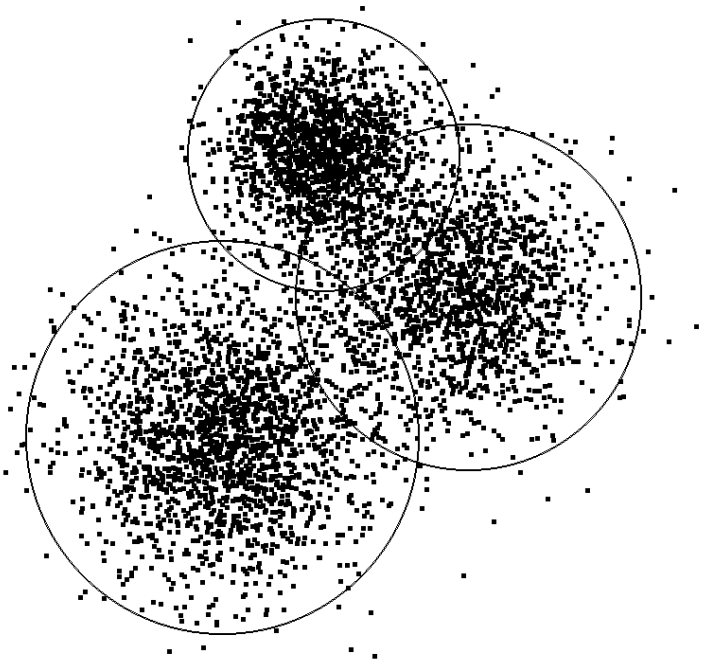


рис 1

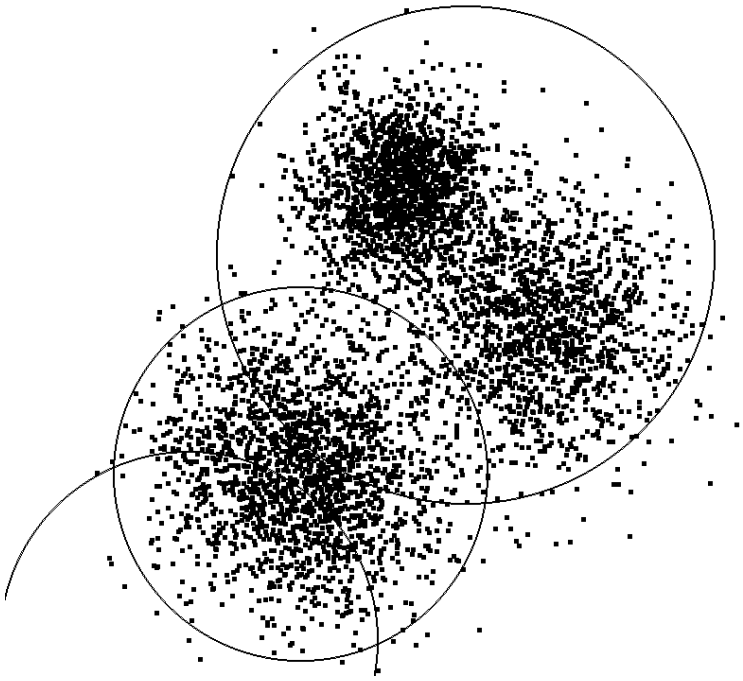


рис 2

Метод кластеризации основанный на плотности

В данной лекции мы используем метод кластеризации основанный на оценке плотности событий в пространстве характеристик. Основная идея данного метода заключается в том, что некоторая область принадлежит кластеру, если плотность в этой области превышает заданную пороговую величину. Подобные методы хорошо подходят для фильтрации выбросов и для создания кластеров произвольной формы.

Введём отношение порядка на множестве событий D: будем считать, что между некоторыми парами событий D_1 и D_2 существует отношение $D_1 \le D_2$, если для соответствующих характеристик $X_1 = (x_1, y_1)$ и $X_2 = (x_2, y_2)$ одновременно выполняются условия $x_1 \le x_2$ и $y_1 \le y_2$. Другими словами: если в двумерном пространстве характеристик точка (x_2, y_2) расположена в первом квадранте относительно точки (x_1, y_1), то $D_1 \le D_2$.

Для накопления, хранения и изменения информации о текущих событиях мы будем использовать бинарное упорядоченное дерево T. Событию D_i соответствует узел дерева $node_i$, в этом узле хранится следующая информация: x_i и y_i - характеристики данного события, n_i^R - количество элементов в правом поддереве и т.п.. Таким образом бинарное дерево можно представить в виде:

$$T = tree(node, T^L, T^R)$$

где $node = \{x, y, n^R, \ldots\}$ - информация о событии, соответствующем данному

узлу дерева, T^L и T^R - левое и правое поддеревья для данного узла.

Так как мы используем упорядоченное дерево, то для каждого элемента правого поддерева выполняется соотношение: $D_i \le D_j$, $D_j \in T_i^R$. В этом случае применимы стандартные алгоритмы включения новых и поиск заданных элементов в бинарном упорядоченном дереве.

Введённые отношения порядка и способ построения дерева позволяют вычислить количество элементов дерева T расположенных в первом квадранте относительно заданной точки (x_0, y_0). Для этой цели определим функцию $count(T, x_0, y_0)$ которая в качестве своего значения возвращает искомое число элементов (определение этой функции дано в приложении). В этом случае плотность данных в окрестности точки (x_0, y_0) может быть вычислена по формуле:

$$\rho(x_0, y_0) = \frac{count(T, x_1, y_1) - count(T, x_2, y_1) - count(T, x_1, y_2) + count(T, x_2, y_2)}{4 \cdot \delta^2}$$ где

$x_1 = x_0 - \delta$, $x_2 = x_0 + \delta$, $y_1 = y_0 - \delta$, $y_2 = y_0 + \delta$.

После того как на основе предварительного набора данных создан набор кластеров, система готова к обнаружению нарушений. Каждое событие D_i обрабатывается следующим образом:

1. После соответствующей нормировки получаем вектор X_i в пространстве событий соответствующий событию D_i.

2. Находим плотность событий в пространстве характеристик в окрестности точки X_i.

3. Классифицируем событие D_i в соответствии с найденным значением плотности как нормальное или аномальное.

В качестве примера практического использования предлагаемого метода расчёта плотности в пространстве событий приведём результаты обработки потока пакетов в сети на одном из серверов факультета математики, механики и компьютерных наук ЮФУ. Log-файл сетевого трафика содержит информацию о более чем 1 млн. пакетов и 200 тыс. нормальных TCP соединениях.

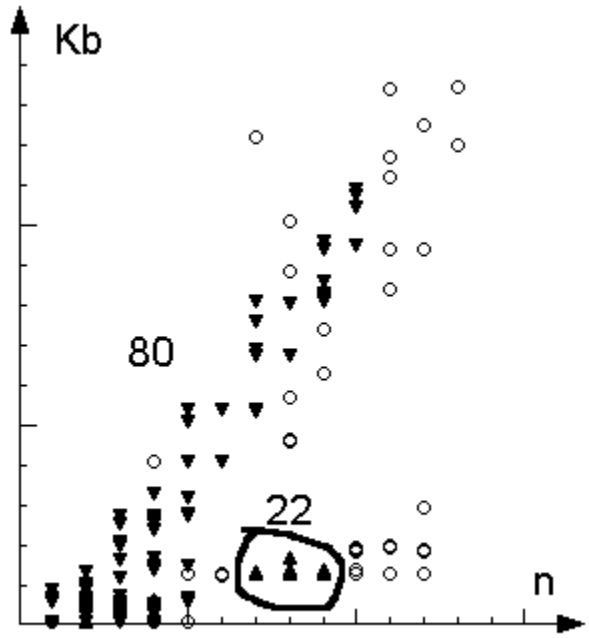


Рис 1

События в сети будем считать установленные и корректно завершённые TCP соединения. В качестве числовых характеристик будем использовать количество пакетов и объём информации переданных в течении одного соединения со стороны узла приёмника соединения. Точки, представленные на рисунках 1-2, соответствуют событиям в пространстве характеристик. Тёмные точки обозначают область высокой плотности событий : $\rho(x,y) \geq \rho_0$, где ρ_0 - средняя плотность в пространстве событий; светлые точки — область низкой плотности. На каждом рисунке присутствует около 5000 событий. Многие точки, представляющие отдельные события, расположены достаточно близко друг к другу, на рисунках они не различимы.

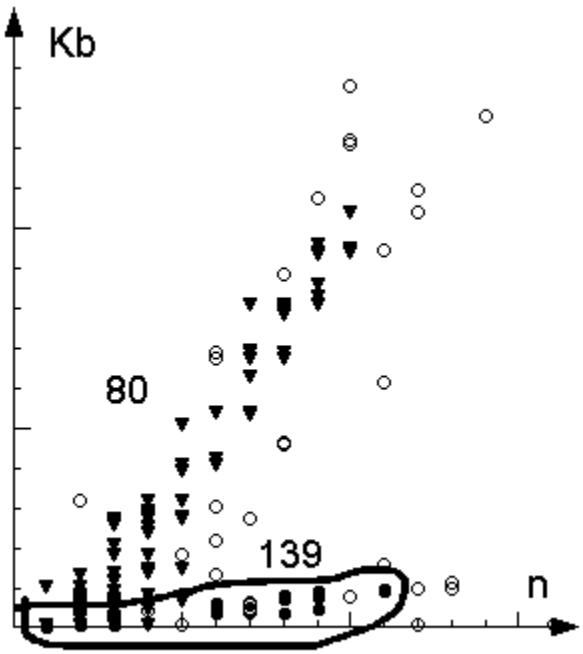


Рис 2

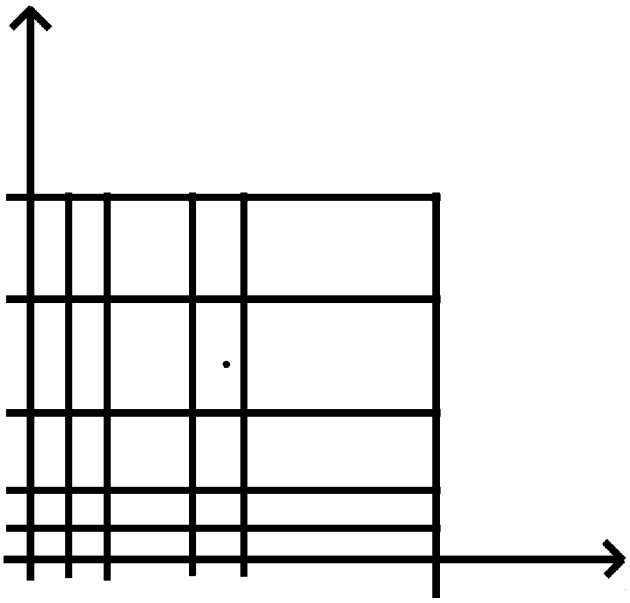
События на Рисунке 1 соответствуют пакетам передаваемым из 80-го (http протокол) и 22-го (ssh протокол), на Рисунке 2 - из 139 и 445 портов (netbios и microsoft-ds сервисы соответственно). Хорошо видна разница в форме кластеров (областей высокой плотности) на этих рисунках. Это обстоятельство позволяет использовать полученные результаты для построения классификационной модели системы [4] и выявления аномальных событий: вначале определяем принадлежность события тому или иному кластеру (если это возможно) а затем определяем соответствие номера порта источника пакета с выбранным кластером.

Сеточные методы

Сеточные методы основаны на том, что пространство событий разбивается на клетки, что даёт возможность держать в памяти не конкретные события, а лишь счётчик событий в данной клетке. Если число событий в клетке велико - она соответствует нормальному состоянию, мало - аномальному. Плотность клетки вычисляем по формуле:

$$\rho_i = \frac{n_i}{\Delta x_i \Delta y_i}$$

где Δx_i , Δy_i - величина клетки n_i - число событий, в неё попавших. Эту плотность сравниваем с пороговой и по результату делаем вывод относительно нормальных или аномальных событий. Может случиться так, что в небольшом числе клеток группируется большинство событий. В таком случае целесообразно использовать **адаптивные сетки**: если число событий в клетке велико - её размер уменьшают, мало - увеличивают. В идеале числа событий в каждой клетке равны.



Другой способ построения адаптивных сеток - **иерархический**: большой квадрат разбивается на ещё меньшие.

Классификационные методы выявления аномалий

Метод классификации заключается в создании правил отображения элементов данных на предопределённые классы. Результатом работы таких алгоритмов будет “классификатор”, например, в форме дерева принятия решений или правил. Идеальным приложением для определения вторжений будет сбор достаточного объёма “нормальных” и “аномальных” данных и использовать классификационный алгоритм для обучения классификатора, который впоследствии будет разделять входные данные на класс нормальных и класс аномальных элементов. В этой лекции мы рассмотрим методику применения классификационных методов для выделения сетевых атак из нормального трафика при обработке данных перехваченных при помощи программы **`TCPdump`**. Программа **`TCPdump`** запускается на шлюзе, который соединяет локальную сеть (LAN) и внешнюю сеть. Таким образом, собирается сетевой трафик между LAN и внешней сетью и широковещательные пакеты внутри LAN. Фильтры **`TCPdump`** настраиваются так, что перехватываются только пакеты протоколов TCP и UDP. Мы рассмотрим методику построения модели выявления аномалий из входного набора данных.

Перед применением методов классификации данных проведём предварительную обработку данных для выявления очевидных нарушений, не требующих применения методов аналитической обработки данных. Для каждого TCP соединения между двумя портами конкретных узлов выполняем следующие проверки:

- ✓ Проверяем правильность “тройного рукопожатия” при создании соединения. Фиксируются следующие ошибки: не принятое соединение (инициатор соединения не получает ACK подтверждения); по-

пытка соединения без установки соединения (инициатор соединения не получает SYN подтверждения); получение ненужного SYN подтверждения (не было запроса на соединение – первого SYN пакета).

- ✓ Контролируем каждый пакет данных и ACK пакет. Поддерживает счётчики для сбора статистики о соединении: оценка дублирования ACK, оценка неправильных размеров пакетов, количество байт в каждом направлении, процент пакетов данных и управляющих пакетов.
- ✓ Проверяем завершение каждого соединения: нормальное (обе стороны корректно посылают и получают FIN пакеты), прерывание соединения (одна сторона посылает RST пакет, все пакеты данных должным образом подтверждаются), половинное закрытие (только одна сторона посылает FIN), разрыв соединения.

Программа для выделения TCP-соединений из результатов применения пакета **TCPdump** приведена в одной из предыдущих лекций в разделе «Программная обработка результатов мониторинга». Так как UDP протокол не требует соединения, мы можем трактовать каждый пакет как отдельное соединение.

После предварительной обработки каждый элемент входных данных представляет собой запись о соединении и может иметь следующие поля (признаки, характеристики): время начала соединения, продолжительность, адреса узлов, порты, статистики соединения (продолжительность соединения, количество пакетов и объём информации переданной в обоих направлениях и т.п.), флаг (“нормальное” или ошибки соединения/завершения) и протокол (TCP или UDP).

Для получения дополнительной информации о сетевой активности и

повышения точности используемого метода выявления нарушений в каждую запись о соединении можно ввести дополнительные поля. Например, за последние *t* секунд соберём статистическую информацию о следующих величинах: Общее число соединений устанавливаемых с ошибками. Число других типов ошибок. Число соединений относящихся к системным сервисам (*ftp, http, ...*). Число соединений пользовательских приложений. Число соединения той же самой службы, что и текущее соединение. Средняя продолжительность и число байт данных (в обоих направлениях) для всех соединений и те же самые средние для текущей службы. Величина временного интервала *t* подбирается экспериментальным путём. Установлено, что что изменения значений *t* выше 30 секунд практически не влияют на результаты применения рассматриваемого метода.

В дальнейшем мы будем называть инициирующий соединение узел (узел пославший первый SYN пакет) источником и другой узел – приёмником соединения. В зависимости от направления от источника к приёмнику соединение может быть одним из трёх типов: исходящее (out-going) – из LAN во внешнюю сеть; входящее (in-coming) из внешней сети в LAN; внутреннее (inter-LAN) – внутри LAN. Раздельный анализ этих типов соединений и конструирование соответствующих моделей определения может улучшать точность определений вторжений.

Для применения методов классификации данных в записях о соединениях мы будем использовать порт приёмника соединения как метку класса и остальные характеристики соединения как атрибуты. Теперь проведём построение классификатора (формулирование правил классификации) на тренировочном наборе данных. После обучения системы на тренировочном наборе данных мы получим набор правил, позволяющих определить класс элемента данных по его атрибутам.

Полученный набор правил можно применять к входному потоку данных: если предсказанный класс очередного элемента данных совпадает с портом приёмника соединения, то этот элемент данных считаем “нормальным” элементом, в противном случае элемент данных является “аномальным” и может нести в себе угрозу вторжения или сетевой атаки. Доля данных с ошибочной классификацией должна быть низкой для данных соответствующих правильным соединениям и высокой для данных с нарушениями.

Одним из способов построения «классификатора» событий в сети может быть использование методов кластерного анализа рассмотренных в предыдущей лекции. Как и прежде мы используем результаты обработки потока пакетов в сети на одном из серверов факультета математики, механики и компьютерных наук ЮФУ.

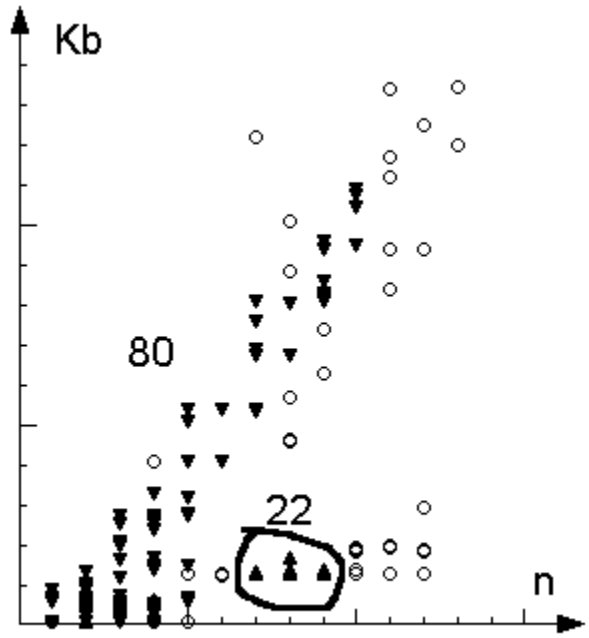


Рис 1

События в сети мы считаем установленные и корректно завершённые TCP-соединения. В качестве числовых характеристик используем количество пакетов и объём информации переданных в течении одного сеанса со стороны узла приёмника соединения. Точки, представленные на рисунке 1, соответствуют событиям в пространстве характеристик. Тёмные точки обозначают область высокой плотности событий : $\rho(x,y) \geq \rho_0$, где ρ_0 - средняя плотность в пространстве событий; светлые точки — область низкой плотности.

Из приведённых данных видно, что кластеры, соответствующие разным сетевым сервисам, разделены в пространстве характеристик. И, следовательно, зафиксированные события, соответствующие http или ssh соединениям, должны должны попадать в разные области пространства характеристик. Если вдруг TCP-соединение с 80-м портом попадёт в область кластера 22-го порта, то это, скорее всего, будет признаком туннелирования ssh-соединения через http-соединение.

Оценка эффективности метода выявления нарушений

Исследуемая система (сеть, вычислительная система, отдельный компьютер) может находиться в двух состояниях: в нормальном (если нет вторжений) и в аномальном (если вторжение произошло). Но иногда аномальное состояние может быть принято за нормальное, а нормальное за аномальное. Отсюда возникают четыре варианта состояний системы:

- **True Negative** – нет аномалий, и состояние системы также считается нормальным
- **False Negative** - аномалии есть, но состояние системы считается нормальным
- **True Positive** – есть атака, и она обнаружена
- **False Positive** – нет атак, но состояние системы считается аномальным

Для численных оценок качества системы обнаружения вторжений вводятся относительные величины:

$$FalsePositiveRate = \frac{число\_событий\_отмеченных\_как\_атака}{общее\_число\_нормальных\_событий}.$$

- эта величина описывает ложные срабатывания системы; чем она меньше, тем лучше оценивается состояние системы.

$$TruePositiveRate = \frac{число\_выявленных\_атак}{общее\_число\_атак}.$$

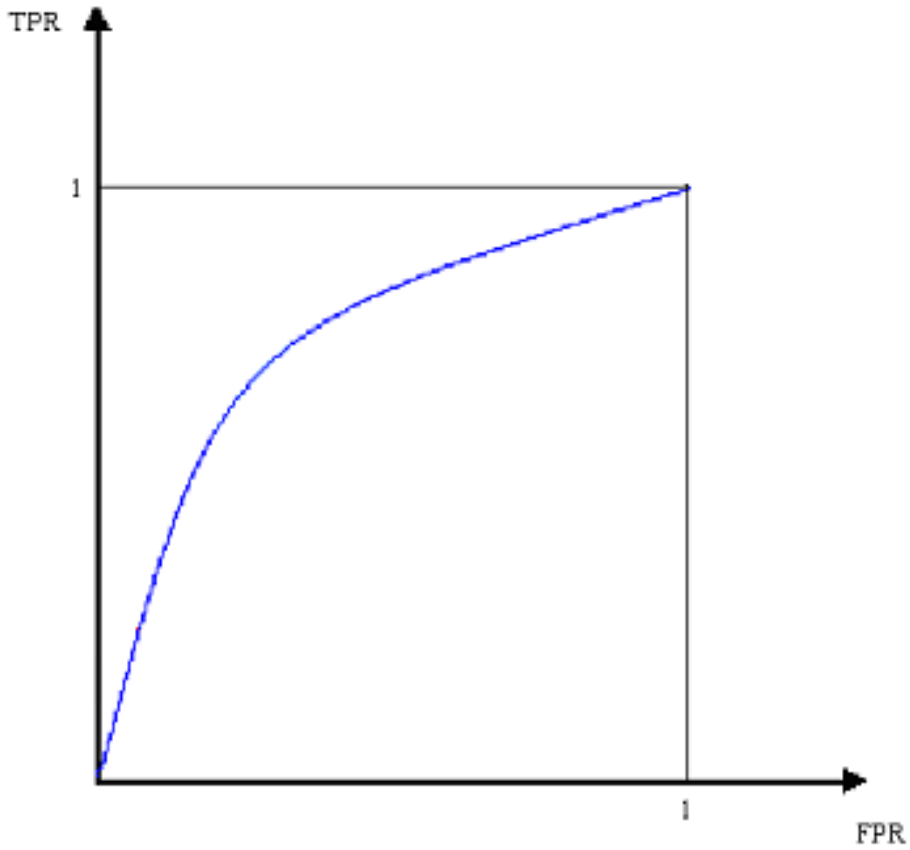
- эта величина описывает количество правильно предсказанных атак; чем она больше, тем лучше оценивается состояние системы.

Эти метрики для оценки качества систем обнаружения вторжений можно представить в виде таблицы:

Метрика		**Предсказанное состояние системы**	
		Normal	**Attacks**
Реальное состояние системы	**Normal**	**True Negative Rate**	**False Positive Rate (false alarm)**
	Attacks	**False Negative Rate**	**True Positive Rate (correctly detected attacks)**

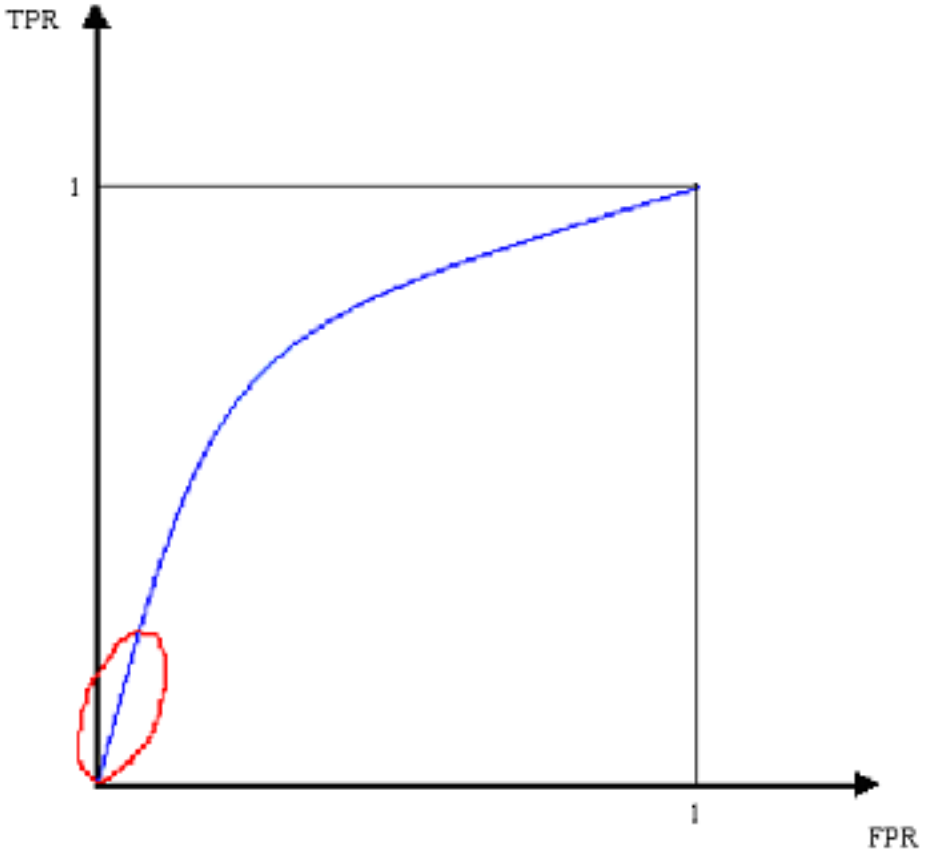
Идеальной системой считается та, для которой $FalsePositiveRate \to 0$ и $TruePositiveRate \to 1$.

Эти параметры взаимосвязаны. Например, для заданного метода обнаружения вторжений, в зависимости от свободных параметров модели (например порога срабатывания), на тестовом наборе данных строится такой график:



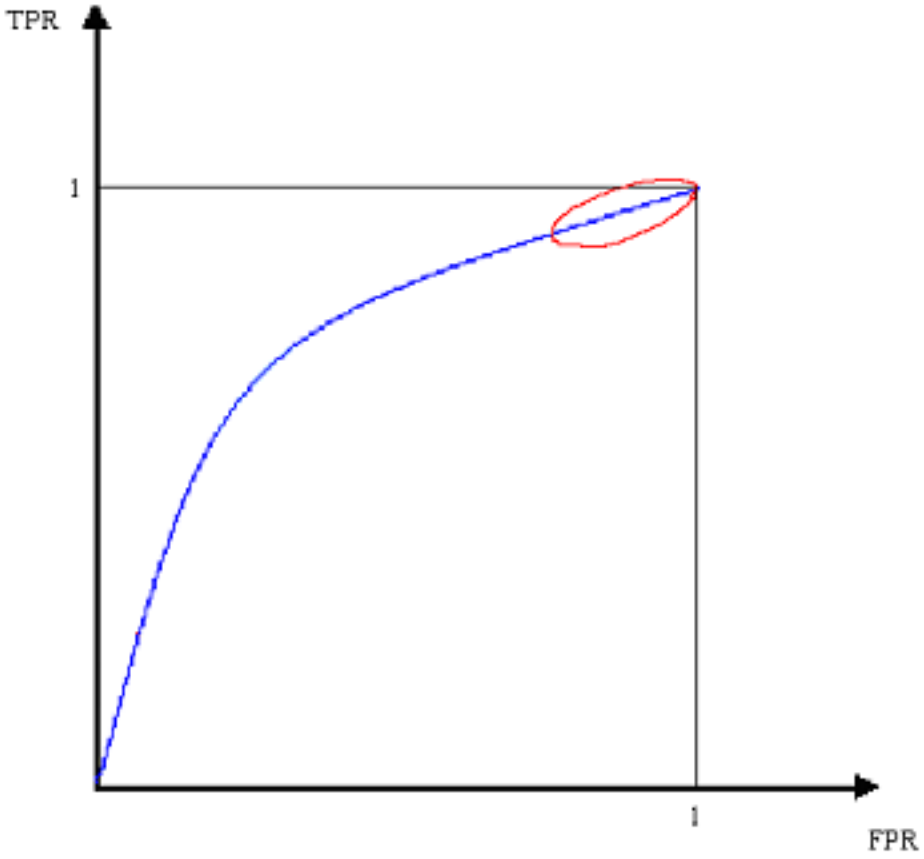
Такие графики называются ROC – кривыми (ROC – диаграммами), они полезны при выборе параметров системы обнаружения вторжений и оценки её качества.

В общем случае нам необходимо свести к минимуму число ложных срабатываний при наибольшее число правильно выявленных нарушений, необходимо подобрать оптимальное соотношение $FalsePositiveRate$ и $TruePositiveRate$.



В выделенной на графике области вблизи нулевых значений TPR и FPR производная функции больше единицы и небольшое увеличение ложных срабатываний (FPR) приводит к существенному увеличению корректно выявленных атак (TPR). Поэтому перемещение от 0 в область больших значений FPR повышает эффективность системы.

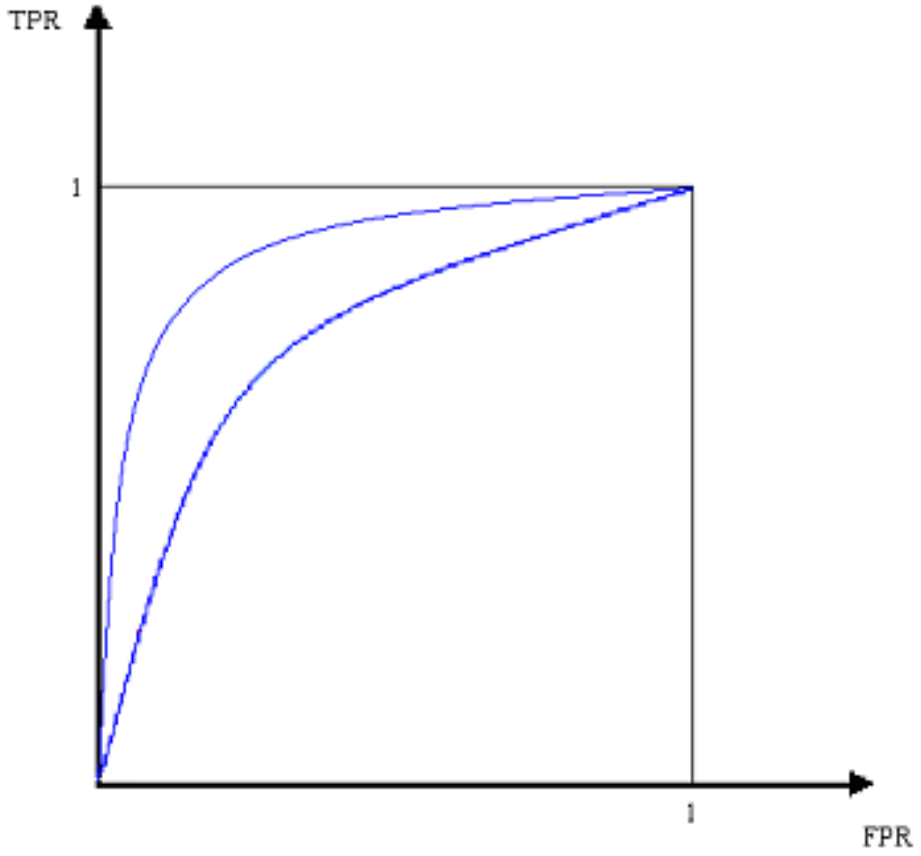
В случае значений TPR и FPR вблизи 1



производная функции меньше единицы небольшое уменьшение правильно выявленных атак (TPR) приводит к сильному снижению ложных срабатываний (FPR). В этом случае для повышения эффективности системы выгодно переместиться в область меньших значений TPR и FPR.

Таким образом оптимальная область находится в середине ROC-кривой, где производная функции близка к 1.

При сравнении эффективности разных систем обнаружения вторжений также можно использовать ROC-кривые. Для этого на графике строятся кривые разных систем на одинаковой последовательности событий.



Лучшей считается та система, у которой график лежит выше, т.к. при равном числе правильных срабатываний ложных срабатываний у нее меньше.

Анализ результатов и предотвращение вторжений

По результатам работы системы обнаружения нарушений сама система или администратор принимает решения направленные на предотвращение выявленных нарушений в дальнейшем. Обычно эти действия сводятся к пополнению базы данных сигнатур атак и настройки сетевых фильтров для блокирования вредоносного трафика.

Firewall’ы защищают компьютеры и сети от попыток несанкционированного доступа с использованием уязвимых мест, существующих в семействе протоколов TCP/IP. Дополнительно они помогают решать проблемы безопасности, связанные с использованием уязвимых систем и с наличием большого числа компьютеров в локальной сети. Существует несколько типов firewall’ов, начиная от пакетных фильтров, которые обеспечивают управление доступом для IP-пакетов, до мощных firewall’ов, которые могут закрывать уязвимости в большом количестве уровней семейства протоколов TCP/IP, и еще более мощных firewall'ов, которые могут фильтровать трафик на основании всего содержимого пакета.

Общий обзор firewall’ов

Firewall’ы являются устройствами или системами, которые управляют потоком сетевого трафика между сетями с различными требованиями к безопасности. В большинстве современных приложений firewall’ы и их

окружения обсуждаются в контексте соединений в Интернете и, следовательно, в контексте стека протоколов TCP/IP. Однако firewall'ы применяются и в сетевых окружениях, которые не требуют обязательного подключения к Интернету. Например, многие корпоративные сети предприятия ставят firewall'ы для ограничения соединений из и во внутренние сети, обрабатывающие информацию разного уровня чувствительности, такую как бухгалтерская информация или информация о заказчиках. Ставя firewall'ы для контроля соединений с этими областями можно предотвратить неавторизованный доступ к соответствующим системам и ресурсам внутри чувствительных областей. Тем самым, использование firewall'а обеспечивает дополнительный уровень безопасности, который иначе не может быть достигнут.

В настоящее время существует несколько типов firewall'ов. Одним из способов сравнения их возможностей является перечисление уровней модели OSI, которые данный тип firewall'а может анализировать. Рассмотрим уровни модели OSI, относящиеся к firewall'ам.

Стек протоколов модели OSI определяется следующим образом:

Стек протоколов модели OSI	
Уровень 7	Прикладной
Уровень 6	Представительный
Уровень 5	Сеансовый
Уровень 4	Транспортный
Уровень 3	Сетевой
Уровень 2	Канальный
Уровень 1	Физический

Уровень 1 представляет собой реальную аппаратуру физического соединения и среду, такую как Ethernet. Уровень 2 — уровень, на котором сетевой трафик передается по локальной сети (LAN). Он также является первым уровнем, обладающим возможностью адресации, с помощью которой можно идентифицировать отдельную машину. Адреса назначаются на сетевые интерфейсы и называются MAC (Media Access Control) адресами. Ethernet-адрес, принадлежащий Ethernet-карте, является примером MAC-адреса уровня 2. Уровень 3 является уровнем, отвечающим за доставку сетевого трафика по WAN. В Интернете адреса уровня 3 называются IP-адресами; адреса обычно являются уникальными, но при определенных обстоятельствах, например, при трансляции сетевых адресов (NAT) возможны ситуации, когда различные физические системы имеют один и тот же IP-адрес уровня 3. Уровень 4 идентифицирует конкретное сетевое приложение и коммуникационную **сессию** в дополнение к сетевым адресам; система может иметь большое число сессий уровня 4 с другими ОС. Терминология, связанная с семейством протоколов TCP/IP, включает понятие **портов**, которые могут рассматриваться как конечные точки сессий: номер порта **источника** определяет коммуникационную сессию на исходной системе; номер порта **назначения** определяет коммуникационную сессию системы назначения. Более высокие уровни (5, 6 и 7) представляют приложения и системы конечного пользователя.

Пакетные фильтры

Самый основной, базовый, первоначально разработанный тип

firewall’a называется пакетным фильтром. Пакетные фильтры в основном являются частью устройств роутинга, которые могут управлять доступом на уровне системных адресов и коммуникационных сессий. Функциональность управления доступом обеспечивается с помощью множества директив, называемых ruleset или rules (правила). Изначально пакетные фильтры функционировали на уровне 3 (Network) модели OSI. Данная функциональность разработана для обеспечения управления сетевым доступом, основываясь на нескольких блоках информации, содержащихся в сетевом пакете. В настоящее время все пакетные фильтры также анализируют и уровень 4 (Transport).

Пакетные фильтры анализируют следующую информацию, содержащуюся в заголовках пакетов 3-го и 4-го уровней:

- **Адрес источника пакета**, например, адрес уровня 3 системы или устройства, откуда получен исходный сетевой пакет (IP-адрес, такой как 192.168.1.1).

- **Адрес назначения пакета**, например, адрес уровня 3 пакета, который он пытается достигнуть (например, 192.168.1.2).

- **Тип коммуникационной сессии**, т.е. конкретный сетевой протокол, используемый для взаимодействия между системами или устройствами источника и назначения (например, TCP, UDP или ICMP).

- Возможно некоторые **характеристики коммуникационных сессий уровня 4**, такие как порты источника и назначения сессий (например, TCP:80 для порта назначения, обычно принадлежащий web-серверу, TCP:1320 для порта источника, принадлежащий персональному компьютеру, который осуществляет доступ к серверу).

- Иногда информация, относящаяся к **интерфейсу роутера**, на который при-

шел пакет, и информация о том, какому интерфейсу роутера она предназначена; это используется для роутеров с тремя и более сетевыми интерфейсами.

•Иногда информация, характеризующая **направление**, в котором пакет пересекает интерфейс, т.е. входящий или исходящий пакет для данного интерфейса.

•Иногда можно также указать свойства, относящиеся к созданию **логов** для данного пакета.

Пакетные фильтры обычно размещаются в сетевой инфраструктуре, использующей TCP/IP. Однако они могут также быть размещены в любой сетевой инфраструктуре, которая имеет адресацию уровня 3, например, IPX (Novell NetWare) сети. В современных сетевых инфраструктурах firewall'ы на уровне 2 могут также использоваться для обеспечения балансировки нагрузки и/или в приложениях с высокими требованиями к доступности, в которых два или более firewall'а используются для увеличения пропускной способности или для выполнения восстановительных операций.

Некоторые пакетные фильтры, встроенные в роутеры, могут также фильтровать сетевой трафик, основываясь на определенных характеристиках этого трафика, для предотвращения DoS- и DdoS-атак.

Преимущества пакетных фильтров:

- Основным преимуществом пакетных фильтров является их скорость.
- Пакетный фильтр прозрачен для клиентов и серверов, так как не разрывает TCP-соединение.

Недостатки пакетных фильтров:

- Так как пакетные фильтры не анализируют данные более высоких уровней, они не могут предотвратить атаки, которые используют уязвимости или функции, специфичные для приложения. Например, пакетный фильтр не может блокировать конкретные команды приложения; если пакетный фильтр разрешает данный трафик для приложения, то все функции, доступные данному приложению, будут разрешены.
- Так как firewall'у доступна ограниченная информация, возможности логов в пакетных фильтрах ограничены. Логи пакетного фильтра обычно содержат ту же информацию, которая использовалась при принятии решения о возможности доступа (адрес источника, адрес назначения, тип трафика и т.п.).
- Большинство пакетных фильтров не поддерживают возможность аутентификации пользователя. Данная возможность обеспечивается firewall'ами, анализирующими более высокие уровни.
- Они обычно уязвимы для атак, которые используют такие проблемы TCP/IP, как **подделка (spoofing) сетевого адреса**. Многие пакетные фильтры не могут определить, что в сетевом пакете изменена адресная информация уровня 3 OSI. Spoofing-атаки обычно выполняются для обхода управления доступа, осуществляемого firewall'ом.
- При принятии решений о предоставлении доступа используется небольшое количество информации.
- Пакетные фильтры трудно конфигурировать. Можно случайно переконфигурировать пакетный фильтр для разрешения типов трафика, источников и назначений, которые должны быть запрещены на основе политики безопасности организации.

Таким образом, пакетные фильтры больше всего подходят для высокоскоростных окружений, когда создание логов и аутентификация пользователя для сетевых ресурсов не столь важна.

Персональные firewall'ы и персональные устройства firewall'а

Обеспечение безопасности персональных компьютеров дома или в мобильном варианте сейчас становится столь же важным, как и обеспечение безопасности компьютеров в офисе; домашние пользователи, подключающиеся по dial-up к провайдеру, могут обеспечить защиту с помощью небольшого firewall'а, доступного им. Это необходимо, потому что провайдер может иметь много различных политик безопасности, не всегда соответствующих нуждам конкретного пользователя. Тем самым персональные firewall'ы разрабатываются для обеспечения защиты удаленных систем и выполняют во многом те же самые функции, что и большие firewall'ы.

Эти программные продукты обычно реализованы в одном из двух вариантов. Первый вариант представляет собой **Personal Firewall**, который инсталлируется на защищаемую систему; персональные firewall'ы обычно не предполагают защиту каких-либо других систем или ресурсов. Более того, персональные firewall'ы обычно не обеспечивают управление сетевым трафиком, который проходит через компьютер – они только защищают систему, на которой они инсталлированы.

Второй вариант называется устройством персонального firewall'а, чей подход более похож на традиционный firewall. В большинстве случаев **устрой-**

ства персонального firewall’а разрабатываются для защиты небольших сетей, таких как домашние сети. Эти устройства обычно изготавливаются на специализированной аппаратуре и имеют некоторые другие компоненты сетевой архитектуры в дополнение к самому firewall’у, включая следующие:

- WAN-роутер кабельного модема;
- LAN-роутер (поддержка динамического роутинга);
- сетевой концентратор (hub);
- сетевой коммутатор (switch);
- DHCP;
- агент SNMP;
- агенты прикладного прокси.

Размещение этих инфраструктурных компонент в устройстве firewall’а позволяет использовать единственное аппаратное устройство для эффективного решения нескольких задач.

Хотя персональные firewall’ы и устройства персональных firewall’ов теряют некоторые преимущества и возможности масштабируемости традиционных firewall’ов, они могут быть достаточно эффективны для обеспечения общей безопасности организации. Персональные firewall’ы и устройства персональных firewall’ов обычно предназначены для филиалов офисов. Тем не менее некоторые организации используют эти устройства для создания Интранета, обеспечивая стратегию обороны вглубь. Персональные firewall’ы и устройства персональных firewall’ов могут также использоваться как конечные точки VPN.

Удобство управления устройством или приложением является важным фак-

тором при оценки или выборе персонального firewall'а и устройства персонального firewall'а. Идеально, когда управление позволяет реализовать определяемую организацией политику безопасности на всех системах, которые присоединяются к сети и системе. Поэтому управление персональным firewall'ом или устройством персонального firewall'а должно быть по возможности централизовано. Это позволяет организации реализовывать определенную политику и поддерживать согласованное состояние систем, которые подсоединены удаленно. Наилучший способ обеспечить подобную функциональность является создание профиля безопасной конфигурации, который сопровождает конечного пользователя при его входе на любую систему. При таком подходе политика безопасности всегда будет эффективно реализовываться в момент доступа пользователя к ресурсам.

Что можно сказать об удаленных пользователях, которые подсоединяются к dial-in серверу организации или к провайдерам? Если политика безопасности провайдера является менее ограничительной, чем в организации, то риск, что компьютер будет инфицирован вирусом или будет выполнена другая атака, будет больше. Существует также проблема, что многие пользователи используют свои персональные компьютеры как для работы, так и для целей, далеких от служебных.

Одно из возможных решений состоит в использовании отдельных компьютеров в офисе и вне офиса.

Если такое решение недоступно, то персональный firewall должен использоваться все время и должен быть сконфигурирован с учетом наиболее ограничительных установок, используемых в организации. Если, например, разделение файлов в Windows запрещено firewall'ом, то это должно оставаться запрещенным, даже если компьютер используется в нерабочих целях. Также, если установки web-безопасности отвергают определенные

типы содержимого, данный запрет должен оставаться в силе все время. Такая политика должна быть реализована для dial-in сервера; он, в свою очередь, должен быть размещен таким образом, чтобы firewall и прокси фильтровали входящий трафик от dial-in соединений.

Прокси-сервер прикладного уровня

Прокси прикладного уровня являются более мощными firewall'ами, которые комбинируют управление доступом на низком уровне с функциональностью более высокого уровня (уровень 7 – Application). При использовании firewall'а прикладного уровня обычно, как и в случае пакетного фильтра не требуется дополнительное устройство для выполнения роутинга: firewall выполняет его сам. Все сетевые пакеты, которые поступают на любой из интерфейсов firewall'а, находятся под управлением этого прикладного прокси. Прокси-сервер имеет набор правил управления доступом для определения того, какому трафику может быть разрешено проходить через firewall.

Аутентификация пользователя может иметь много форм, например такие:

- с помощью User ID и пароля;
- с помощью аппаратного или программного токена;
- по адресу источника;
- биометрическая аутентификация.

Прокси-сервер, который анализирует конкретный протокол прикладного уровня, называется агентом прокси.

Firewall’ы прикладного уровня имеют много преимуществ по сравнению с пакетными фильтрами и stateful inspection пакетными фильтрами.

Преимущества прокси-сервера **прикладного уровня:**

- Прокси имеет возможность запросить аутентификацию пользователя. Часто существует возможность указывать тип аутентификации, который считается необходимым для данной инфраструктуры. Прикладные прокси имеют возможность аутентифицировать самих пользователей, в противоположность пакетным фильтрам и stateful inspection пакетным фильтрам, обычно проверяющим только адрес сетевого уровня, с которого пришел пользователь. Эти адреса сетевого уровня могут быть легко подменены без обнаружения подмены пакетным фильтром.
- Возможности аутентификации, свойственные архитектуре прикладного уровня, лучше по сравнению с теми, которые существуют в пакетных фильтрах и stateful inspection пакетных фильтрах. Благодаря этому прикладные прокси могут быть сделаны менее уязвимыми для атак подделки адреса.
- Firewall’ы прикладного уровня обычно имеют больше возможностей анализировать весь сетевой пакет, а не только сетевые адреса и порты. Например, они могут определять команды и данные, специфичные для каждого приложения.
- Как правило, прокси прикладного уровня создают более подробные логи.

Более развитая функциональность прикладного прокси имеет также несколько недостатков по сравнению с пакетными фильтрами и stateful inspection пакетными фильтрами.

Недостатки прокси-сервера **прикладного уровня:**

- Так как прикладные прокси "знают о пакете все", firewall вынужден тратить много времени для чтения и интерпретации каждого пакета. По этой причине прикладные прокси обычно не подходят для приложений, которым необходима высокая пропускная способность, или приложений реального времени. Чтобы уменьшить загрузку firewall’а, может использоваться выделенный прокси-сервер для обеспечения безопасности менее чувствительных ко времени сервисов, таких как e-mail и большинство web-трафика.
- Прикладные прокси обрабатывают ограниченное количество сетевых приложений и протоколов и не могут автоматически поддерживать новые сетевые приложения и протоколы. Для каждого прикладного протокола, который должен проходить через firewall, необходим свой агент прокси. Большинство производителей прикладных прокси предоставляют общих агентов прокси для поддержки неизвестных сетевых приложений или протоколов. Однако эти общие агенты не имеют большинства преимуществ прикладных прокси: как правило, они просто туннелируют трафик через firewall.

Выделенные прокси-серверы

Выделенные прокси-серверы отличаются от прикладных прокси в том, что они анализируют трафик только конкретного прикладного протокола и не обладают возможностями анализа всего трафика, что все-таки характерно для firewall’а прикладного уровня. По этой причине они обычно развертываются позади firewall’ов прикладного уровня. Типичное исполь-

зование таково: основной firewall получает входящий трафик, определяет, какому приложению он предназначен, и затем передает обработку конкретного типа трафика соответствующему выделенному прокси-серверу, например, e-mail прокси серверу. Выделенный прокси-сервер при этом выполняет операции фильтрации и логирования трафика и затем перенаправляет его во внутренние системы. Этот сервер может также принимать исходящий трафик непосредственно от внутренних систем, фильтровать трафик и создавать логи, а затем передавать его firewall'у для последующей доставки. Обычно выделенные прокси-серверы используются для уменьшения нагрузки на firewall и выполнения более специализированной фильтрации и создания логов.

Как и в случае прикладных прокси, выделенные прокси позволяют выполнить аутентификацию пользователей. В случае использования выделенного прокси легче более точно ограничить исходящий трафик или проверять весь исходящий и входящий трафик, например, на наличие вирусов. Выделенные прокси-серверы могут также помочь в отслеживании внутренних атак или враждебного поведения внутренних пользователей. Фильтрация всего исходящего трафика сильно загружает общий firewall прикладного уровня и увеличивает стоимость администрирования.

В дополнение к функциям аутентификации и создания логов, выделенные прокси-серверы используются для сканирования web и e-mail содержимого, включая следующие функции:

- фильтрование Java-апплетов или приложений;
- фильтрование управлений ActiveX;
- фильтрование JavaScript;

- блокирование конкретных MIME-типов – например, "application/msword";
- сканирование и удаление вирусов;
- блокирование команд, специфичных для приложения, например, блокирование HTTP-команды PUT;
- блокирование команд, специфичных для пользователя, включая блокирование некоторых типов содержимого для конкретных пользователей.

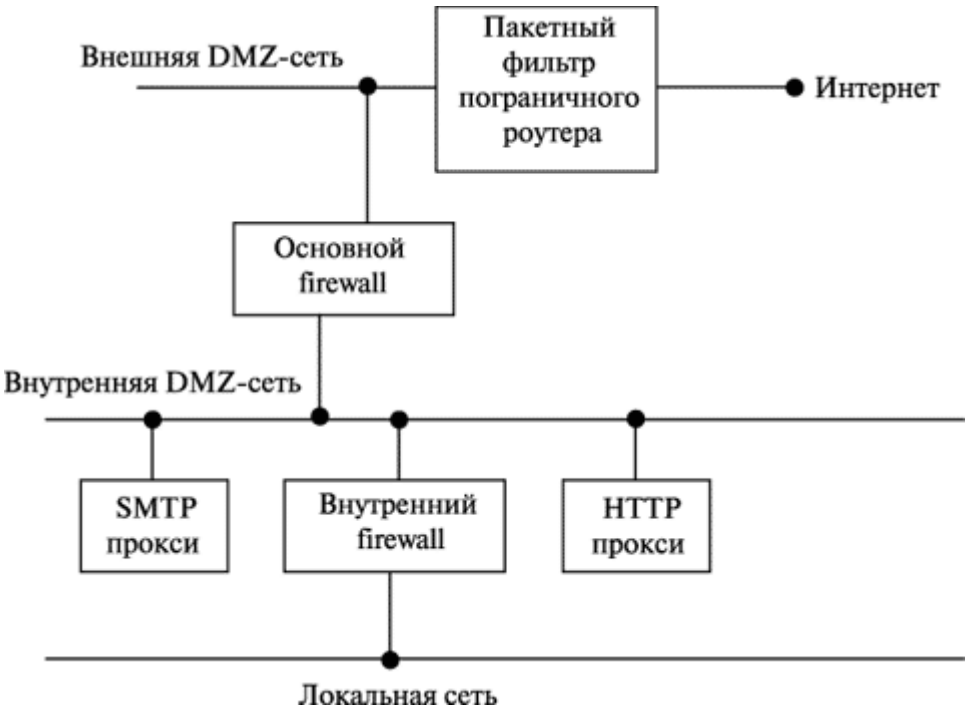


Рис. 1.4. Примеры выделенных прикладных прокси-серверов

На рис 1.4 показан пример топологии сети, которая имеет выделенные прокси-серверы для HTTP и e-mail, расположенные позади основного firewall'a. В этом случае e-mail прокси может быть SMTP-шлюзом организации для входящей почты. Основной firewall будет перенаправлять входящую почту к прокси для сканирования содержимого, после чего почта может становиться доступной внутренним пользователям на SMTP-сервере, например, по протоколам POP3 или IMAP. HTTP-прокси должен обрабаты-

вать исходящие соединения ко внешним web-серверам и, возможно, фильтровать активное содержимое. Прокси-сервером может выполняться кэширование часто используемых web-страниц, тем самым уменьшая трафик к firewall'у.

Гибридные технологии firewall'а

Последние разработки в области сетевой инфраструктуры и информационной безопасности привели к стиранию границ между различными типами firewall'ов, которые обсуждались выше. Как результат, программные продукты firewall'ов соединяют функциональности нескольких различных типов firewall'ов. Например, многие производители прикладных прокси реализуют базовую функциональность пакетных фильтров.

Также многие разработчики пакетных фильтров или stateful inspection пакетных фильтров реализуют базовую функциональность прикладных прокси для ликвидации слабых мест, связанных с этими типами firewall'ов. В большинстве случаев производители реализуют прикладные прокси для улучшения создания логов и аутентификации пользователя.

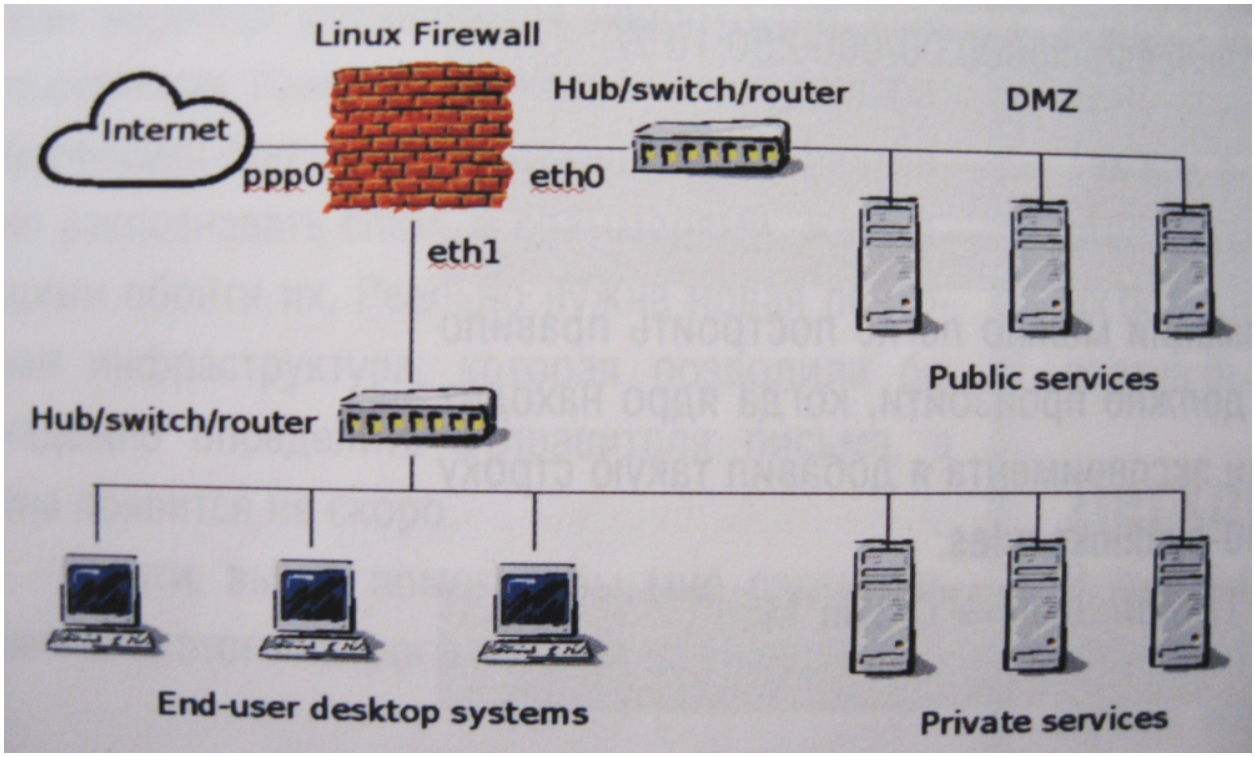
После того как все основные производители будут использовать в своих продуктах в том или ином виде гибридные технологии, не всегда просто будет решить, какой продукт наиболее подходит для данного приложения или инфраструктуры предприятия. Гибридные свойства платформ firewall'ов делают особенно важной фазу оценки firewall'а. При выборе продукта важнее оценить поддерживаемое количество возможностей, чем формально смотреть тип firewall'а.

Пакетный фильтр iptables.

Firewall (брандмауэр) - это аппаратное устройство или программное средство, которое защищает локальную сеть от атак, исходящих из Интернета. Firewall может быть использован для разделения сети на несколько изолированных подсетей или предотвращения подключения пользователей пользователей локальной сети к некоторым узлам Интернета.

Ниже приведена схема типичного использования Firewall'a для разделения сети на три зоны с разными уровнями безопасности и доступа:

1. Зона Интернет (Интернет начинается на дальнем интерфейсе маршрутизатора интернет-провайдера. Мы не можем контролировать какие-либо аспекты функционирования Интернета).
2. Демилитаризованная зона (DMZ содержит подконтрольные нам ресурсы и служит буфером между внутренней сетью и Интернетом. В эту зону выносятся те службы, для которых открыт доступ из сети Интернет).



3. Доверенная зона локальной сети (Доверенная сеть содержит компоненты требующие максимальной степени защиты и не имеет служб открытых для доступа из внешнего мира). Существует два основных типа брандмауэров: пакетные фильтры (packet filters) и брандмауэры proxy.

Пакетные фильтры анализируют заголовки пакетов (разных уровней стека протоколов) и на основе этой информации принимают решение о дальнейшей судьбе пакета.

Брандмауэр proxy не передают пакеты между изолированными сетями напрямую, а получив запрос от клиента самостоятельно выполняет запрос к серверу от имени клиента, исходные IP-пакеты от клиента не попадают к серверу. Proxy блокирует сетевой трафик и передача данных происходит на прикладном уровне стека протоколов.

В ОС Linux используется пакетный фильтр iptables (или ipchains, для

FreeBSD – ipfw). Пакетный фильтр iptables реализован в виде набора правил, на основе этих правил принимается решение о дальнейшей судьбе пакета: принять, отклонить или передать пакет для дальнейшей обработки.

Правила iptables собраны в три таблицы: Filter, Nat и Mangle.

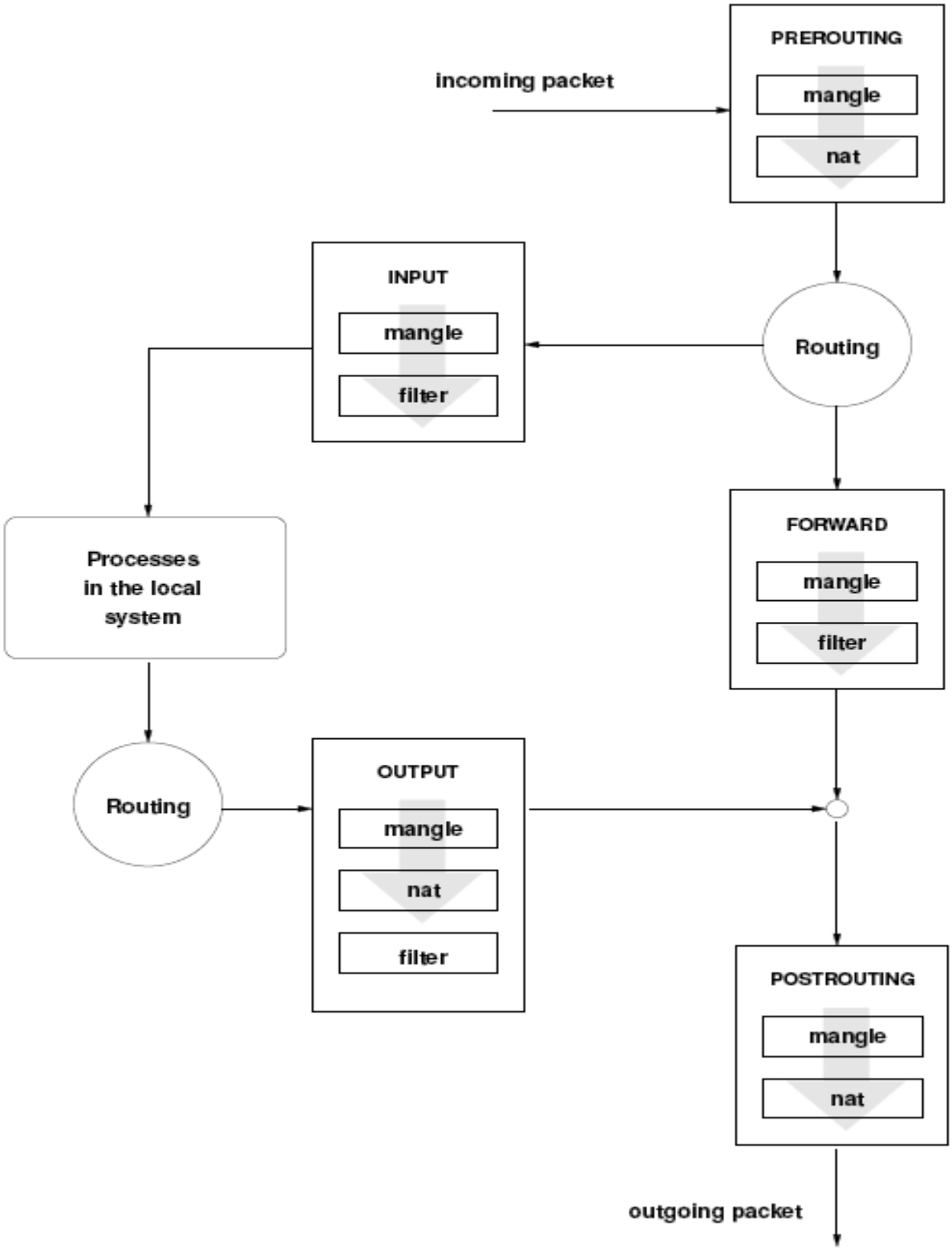
- Filter - в этой таблице содержатся наборы правил для выполнения фильтрации пакетов. Пакеты могут пропускаться далее, либо отвергаться (действия **ACCEPT** и **DROP** соответственно). Можно отфильтровывать пакеты и в других таблицах, но эта таблица существует именно для нужд фильтрации.
- Nat - эта таблица используется для выполнения преобразований сетевых адресов *NAT* (Network Address Translation). Первый пакет из потока проходит через цепочки этой таблицы, трансляция адресов или маскировка применяются ко всем последующим пакетам в потоке автоматически. Для этой таблицы характерны действия: Действие **DNAT** (Destination Network Address Translation) производит преобразование адресов назначения в заголовках пакетов. Другими словами, этим действием производится перенаправление пакетов на другие адреса, отличные от указанных в заголовках пакетов. **SNAT** (Source Network Address Translation) используется для изменения исходных адресов пакетов. С помощью этого действия можно скрыть структуру локальной сети, а заодно и разделить единственный внешний IP адрес между компьютерами локальной сети для выхода в Интернет. В этом случае брандмауэр, с помощью **SNAT**, автоматически производит прямое и обратное преобразование адресов, тем самым давая возможность выполнять подключение к серверам в Интернете с компьютеров в локальной сети. Маскировка (**MASQUERADE**) применяется в тех же целях, что и **SNAT**, но в отличие от последней,

MASQUERADE дает более сильную нагрузку на систему. Происходит это потому, что каждый раз, когда требуется выполнение этого действия - производится запрос IP адреса для указанного в действии сетевого интерфейса, в то время как для **SNAT** IP адрес указывается непосредственно. Однако, благодаря такому отличию, **MASQUERADE** может работать в случаях с динамическим IP адресом

- Mangle - эта таблица предназначена для внесения изменений в заголовки пакетов. В этой таблице допускается выполнять только следующие действия: **TOS**, **TTL** и **MARK**. Действие **TOS** выполняет установку битов поля *Type of Service* в пакете. Это поле используется для назначения сетевой политики обслуживания пакета, т.е. задает желаемый вариант маршрутизации. Однако, следует заметить, что данное свойство в действительности используется на незначительном количестве маршрутизаторов в Интернете. Другими словами, не следует изменять состояние этого поля для пакетов, уходящих в Интернет, потому что на роутерах, которые таки обслуживают это поле, может быть принято неправильное решение при выборе маршрута. Действие **TTL** используется для установки значения поля *TTL* (Time To Live) пакета. Есть одно неплохое применение этому действию. Мы можем присваивать определенное значение этому полю, чтобы скрыть наш брандмауэр от чересчур любопытных провайдеров (Internet Service Providers). Дело в том, что отдельные провайдеры очень не любят когда одно подключение разделяется несколькими компьютерами. и тогда они начинают проверять значение *TTL* приходящих пакетов и используют его как один из критериев определения того, один компьютер "сидит" на подключении или несколько. Дей-

ствие **MARK** устанавливает специальную метку на пакет, которая затем может быть проверена другими правилами в iptables или другими программами, например **iproute2**. С помощью "меток" можно управлять маршрутизацией пакетов, ограничивать трафик и т.п.

В пределах таблиц правила собраны в группы - цепочки. В iptables предопределены стандартные цепочки с именами: **PREROUTING**, **FORWARD**, **POSTROUTING**, **INPUT**, **OUTPUT**. Можно вводить новые цепочки с другими именами.


PREROUTING
incoming packet
mangle
nat
INPUT
mangle
filter
Routing
Processes
in the local
system
FORWARD
mangle
filter
Routing
OUTPUT
mangle
nat
filter
POSTROUTING
mangle
nat
outgoing packet

Порядок прохождения пакетов через цепочки правил показан на рисунке.

Каждое правило iptables - это строка, содержащая в себе критерии отбора пакетов и действие, которое необходимо выполнить в случае выполнения критерия. В общем виде правила записываются примерно так:

```
iptables [-t table] command [match] [target/jump]
```

Если в правило не включается спецификатор [-t *table*], то по умолчанию предполагается использование таблицы Filter. Раздел match задает критерии проверки, по которым определяется подпадает ли пакет под действие этого правила или нет. Здесь мы можем указать самые разные критерии -- IP-адрес источника пакета или сети, IP-адрес места назначения,порт, протокол, сетевой интерфейс и т.д. Значение target указывает, какое действие должно быть выполнено при условии выполнения критериев в правиле. Здесь можно заставить ядро передать пакет в другую цепочку правил, "сбросить" пакет и забыть про него, выдать на источник сообщение об ошибке и т.п.

Команды (список доступных команд можно просмотреть с помощью команды **iptables -h**, возможно использование дополнительных ключей):

-A, **--append**	Добавляет новое правило в конец заданной цепочки.
пример:	**iptables -A INPUT ...**
-D, **--delete**	Удаляет правило из цепочки.
пример:	**iptables -D INPUT --dport 80 -j DROP** **iptables -D INPUT 1**

-R, **--replace**	Эта команда заменяет одно правило другим (используется во время отладки новых правил).
пример:	**iptables -R INPUT 1 -s 192.168.0.1 -j DROP**
-I, **--insert**	Вставляет новое правило в цепочку (число задает номер для вставляемого правила).
пример:	**iptables -I INPUT 1 --dport 80 -j ACCEPT**
-L, **--list**	Выводит список правил в заданной цепочке.
пример:	**iptables -L INPUT**

-F, --flush	Удаляет все правила из заданной цепочки. Если имя цепочки не указывается, то удаляются все правила, во всех цепочках.
пример:	**iptables -F INPUT**
-Z, --zero	Обнуление всех счетчиков в заданной цепочке. Если имя цепочки не указывается, то подразумеваются все цепочки.
пример:	**iptables -Z INPUT**
-N, --new-chain	Создаёт новую цепочку с заданным именем в заданной таблице
пример:	**iptables -N name**
-X, --delete-chain	Удаляет заданную цепочку из заданной таблицы.
пример:	**iptables -X name**
-P, --policy	Задает политику по-умолчанию для заданной цепочки. Политика по-умолчанию определяет действие, применяемое к пакетам не попавшим под действие ни одного из правил в цепочке. В качестве политики по умолчанию допускается использовать **DROP** и **ACCEPT**.
пример:	**iptables -P INPUT DROP**
-E, --rename-chain	Выполняет переименование пользовательской цепочки.
пример:	**iptables -E old_name new_name**

Критерии:

Общие критерии (общие критерии употребляются в любых правилах, они не зависят от типа протокола):

-p, --protocol	Критерий используется для указания типа протокола. Протоколы - *TCP*, *UDP* и *ICMP*
пример:	**iptables -A INPUT -p tcp**
-s, --src, --source **-d, --dst, --destination**	IP-адрес источника (приёмника) пакета. Можно указать адрес в виде address/mask: *192.168.0.0/255.255.255.0* или *192.168.0.0/24*
пример:	**iptables -A INPUT -s 192.168.1.1**
-i, --in-interface **-o, --out-interface**	Интерфейс, с которого был получен (отправлен) пакет.
пример:	**iptables -A INPUT -i eth0**
-f, --fragment	Правило распространяется на все фрагменты фрагментированного пакета, кроме первого.
пример:	**iptables -A INPUT -f**

Неявные критерии (становятся доступны при указании общего критерия). Используются TCP, UDP, ICMP и соответствующие модули ядра подгружаются автоматически. Эти критерии позволяют указывать номера портов, флаги и другие поля из заголовков пакетов.

Явные критерии. Перед использованием этих критериев при помощью ключа **-m** должны быть загружены соответствующие модули ядра Linux. Наиболее интересные из них **limit** и **state**.

limit — позволяет установить предельное число пакетов в единицу времени, которое способно пропустить правило.

Пример:

iptables -A INPUT -m limit - -limit 5/second --limit-burst 10

Ключ --limit-burst - это максимальное значение счетчика пакетов, при котором срабатывает ограничение.

Ключ --limit - это скорость, с которой счетчик burst limit "откручивается назад".

state - критерий позволяет нам получать информацию о признаке состояния соединения, возможные значения:

- **INVALID** подразумевает, что пакет связан с неизвестным потоком или соединением и, возможно содержит ошибку в данных или в заголовке.
- **ESTABLISHED** указывает на то, что пакет принадлежит уже установленному соединению через которое пакеты идут в обеих направлениях.
- **NEW** подразумевает, что пакет открывает новое соединение или пакет принадлежит однонаправленному потоку.
- **RELATED** указывает на то что пакет принадлежит уже существующему соединению, но при этом он открывает новое соединение

Пример:

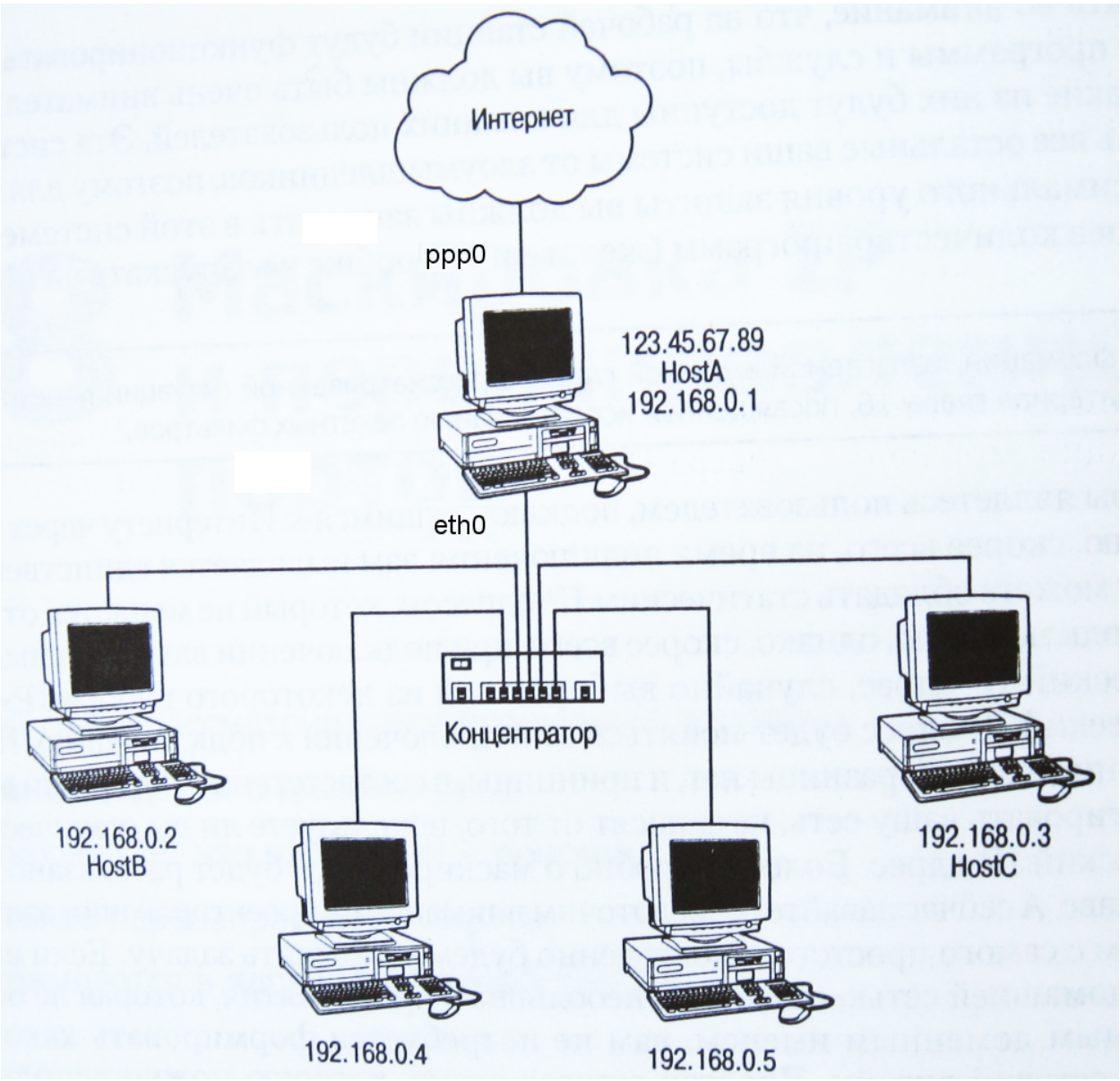
iptables -A INPUT -m state --state RELATED,ESTABLISHED

Действия и **переходы** сообщают правилу, что необходимо выполнить, если пакет соответствует заданному критерию. Чаще всего используются:

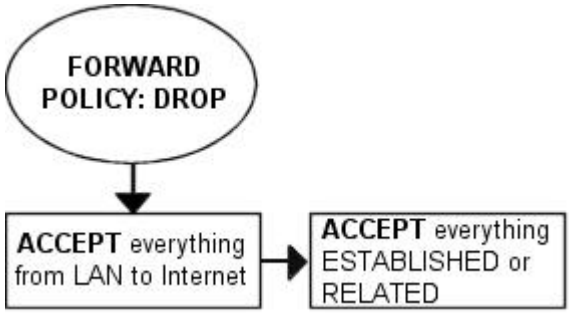
- **-j ACCEPT** - пакет прекращает движение по цепочке и считается ПРИНЯТЫМ
- **-j DROP** - действие просто "сбрасывает" пакет и iptables "забывает" о его существовании.
- **-j LOG** - действие, которое служит для журналирования отдельных пакетов и событий (полезно на период отладки правил вместо действия **DROP** использовать действие **LOG**, чтобы до конца убедиться, что ваш брандмауэр работает безупречно).
- **-j new_chain** - производит переход во вложенную цепочку с именем new_chain.
- **-j RETURN** - прекращает движение пакета по текущей цепочке правил и производит возврат в вызывающую цепочку, если текущая цепочка была вложенной, или, если текущая цепочка лежит на самом верхнем уровне (например *INPUT*), то к пакету будет применена политика по-умолчанию.

Здесь мы не рассматриваем правила и действия для задач трансляции адресов NAT.

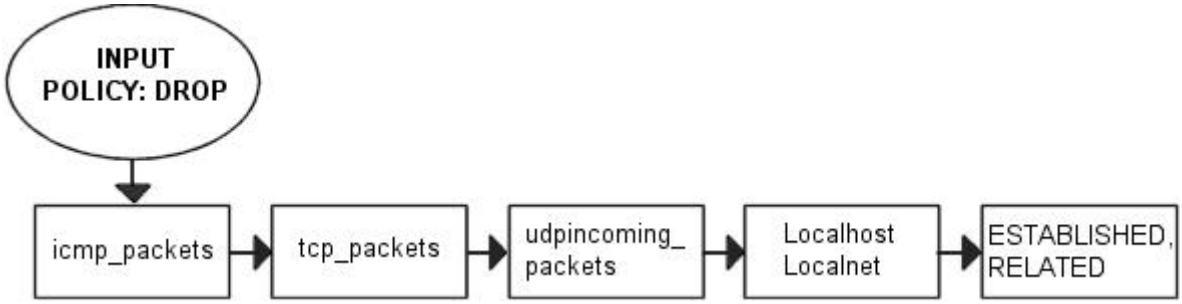
Пример:



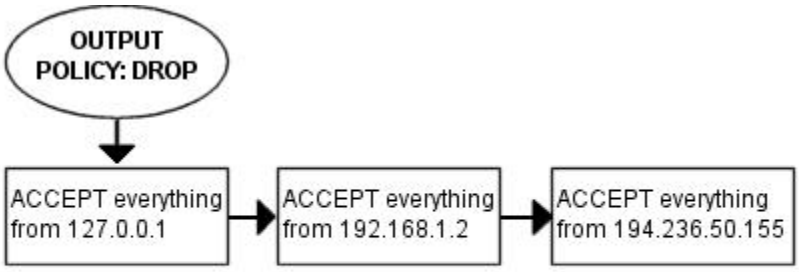
Размещение правил по цепочкам:



Выполнение некоторой фильтрации в цепочке *FORWARD*. Если мы полностью доверяем нашей локальной сети, пропуская весь трафик в Интернет, то это еще не означает доверия к Интернет и, следовательно необходимо вводить ограничения на доступ к нашим компьютерам извне. В нашем случае мы допускаем прохождение пакетов в нашу сеть только в случае уже установленного соединения, либо в случае открытия нового соединения, но в рамках уже существующего (ESTABLISHED и RELATED).



Для брандмауэра необходимо до минимума свести сервисы, работающие с Интернет. Следовательно мы допускаем только *HTTP*, *FTP*, *SSH* доступ к брандмауэру. Все эти протоколы мы будем считать допустимыми в цепочке *INPUT*, соответственно нам необходимо разрешить "ответный" трафик в цепочке *OUTPUT*. Поскольку мы предполагаем доверительные взаимоотношения с локальной сетью, то мы добавляем правила для диапазона адресов локальной сети, а заодно и для локального сетевого интерфейса и локального IP адреса (127.0.0.1). Как уже упоминалось выше, существует ряд диапазонов адресов, выделенных специально для локальных сетей, эти адреса считаются в Интернет ошибочными и как правило не обслуживаются. Поэтому и мы запретим любой трафик из Интернет с исходящим адресом, принадлежащим диапазонам локальных сетей.



Если брандмауэр работает одновременно и как рабочая станция мы устанавливаем правила, позволяющие прохождение пакетов только с нашим ад-

ресом в локальной сети, с нашим локальным адресом (127.0.0.1) и с нашим адресом в Интернет. С этих адресов пакеты пропускаются цепочкой *OUTPUT*, все остальные (скорее всего сфальсифицированные) отсекаются политикой по-умолчанию *DROP*.

Здесь приведён пример набора правил пакетного фильтра. Этот фильтр разрешает только исходящие TCP-соединения и пропускает TCP-пакеты только установленных соединений. Правила записаны в виде исполнимого скриптового файла для командного интерпретатора bash операционной системы Linux.

```
#!/bin/bash

iptables -F
iptables -t nat -F

iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT ACCEPT

iptables -N bad_tcp_packs

iptables -A bad_tcp_packs -p tcp ! --syn -m state
--state NEW -j LOG --log-prefix "New not syn:"
iptables -A bad_tcp_packs -p tcp ! --syn -m state
--state NEW -j DROP
iptables -A bad_tcp_packs -p tcp -m state --state
INVALID -j LOG --log-prefix "Invalid TCP:"
```

```
iptables -A bad_tcp_packs -p tcp -m state --state
INVALID -j DROP


iptables -A INPUT -p tcp -j bad_tcp_packs
iptables -A INPUT -p ALL -i eth0 -j ACCEPT
iptables -A INPUT -p ALL -i lo -j ACCEPT
iptables -A INPUT -p ALL -m state --state
ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -p ICMP -s 0/0 --icmp-type echo-
request -j ACCEPT
iptables -A INPUT -p tcp --syn --dport auth -j DROP
iptables -A INPUT -m limit --limit 3/m -j LOG --log-
prefix "IPT INPUT packet died:"


iptables -A FORWARD -p tcp -j bad_tcp_packs
iptables -A FORWARD -p ALL -i eth0 -j ACCEPT
iptables -A FORWARD -p ALL -m state --state
ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -o ppp0 -p udp --sport netbios-ns
-j DROP
iptables -A FORWARD -m limit --limit 3/m -j LOG --log-
prefix "IPT FORWARD packet died:"


iptables -A OUTPUT -o ppp0 -p udp --sport netbios-ns
-j DROP


iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE
```

Литература

1. У. Ричард Стивенсон «Протоколы TCP/IP. Практическое руководство», Санкт-Петербург, «Невский Диалект», 2003.
2. D. E. Denning. An intrusion detection model. IEEE Transactions on Software Engineering, SE-13, 1987, 222-232.
3. С. С. Корт «Теоретические основы защиты информации», Москва, «Гелиос АРВ», 2004.
4. С. Норткатт, Д. Новак, Д. Маклахлен «Обнаружение вторжений в сеть. Настольная книга специалиста по системному анализу», Москва, «Лори», 2001.