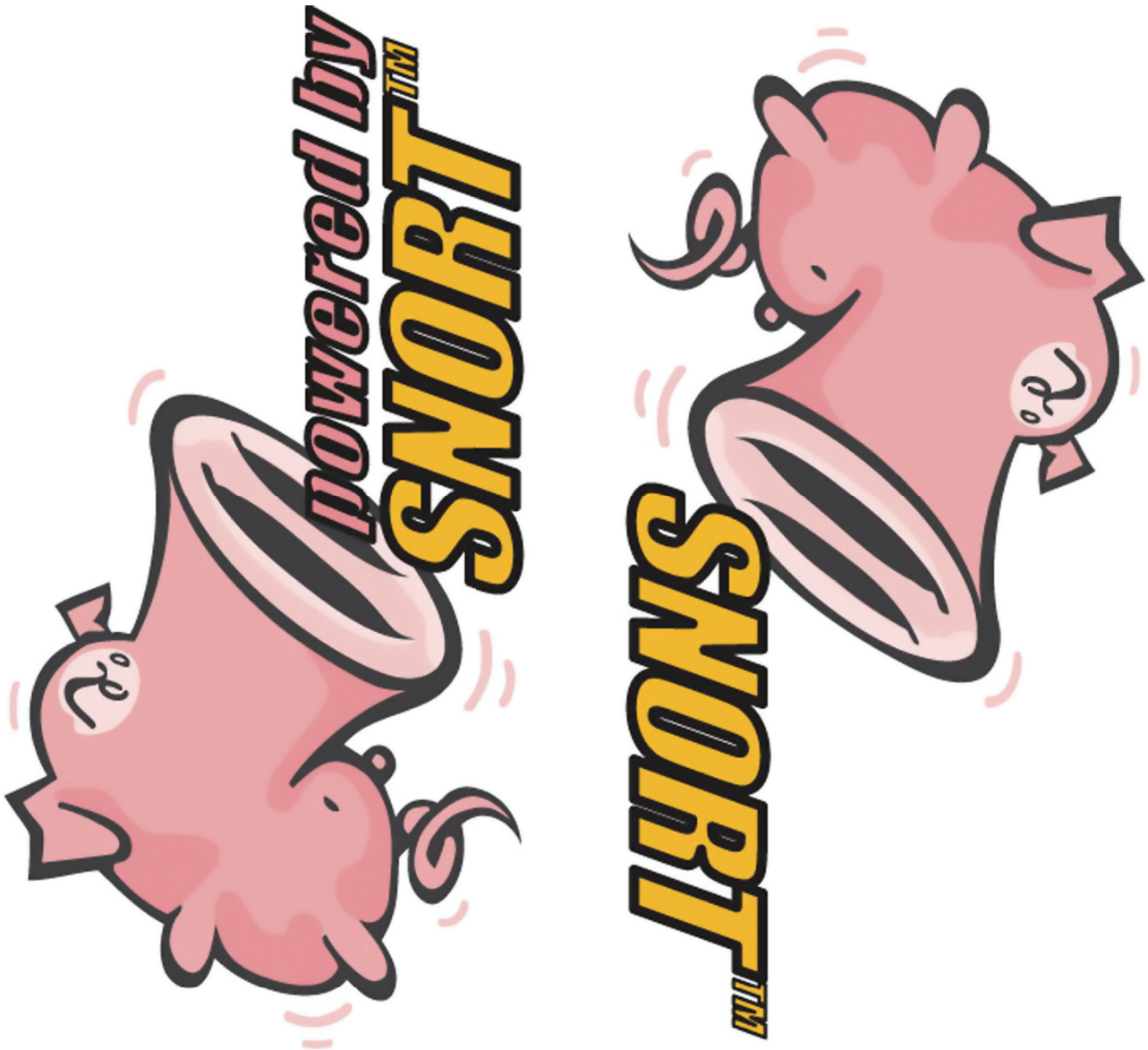
ОБНАРУЖЕНИЕ ТЕЛЕКОММУНИКАЦИОННЫХ АТАК: ТЕОРИЯ И ПРАКТИКА, SNORT



ПАВЕЛ ЗАКЛЯКОВ

По данным CERT[1], число инцидентов в сети растёт не по дням, а по часам – если грубо поделить 100 тысяч на 365 дней и на 24 часа, то получится примерно около 11 регистрируемых инцидентов в час. Ситуацию, сложившуюся в последнее время с червём Blaster, следует убрать из рассмотрения как исключение.

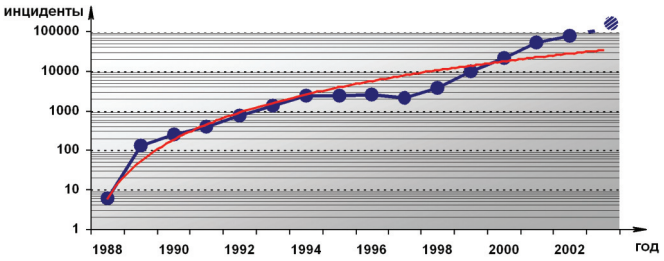


График 1. Рост числа инцидентов по годам (ось инцидентов в логарифмическом масштабе, красная линия показывает тенденцию роста)

Мы видим, что число инцидентов растёт, а в то же время последние номера журнала не изобилуют статьями по вопросам сетевой безопасности, а тем более обнаружению телекоммуникационных атак. На мой взгляд, есть ряд удачных публикаций [2-4]; менее удачных в силу сложности непосредственной применимости [5-8] и неудачных, но все они далеки от обнаружения атак. Даже широко продаваемая литература [9-14] далека от жизни, и за красивыми названиями порой ничего не стоит, кроме полного отставания в вопросах практической реализации. Хотелось бы несколько заполнить этот вакуум полезной информацией, в частности информацией про систему обнаружения атак IDS Snort [15].

Так как теория не стоит на месте и все появляющиеся упоминания скорее неполные, чем неправильные, то хотелось бы закрыть этот вакуум, подведя читателя непосредственно подкованным к вопросу использования IDS Snort.

Замечания, обсуждение и критика вопросов обнаружения телекоммуникационных атак приветствуются.

Способы классификации атак с позиции построения систем их обнаружения

Существуют различные типы классификации атак. Например, деление на активные и пассивные, внешние и внутренние, умышленные и неумышленные и др. Классы, в свою очередь, могут делиться на подклассы, подвиды, часто пересекающиеся, в результате можно получить следующую классификацию:

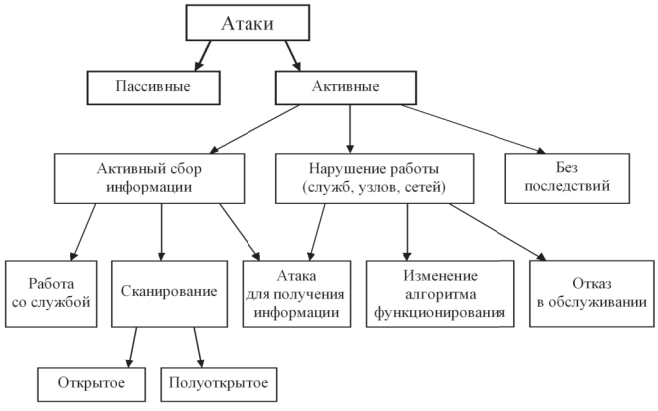


Рисунок 1. Классификация атак

Вышеописанные типы классификации хороши с точки зрения детализации по тем или иным критериям, но часто они выделяют отдельные классы, имеющие минимальное значение с точки зрения систем обнаружения атак (СОА). Поэтому наиболее перспективной представляется уже описанная в литературе следующая классификация, разделяющая атаки на классы, в которых их будет легче обнаружить и классифицировать:

1. Удалённое проникновение (remote penetration). Атаки, которые позволяют реализовать удалённое управление компьютером через сеть. Примерами программ, реализующих такое управление, являются, NetBus или Back Orifice, реверсивный сеанс telnet [18, стр. 314].
2. Локальное проникновение (local penetration). Атака, приводящая к получению несанкционированного доступа к узлу, на котором она запущена, либо повышению прав пользователя. Примером такой программы является GetAdmin или реализация эксплоита для уязвимости в ядрах Linux через ptrace.
3. Удалённый отказ в обслуживании (remote DoS (denial of service)). Атаки, которые позволяют нарушить функционирование системы или перезагрузить компьютер удалённо. Примерами такой атаки являются teardrop, trinoo, fapi.
4. Локальный отказ в обслуживании (local DoS). Атаки, позволяющие нарушить функционирование системы или перезагрузить компьютер, на котором они реализуются. В качестве примера такой атаки можно привести враждебный апплет, который загружает центральный процессор бесконечным циклом, что приводит к невозможности обработки запросов других приложений. Также, в случае неправильной настройки, это может быть ветвящееся приложение с кучей потомков, занимающих все свободные идентификаторы PID.
5. Сетевые сканеры (network scanners). Программы, которые анализируют топологию сети и обнаруживают сервисы, доступные для атаки. Примером такой программы может служить nmap.
6. Сканеры уязвимостей (vulnerability scanners). Программы, осуществляющие поиск уязвимостей на узлах сети и которые могут быть использованы для реализации атак. Примеры: система SATAN или Shadow Security Scanner, XSpider, LanGuard, XFocus-X-Scan и др.
7. Взломщики паролей (password crackers). Программы, которые подбирают пароли пользователей. Примером взломщика паролей может служить L0pht Crack для Windows или Crack для *nix.
8. Анализаторы протоколов и снифферы (sniffers). Программы, которые «прослушивают» сетевой трафик. При помощи этих программ можно автоматически искать такую информацию, как идентификаторы и пароли пользователей, информацию о кредитных картах и так далее. Фактически анализатор протокола – это сниффер плюс некоторая часть, осуществляющая фильтрацию и разбор перехваченных пакетов по некоторым правилам. Часто обе задачи выполняет один и тот же продукт. Примерами таких программ могут служить tcpdump, ethereal, Microsoft Network Monitor, NetXRay компании Network Associates, Lan Explorer.

Принадлежность реальных атак к тому или иному классу не всегда однозначна, так как не всегда можно чётко определить конечную цель атаки, которой можно добиться разными путями. Например, в случае невозможности осуществления напрямую DoS-атаки может быть выполнена атака на удалённое проникновение, после чего могут быть остановлены те или иные службы.

С точки зрения COA не имеет смысла классифицировать атаки по основным видам угроз [13, стр 77]: нарушение конфиденциальности, целостности, доступности, так как эти угрозы достаточно общие и реализуются по-разному, соответственно их обнаружение может быть осуществлено не единственным способом.

Каждая фирма предпочитает классифицировать атаки по-своему, например, компания Internet Security Systems, Inc. [19] ещё больше сократила число возможных категорий, доведя их до пяти:

- сбор информации (Information gathering);
- попытки несанкционированного доступа (Unauthorized access attempts);
- отказ в обслуживании (Denial of service);
- подозрительная активность (Suspicious activity);
- системные атаки (System attack).

Первые четыре категории относятся к удалённым атакам, а последняя – скорее к локальным, так как удалённое совершение системных атак затруднительно. Также в эту классификацию не попали некоторые атаки, например, «ложный DNS-сервер» [34, стр. 75-84], «подмена ARP-сервера» [34, стр. 84-99] и др.

Современные атаки настолько разнообразны, что единой однозначно правильной классификации, учитывающей все нюансы, они не поддаются.

Наиболее правильным подходом является классификация с точки зрения COA, так как в этом случае более чётко можно отнести те или иные действия к тому или иному классу атак, указав реальные, а не гипотетические атаки, и определить способы противодействия и обнаружения.

Давайте рассмотрим различные классы атак вместе со средствами противодействия. С позиции COA изначально атаки следует разделить на два больших класса: активные и пассивные, как это показано на рис. 1.

Пассивные атаки – это различный сбор информации о какой-либо службе, узле или сети узлов без какого-либо воздействия на их работу. Например, прослушивание и перехват трафика. Также пассивной атакой считается анализ открытых источников, однако данный подкласс мы рассматривать не будем, так как это не прямая, а косвенная атака. Более правильным было бы введение модели нарушителя, из которой сразу стало бы ясно, что в нашем случае считать атакой, а что нет, но чтобы не перегружать статью, опустим этот формальный момент.

Пассивные атаки являются наиболее сложными с точки зрения их обнаружения, так как грамотно организованная атака никак себя не обнаруживает. Данным классом не стоит пренебрегать, так как основная часть трафика в сети передаётся в открытом виде и эта атака может служить составной частью более крупномасштабной атаки. Для её реализации чаще всего сетевой адаптер ethernet переводится в режим promiscuous mode. Так как среда ethernet общая для всех, кроме случая точка-точка, сетевой адаптер начинает принимать (прослушивать) весь проходящий трафик. Поступающие так средства называются снифферами. Обычно снифферы запускаются на машинах, реально работающих в сети, то есть имеющих легитимные сетевые адреса.

Если сниффер используется нарушителем, то это считается атакой. Если сниффер санкционированно используется администратором сети, то это мощное средство анализа работы сети. Также сниффер является составной частью любой сеть-ориентированной COA (подробнее о сеть-ориентированных COA см. далее).

Обнаружение снифферов чаще всего основывается на ошибках их работы либо косвенными методами. При неправильной настройке хосты со снифферами могут выдавать ответы на пакеты, адресованные им, обнаруживая себя, в то время как при обычном режиме работы они на эти пакеты отвечать не стали бы. (Например, обнаружение большинства Linux-машин со снифферами несколько проще: в режиме прослушивания они отвечают на все пакеты, адресованные их IP-адресу, вне зависимости, их ли MAC-адрес в пакете или нет, из-за особенностей реализации и используемых настроек по умолчанию. В обычном же режиме работы все пакеты с чужими MAC-адресами отбрасываются не будучи обработанными.)

Другой трудноскрываемой чертой снифферов, а следовательно, выдающей их, является статистическая зависимость. Любому снифферу необходимо обрабатывать трафик либо сохранять его куда-то на диск. В случае недостаточной производительности неизбежны задержки. При этом замедление реакции на обычные пакеты может коррелировать с повышением трафика в сети у узлов, находящихся в режиме прослушивания. Так как снифферы постоянно совершенствуются, то на подобных ошибках можно отловить только уязвимые снифферы или устанавливаемые удалённо (в процессе атак, когда многие мелочи невозможно не только предусмотреть, но и реализовать). Существует несколько способов борьбы со снифферами.

Первым, не самым эффективным способом борьбы является их обнаружение с помощью ***антиснифферов***. Среди таких программных средств, ловящих снифферы в сети, можно назвать следующие: AntiSniff, CPM (Check Promiscuous Mode), neped, sentinel. Со своей задачей данные программные продукты справляются очень неплохо. Однако хорошо настроенный сниффер обнаружить ими невозможно. Поиск необнаруживаемых средств возможен только косвенными способами, например, путём создания ловушек [6,20] и генерации ложного трафика к ним, содержащего ценную информацию, скажем, пароли в открытом виде при обращении к ложному почтовому POP3-серверу. Хост, воспользовавшийся ложной информацией, будет значительным источником информации для успешного поиска сниффера. Однако и в этом случае атакующего могут и не поймать, если никто не воспользуется ловушкой. Ярким примером из жизни может служить история с городом Ковентри во время Второй мировой войны, когда был разбомблён целый город с людьми в цену сокрытия факта получения информации.

Несомненно, что бороться со снифферами лучше предупреждающими способами, сводя эффективность их использования до нуля.

Например, с помощью сильных средств ***аутентификации***. Под «сильными» мы понимаем такие методы аутентификации, которые трудно обойти. Например, использование одноразовых паролей (OTP – One-Time Passwords). Существует несколько способов реализации одноразовых паролей, в том числе и с использованием односторонних функций. Если нарушитель узнаёт одноразовый пароль с помощью сниффера, то эта информация будет бесполезной, потому что в этот момент пароль уже будет использован и выведен из употребления. Заметим, что этот способ борьбы со сниффингом эффективен только для борьбы с перехватом паролей. Снифферы, перехватывающие другую информацию (например, сообщения электронной почты), не теряют своей эффективности.

Ещё одним способом борьбы со сниффингом пакетов в вашей сетевой среде является ***локализация трафика*** путём использования коммутирующей инфраструктуры. Если, к примеру, во всей организации используется коммутируемый ethernet, хакеры могут получить доступ только к трафику, поступающему на тот порт, к которому они подключены. Коммутируемая инфраструктура не ликвидирует угрозу сниффинга, но заметно снижает её остроту. На практике данный класс атак имеет очень узкое применение, ограничивающееся в случае локализации трафика одним сегментом. Сегодняшнее значительное падение цен на коммутаторы (switch-и,$25-55 за 5-16-портовые) позволяет полностью отказаться от концентраторов (hub), чем значительно снизить актуальность применения данной атаки на практике.

В случае использования физической среды, отличной от ethernet, реализация программными средствами прослушивания сильно усложняется, вплоть до невозможности. Физическое снятие информации нами вообще не рассматривается.

В случае прохождения трафика через ненадёжные узлы также возможна утечка информации, попадающая под данный класс атак. Реализация в этом случае не требует рассмотрения. Бороться следует либо шифрованием трафика, либо, если позволяет маршрутизация, заданием маршрута через надёжные узлы.

Использование средств шифрования трафика (***криптографии***) не предотвращает возможности перехвата сообщений и не распознаёт работу снифферов, однако является самым действенным средством по борьбе со снифферами. Если канал связи является криптографически защищённым, то это значит, что нарушитель перехватывает зашифрованные сообщения. Использование стойких алгоритмов шифрования, например ГОСТ 28147-89, делает эту работу бесполезной. На сегодня существуют различные реализации данного способа с различной степенью надёжности. Например, криптография некоторых устройств фирмы Cisco на сетевом уровне базируется на протоколе IPSec, который представляет собой стандартный метод защищённой связи между устройствами с помощью протокола IP. К прочим криптографическим протоколам, которые можно использовать для борьбы со сниффингом, можно отнести SSH (Secure Shell) и SSL (Secure Socket Layer).

Активные атаки – это взаимодействие со службой, узлом или сетью узлов с целью нарушения их функционирования либо получения несанкционированным образом какой-либо информации. (На сегодняшний день нет точного определения термина «атака» [9, стр. 38], как и точного определения «активная атака».)

Исходя из этого определения можно заключить, что оно захватывает основные три типа угроз [13, стр. 77]: нарушение конфиденциальности, целостности, доступности, но, как было замечено выше, реально эти три типа сводятся к двум: активному сбору информации и нарушению работы. Для обоих типов имеются средства противодействия. Результатом нарушения работы может быть утечка информации, например, правила функционирования могут быть таковы, что информация недоступна. Также может быть отказ в обслуживании, замена одних параметров или данных другими. Формально в это определение попадёт и класс атак «без последствий», когда что-то было изменено, но ничего от этого не нарушилось, но от этого атаки не перестали быть атаками, как если бы вы перешли дорогу в отсутствие машин и свидетелей на красный свет, поэтому такие атаки надо также рассматривать. Многие определения упускают этот на первый взгляд безобидный класс атак. С точки зрения COA этот класс атак практически ничем не будет отличаться от других.

Под ***активным сбором информации*** следует понимать все способы, с помощью которых можно получить информацию: сканирование, работу со службами, получение информации в результате нарушения работы. (Существуют способы получения информации не техническими средствами или способами, которым нельзя противопоставить COA. Такие способы не рассматриваются, так как с технической точки зрения ни атаками, ни телекоммуникационными атаками они не являются. Например, обзор печатных источников открытой информации.)

Термин «телекоммуникационная атака» более правильный, но в силу его «громоздкости и длинности» в письменной и устной речи он практически всегда заменяется просто «атакой», все понимают смысл из контекста, хотя формально, например, удар топором по работающему серверу с последующим его выходом из строя будет являться атакой на отказ в обслуживании. Везде и ниже мы будем подразумевать телекоммуникационные атаки, если это не оговорено особо.

Сканирование бывает двух видов: обычное (или открытое) и полуоткрытое (закрытое).

Открытое сканирование – это когда производится попытка соединиться обычным образом с одной или несколькими службами. Так как службы привязаны к портам, то попытка соединения называется «опробованием порта» или просто опробованием. Чаще всего это делается с целью получения информации прямым или косвенным образом. При этом работа узлов, связанных каким-то образом с работой данной службы, не нарушается.

Обычно сканирование/опробование идёт сериями, как вариант, растянутое по времени и случайным образом.

После соединения со службой и получения от неё приветственных заголовков соединение закрывается.

Сканирование осуществляется программами, называемыми «сканерами». Обычно под сканером понимают средство защиты, выявляющее уязвимые места в системе, которое помимо различных сканирований может работать со службами. Если сканирование не санкционировано, то оно считается атакой.

Сканеры могут запускаться как локально, так и удалённо на других узлах. Различные сканеры ищут разные слабые места, но все они могут быть разделены на две категории: сканеры системы и сканеры сети. Сканеры сети подобны человеку, проверяющему, закрыл он дверь или нет, дёргая при этом за ручку. Подёргивание ручки не у своей двери аналогично несанкционированному сканированию.

С целью сбора информации открытому сканированию могут подвергаться не только отдельные службы и узлы, но и целые сети. Сканирование может проводиться как вручную, так и автоматическими средствами. С целью затруднения обнаружения в лог-файлах факта сканирования, оно может проводиться со случайным порядком сканирования портов и продолжительное время. Для борьбы со сканированием рекомендуется использовать программы, отслеживающие обращения с одного IP-адреса в короткое время к различным портам и блокирующие весь трафик для данного адреса. Также рекомендуется вместо сообщения об ошибке, в случае отсутствия сервиса/узла/сети, во внешнюю сеть вообще не высылать никакой информации [21]. Среди программ-сканеров можно перечислить следующие программные продукты, имеющие те или иные дополнительные возможности: nmap, ISS, SATAN, Connect, exsscan, getethers, IdentTCPscan, jakal, portscan, QueSo, sl0scan, spoofscan, strobe, xscan и др. Для защиты от атак, выполняемых сканерами, существуют различные программы, например: courtney, Icmplnfo, scan-detector, klaxon, PortSentry и многие другие, включая большинство СОА.

У обычного открытого сканирования есть один минус, а именно, оно может быть легко обнаружено без использования СОА путём ручного просмотра лог-файлов. На первых порах сканирования так и обнаруживали. В ответ на это был придуман способ закрытого или полуоткрытого сканирования, когда простым способом без использования дополнительных программ обнаружить факт сканирования невозможно.

Полуоткрытое сканирование – это формально незавершённое обычное открытое сканирование. Если рассмотреть обычную процедуру трёхстороннего «рукопожатия» при установке TCP-соединения, то отличие от обычного сканирования будет заключаться в отсутствии третьей – завершающей стадии. Если коротко, то согласно рекомендациям [22, 23] при установке соединения с узла, осуществляющего сканирование, посылается пакет с установленным флагом SYN (первая стадия), в ответ высылается подтверждение установления соединения пакетом с установленными флагами SYN и ACK со сканируемого узла (вторая стадия). Далее требуется подтверждение установления соединения со сканирующего узла пакетом с установленным флагом SYN (третья стадия). При отсутствии последней стадии происходит разрыв формально не установившегося до конца соединения по тайм-ауту, как следствие, запись в лог-файлах об успешном соединении не делается. Отсутствие записи о попытке установления соединения свидетельствуют об отсутствии сканирования.

Также в этот класс следует отнести сканирования пакетами без флагов, пакетами со всеми установленными флагами и различными их неправильными комбинациями, в литературе подобные сканирования носят свои названия, как NULL scan, FIN scan, X-MAS scan, «oddball packet» и др. Так как в RFC чётко не определено, как следует реагировать на неправильные пакеты, то различные операционные системы делают это по-разному, таким образом имеется возможность определить используемую операционную систему на хосте косвенным образом. Ситуация сравнима со следующей: вы подходите или подъезжаете к светофору, а там одновременно горят и красный и зелёный сигналы. Кто-то подождёт, кто-то пойдёт или поедет напролом, также по-разному реагируют и операционные системы на неправильные пакеты. Для борьбы с утечкой информации рекомендуется выбрасывать все неправильные пакеты, не отвечая на них сообщениями об ошибках. Так же можно использовать системы обнаружения сканирований или вторжений. С целью затруднения работы сканирующей стороне возможно использование липучих ловушек. Большинство сканирующих и обнаруживающих программных средств, перечисленных параграфом выше в разделе открытого сканирования, способны работать и с закрытыми сканированиями. Отделять эти программные средства в отдельную группу бессмысленно.

Само по себе сканирование без последующего продолжения атак не несёт никакой угрозы, однако возникшая и существующая по сей день практика безнаказанности заставляет обратить на данный вид атак больше внимания. Безнаказанность приводит к регулярному, практически непрерывному и случайному сканированию всего блока IP-адресов. Это отмечается и в различных последних отчётах по сетевой безопасности [24]. Факт сканирования уже не может, как ранее, использоваться в качестве высокодостоверного сигнала о начале какой-либо атаки. Случайные сканирования не столько представляют опасность для сканируемого узла, сколько «замусоривают» лог-файлы и затрудняют обнаружение атак, непосредственно направленных на данный узел. Для повышения значимости сканирования в качестве сигнального фактора обнаружения первой стадии атаки следует использовать обработку информации с нескольких распределённых узлов. Более подробно об этом будет рассказано в последующих разделах. Сейчас перейдём к следующей, логически вытекающей стадии перерастания сканирования в работу со службами.

Работа со службами – это взаимодействие с одной или несколькими службами, когда целью ставится получение информации, не нарушая работы. Обычно в случае использования автоматических средств нападения и сканирования данный класс атак следует сразу же после сканирования. Нередко разделить стадию сканирования от стадии работы со службами бывает невозможно. Однако у этих двух классов атак есть различие – разная достоверность получаемых результатов. Работа со службами может более точно определять тип и диапазон версий

операционной системы и запущенных сервисов, тем самым подавать более достоверную информацию, что повышает вероятность осуществления успешной атаки на последующих стадиях. Классическим примером борьбы на первой стадии с данным и предыдущим классом атак является замена заголовков, выдаваемых теми или иными сервисами после установления соединения.

Целью любой работы со службой обычно является получение информации для последующего нарушения работы служб.

Нарушение работы может быть направлено на отдельные службы, группы служб, узлы и сети узлов с целью получения информации, изменения алгоритма функционирования атакуемого объекта, отказа в обслуживании. Более ясно представить классификацию атак можно, посмотрев на рисунок 1.

Атака для получения информации. Данный класс является логическим продолжением предыдущих. После работы со службой следуют действия, вызывающие разовое изменение нормального алгоритма работы, в результате чего может произойти утечка информации. Цель атаки – только получение информации, при этом атакующий может восстановить работу узла и попытаться скрыть следы своего присутствия. Защита от данного класса атак ничем не отличается от мер, противодействующих сканированию и другим атакам. Дополнительно принятыми мерами может быть шифрование важной информации, начиная от swap-раздела и заканчивая шифрованием всего трафика. Не имеет смысла пытаться выделить средства, противодействующие только данному классу атак.

Атака с целью изменения алгоритма функционирования является либо продолжением предыдущего класса, либо отдельным классом. Целью атак данного класса, как видно из названия, является явное изменение алгоритма функционирования. Часто атаки попадают сразу под несколько классов, так как имеют несколько целей.

Два вышеописанных класса атак являются классическими, и чаще всего, когда употребляют слово «атака» без уточнений, то имеются в виду именно эти два класса. Системы обнаружения атак и/или вторжений изначально делают больший акцент на эти классы, так как ущерб от них, особенно от второй, сразу становится явным.

Формально изменение алгоритма функционирования может быть произведено таким образом, что узел или служба «зависнет» либо просто не будет никак реагировать на обращения к нему. Однако данный вид атак обычно выделяют отдельным классом, так как привести систему к состоянию отказа в обслуживании бывает гораздо более простой задачей, чем что-то изменить в алгоритмах функционирования, не нарушив работоспособности.

Отказ в обслуживании. (DoS, Denial of Service). Отдельный вид атак, когда целью ставится перевести узел, сеть или службу в такое состояние, когда она не могла бы обслуживать легальных пользователей, выдавая им сообщение о недоступности или вообще не отвечая. Добиться данного можно двумя путями. Первый – с помощью уязвимостей в атакуемом объекте. Несмотря на то, что цель атак подходит под определение данного класса, их принято классифицировать как «изменение алгоритма функционирования», см. выше. Второй – с помощью большого числа запросов. Любой узел, даже без уязвимостей, с верифицированным алгоритмом работы, можно перевести в состояние отказа в обслуживании. Исторически изначально была попытка борьбы с данным классом атак путём увеличения производительности узлов и скорости пропускных каналов к ним, но тут же последовал ответ в виде распределённых атак на отказ в обслуживании (DDoS, Distributed Denial of Service). Распределённая атака на отказ в обслуживании – это когда атака ведётся не с одного узла в сети, а с нескольких. Логично заметить, что путём увеличения числа атакующих узлов, даже при маленькой пропускной способности и производительности, на них можно «забить» любой высокоскоростной узел/канал. Набор узлов для проведения подобных атак обычно является следствием других атак. В результате уязвимостей на узлах они атакуются и на них размещается некоторый код, который работает по продуманному алгоритму.

С точки зрения выбираемой цели совершения той или иной атаки в данный класс возможно включение и некоторых видов вирусов-червей. Размещённый код может выжидать команды, а может и атаковать и «завербовывать» себе другие узлы, создавая распределённую сеть из заражённых узлов. Когда распределённая сеть из заражённых узлов начнёт атаку на отказ в обслуживании, то защищаться от данной атаки отдельно невозможно. Последствия можно как-то уменьшить, если у вышестоящего провайдера во время атаки ставить up-stream-фильтры. Борьба в данном случае ведётся уже вручную, а не автоматическими средствами, постепенным отключением каждого замусоривающего сеть узла через его провайдера.

Наиболее эффективно данный класс атак предупреждать, так как заблокировать данные атаки за короткое время на сегодняшний день затруднительно. Если бы в узлах не существовали уязвимости, то данный класс атак было бы очень сложно реализовать по причине трудности создания объединённой атакующей сети. С нескомпрометированных узлов атаковать можно, но велик риск быть пойманным и наказанным. Атакующие предпочитают не рисковать таким образом.

Сейчас этот класс в большинстве составляют именно распределённые атаки на отказ в обслуживании, так что слово «распределённые» часто опускается.

После того как мы познакомились с классификацией атак с точки зрения систем их обнаружения, давайте рассмотрим более подробно используемые средства защиты и обнаружения на отдельных узлах.

Обнаружение атак и защита от них на отдельных узлах распределённой системы

С существованием атак тесно связано их обнаружение. Если атаки нельзя было бы обнаруживать, то это было бы просто бедствием с точки зрения безопасности, наоборот, если бы все атаки обнаруживались, то нечего было бы исследовать и не от чего было бы защищаться.

Для защиты от атак на практике используются не только программные продукты, описанные выше, но и специализи-

рованные программно-аппаратные средства. Использование аппаратной компоненты с точки зрения теории практически не вносит ничего нового, кроме особенностей функционирования, и призвано лишь удешевлять существующие решения при требованиях большей производительности и безопасности. Где есть необходимость в использовании аппаратных средств, будет оговорено отдельно.

Многие классы атак являются включающими друг дуга, поэтому программные продукты, выполняющие различные функции предупреждения и защиты от атак, можно разделить на следующие категории, однозначно не совпадающие с приведённой выше классификацией атак. Эффективность обнаружения атак от этого не ухудшается:

1. ***Межсетевые экраны*** – средства, организующие фильтрацию пакетов на основе их заголовков и/или других критериев. Подробнее о межсетевых экранах можно прочитать в следующей литературе [21, 25, 26].
2. ***Антивирусные программы***, осуществляющие поиск вирусов и подозрений на вирусы в файлах или информационных потоках. Подробнее смотрите [27, 28].
3. ***Снифферы*** – программы, осуществляющие перехват всего проходящего трафика в сегменте для дальнейшего его анализа вручную или автоматическими средствами [14, стр. 111-127].
4. ***Средства обнаружения атак/вторжений*** – так же, как и снифферы, перехватывают весь или часть траффика и осуществляют поиск в нём подозрительных событий. Используются различные методы поиска, чаще всего сигнатурный метод. Иногда средства обнаружения вторжений дополнительно имеют свойства из других категорий [9].
5. ***Средства контроля целостности файловых систем*** осуществляют периодическую проверку файловых систем, на которых установлены операционные системы, которые могут быть скомпрометированы на факт изменения или удаления «неизменяемых» файлов, появления новых. Проверка чаще всего осуществляется с использованием средств криптографии с целью повышения надёжности. По результатам проверки возможны различные заранее запрограммированные действия [14, стр 100-109], [4].
6. ***Ловушки*** – осуществляющие имитацию работы той или иной службы/хоста/сети. Контролирующие и протоколирующие все обращения к ним. Являются развивающимся классом на сегодняшний день. Очень перспективны с точки зрения сбора доказательств злого умысла нападающего, не подвергая при этом реальные системы какой-либо опасности [6, 20].

Разберём все эти классы более подробно.

Межсетевые экраны

«МЭ представляет собой локальное (однокомпонентное) или функционально-распределённое средство (комплекс), реализующее контроль за информацией, поступающей в АС и/или выходящей из АС, и обеспечивает защиту АС посредством фильтрации информации, т.е. её анализа по совокупности критериев и принятия решения о её распространении в (из) АС» [25].

В отечественной литературе в последнее время наблюдается разнобой в терминологии, в качестве синонима термина МЭ часто используются слова: «брандмауэр» и «firewall», заимствованные из немецкого и английского языков, соответственно дословно обозначающие «огненная стена».

Работу МЭ можно разделить на несколько составляющих:

- Анализ и фильтрация пакетов. Пакеты могут быть различных протоколов.
- Блокирование пакетов протоколов или содержимого.
- Аутентификация пользователя (подключения) и шифрование сеанса. Одновременно в МЭ могут присутствовать любые составляющие из перечисленных в зависимости от требований, предъявляемых к МЭ. Если классифицировать МЭ по ЭМВОС (OSI/ISO) [29 стр. 19-25], [30 стр. 66-74] уровню, то имеется два основных типа МЭ:
  - МЭ сетевого уровня или фильтры пакетов;
  - шлюзы приложений.

На канальном уровне тоже можно установить МЭ, относить данный МЭ следует к первому классу. Синонимом пакета в данном случае может служить и дейтаграмма, и ячейка, несмотря на то, что это несколько разные понятия.

МЭ больше подходят для защиты, нежели для обнаружения атак, однако ведение логов и использование этих средств совместно с другими может расширять сферу их применения в области защиты.

Под ***анализом и фильтрацией пакетов*** обычно понимается соответствие заголовков или поля данных какому-либо критерию. В процессе совершения атак или ведения подготовки к ним обмен с потенциально атакуемым объектом ведётся посредством обмена пакетами. МЭ является узким местом, где можно отсеять ненужные пакеты. Таким образом, если знать адреса нарушителей, то можно запретить обмен любым трафиком с нарушителями. Так, возможно разрешение или запрещение использования каких-либо служб какими-то отдельными узлами. Например, если имеется внутренняя БД, например, на порту 1433, то можно запретить все входящие пакеты, имеющие порт назначения 1433, тем самым исключив возможность атаки на этот порт. Также фильтрация может осуществляться на основе критерия «направления установки соединения» изнутри наружу или снаружи внутрь по флагам в пакетах. Можно запретить все входящие соединения.

Фильтрация неправильных пакетов может предупредить различные атаки, направленные на переполнение буфера, определение операционной системы, сканирование портов. Фильтрация таких пакетов есть способ борьбы с полуоткрытым сканированием, описанным выше в разделе классификации атак.

Многие операционные системы имеют мощные встроенные МЭ. Обычно это пакетные фильтры с расширяемыми возможностями. Так, операционная система Linux имеет пакет iptables (ipchains или ipfw), позволяющий производить фильтрацию [7, 8]. ОС OpenBSD, FreeBSD и другие также имеют МЭ. Различные версии Windows (на базе NT) имеют также встроенные, но с меньшим набором функциональных возможностей МЭ. Малый набор фильтрую-

щих возможностей штатными средствами, особенно у семейства операционных систем Windows, компенсируется наличием большого числа коммерческих продуктов от третьих производителей, например AtGuard, ZoneAlarm и др. Для ОС с открытым кодом такие продукты по большей части бесполезны, так как они не могут фильтровать лучше, чем сама ОС, средствами ядра.

Имеется множество программно-аппаратных средств от различных производителей, реализующих функции фильтрации «в виде отдельного блока» независимо от ОС: Cisco Secure IDS, ISS RealSecure for Nokia, NFR Intrusion Detection Appliance, SecureCom, Citadel и многие другие. Некоторые из них даже сертифицированы бывшим ФАПСИ.

С точки зрения размещения МЭ может располагаться как на защищаемом узле, так и отдельно. Первый вариант обходится дешевле, но с большим риском для безопасности. Существует конечная вероятность того, что атака может вывести МЭ из строя. Отдельные узлы, особенно не имеющие IP-адресов (например, различные МЭ-мосты [26]) и администрируемые исключительно с консоли или доверенного интерфейса, практически имеют очень большую надёжность, определяемую только используемым ПО внутри этих средств, которое может случайно содержать ошибки. Так как извне к таким устройствам обратиться невозможно, то невозможно и как-то повлиять на их работу. Даже атака на отказ в обслуживании экранируемой сети при правильном расчёте не приведёт к выводу из строя или изменению алгоритма работы таких устройств. Пакеты, предназначенные для фильтрования, будут отфильтрованы в любом случае.

Ещё одним плюсом за использование отдельных МЭ можно назвать безразличность этих средств к вирусам, так как обработка пакетов напоминает чем-то гарвардскую архитектуру, где разделены потоки команд и данных. Единственным минусом таких средств является их взаимодействие с внешним миром. Чем больше возможностей для связи имеется, тем больше риск атаки на МЭ. Приходится выбирать между удобствами администрирования и безопасностью.

Многие вещи довольно сложно или неэффективно реализовать с помощью фильтров пакетов, например, если надо разрешить посещать какой-то сайт, но в то же время запретить посещение его какой-то части, то для выполнения данной задачи потребуется разборщик и анализатор пакетов, и пр. В конечном счёте получится что-то вроде прозрачного шлюза приложений. Поэтому лучше сразу заметить, что с подобной задачей легко справляются шлюзы приложений. По сложившейся практике никто не называет шлюзы межсетевыми экранами, хотя они являются их подтипом. Шлюзов приложений также много, сколько и приложений, их использующих. Наибольшей популярностью пользуются http-прокси-сервера, насколько они популярны, настолько же часто они подвергаются различным атакам. Среди открытых программных продуктов, распространяемых по лицензии GNU Public License, наибольшей популярностью пользуется пакет squid. Среди программных продуктов с закрытым кодом сложно перечислить однозначно наиболее часто используемые продукты.

Правильно настроенный прокси-сервер может защищать от многих атак. Во-первых, помимо защиты он может просто ускорять работу, кэшируя различные данные. Во-вторых, он может скрывать пользователя, будучи анонимизирующим прокси. Злоумышленнику придётся сначала сломать прокси-сервер, прежде чем он доберётся до того, что скрывается за ним. Довольно часто прокси-сервера устанавливаются на шлюзах. Также прокси-сервера могут бороться с неавторизованными пользователями, не пропуская пакеты от них. Так как прокси-сервер сам устанавливает соединение на уровне приложения, то появляются дополнительные возможности по изменению передаваемого содержимого. Перед передачей данных пользователю их можно проверять на вирус «на лету» и блокировать или изменять в зависимости от результатов проверок. В данном случае МЭ уже становится комбинированным средством, обладающим свойствами различных классов программ, начиная от фильтра пакетов и заканчивая антивирусными программами.

Иногда прокси-сервера ставят прямо перед веб-серверами. Получается подключение в разрыв. В данном случае доверенный прокси-сервер будет защитой от передачи нестандартных запросов недоверенному веб-серверу. Не секрет, что есть случаи, когда производители оставляют специально или забывают убрать, по ошибке, различные недокументированные функции из своих программ. Злоумышленники, узнав про такие возможности, могут ими воспользоваться. Именно по этой причине на первый взгляд может показаться, что программные продукты с открытым кодом содержат большее число уязвимостей. На самом деле реальное число уязвимостей оценить сложно, а в закрытом программном коде такие уязвимости скрыты от простого просмотра, и для их выявления требуются гораздо большие усилия. Использование прокси-сервера с верифицированным алгоритмом работы исключит всякие отклонения от предусмотренного режима работы.

Также из внешней сети будет виден прокси-сервер. Попытка определения операционной системы узла, скрывающегося за прокси, встретит на своём пути трудности. Фактически при правильной настройке возможно полностью исключить возможность определения ОС, функционирующих на узлах за прокси-сервером, а также топологию сети даже косвенными методами. К сожалению, данные способы использования прокси ввиду их небольшой популярности практически не описываются в литературе. В литературе большее внимание уделяется кэширующим способностям прокси-серверов.

Подключение в разрыв может быть как прозрачным (без изменения сетевых настроек), так и нет (когда требуется изменение настроек). Возможно прозрачное установление прокси в «разрыв» так, что даже легитимные пользователи, общающиеся с внешней сетью, его не заметят. В то же время будет иметься возможность полного контроля их соединений. Многие операционные системы, в частности ОС Linux, позволяют делать данные вещи штатными средствами (bridge, divert, squid) [26].

Использование межсетевых экранов – это всё равно, что мощная бронированная железная дверь, а разрешающие и запрещающие правила – это замок. По-хорошему, такую дверь сломать нельзя, однако никто не мешает попасть внутрь через окно или через дырку в стене, которую могли случайно оставить по ошибке рабочие. Или вам про-

сто изнутри эту дверь кто-то откроет. Поэтому комплексная защита межсетевыми экранами не ограничивается.

Следующим отдельным классом идут антивирусные средства.

Антивирусные программы

История вирусов очень богатая, а классификация вирусов очень широкая. Мы будем рассматривать только те вирусы, которые имеют какие-то сетевые компоненты и могут оказывать какое-то влияние на другие сетевые устройства через сеть. Если какой-то компьютер заражён boot-вирусом, то он никоим образом не влияет на работу независимого сетевого окружения, поэтому и не стоит подобные классы очень подробно рассматривать в рамках обнаружения телекоммуникационных атак.

Наибольший интерес представляют вирусы последнего поколения, сочетающие в себе много различных функций. Природа вирусов обычно такова, что они должны распространяться, заражая при этом максимально возможное число компьютеров. Если ранее обмен данными чаще происходил посредством обмена дискетами, то на сегодняшний день большую часть занимает обмен сетевым трафиком. Некоторые компьютеры и дисковода не имеют. Поэтому гораздо выгоднее стало писать вирусы, распространяющиеся через сетевые среды. Дополнительным фактором, стимулирующим написание сетевых вирусов, является относительно постоянное подключение заражаемых компьютеров к сети, даже пользователь, подключившийся по модему, в течение некоторого отрезка времени может быть рассмотрен как находящийся на постоянном подключении, и, следовательно, его ресурсы могут быть атакованы, заражены и послужить источником для дальнейшего цепного распространения.

Первым сетевым вирусом, положившим название целому классу вирусов-червей, был червь Морриса. Алгоритм работы вирусов-червей не сильно изменился за последние годы. Поэтому для противодействия вирусам необходимо учитывать особенности их появления. Для написания вируса необходимо совершить следующие шаги:

- Найти какую-то уязвимость в программном обеспечении.
- Написать программу, автоматически заражающую удалённый компьютер через эту уязвимость.
- Написать программу, осуществляющую автоматический поиск компьютеров для их поражения программой из второго пункта.
- Разработать средство взаимодействия заражённых компьютеров.

Противостоять данному виду вирусов можно либо путём блокирования уязвимостей, либо путём их устранения, либо путём их защиты дополнительными средствами. Можно попытаться противостоять взаимодействию частей вируса, однако существуют вирусы, в которых не осуществляется никакого взаимодействия и процесс заражения никем не координируется.

Направить основные усилия на устранение недостатков в существующем ПО не представляется возможным по причине того, что объёмы программного кода постоянно растут, программные продукты очень быстро устаревают, поэтому невыгодно производить дорогостоящие проверки и тестирования. Создание верифицированных продуктов выливается в круглую сумму. Работу продуктов с закрытым кодом проанализировать и проверить сложно, а зачастую это противоречит лицензии того или иного продукта, поэтому многие уязвимости так и не обнаруживаются. С открытым и бесплатным кодом дела обстоят проще, поиск уязвимостей ведётся по мере чьей-то заинтересованности в этом и большим числом народа. Поэтому уязвимости в ПО с открытым кодом обнаруживаются чаще.

Можно смело утверждать, что в любом ПО средних и больших размеров существуют ошибки. Коренным образом решить проблему их появления невозможно. Таким образом, системам защиты приходится защищаться косвенными образом. Если у вас на двери сломан замок и его нельзя заменить, то можно около двери поставить сторожа.

Обычно таким местом, где следует поставить антивирусного сторожа, являются почтовые сервера и межсетевые экраны. На первых осуществляется проверка всей электронной корреспонденции на наличие известных вирусов. На вторых может проверяться любой сетевой поток вне зависимости, куда он идёт или откуда поступает. Для поиска вирусов используется сигнатурный метод. Данный процесс фактически ничем не отличается от сигнатурного обнаружения атак, однако тут есть своя специфика существующей БД и проверяемого потока. Например, антивирусные программы могут исправлять поток данных, вылечивая заражённые объекты «на лету», не блокируя их, в то время как СОА в основной массе не имеют аналогичных возможностей.

Для поиска вирусов на отдельных локальных компьютерах используются подобные средства мониторинга, осуществляющие проверку всех получаемых и открываемых файлов. Использование сигнатурных баз данных требует постоянного их обновления, поэтому для создания надёжной защиты следует использовать данные средства совместно с другими. Например, довольно часто антивирусные программы могут осуществлять контроль целостности файлов. Либо являться компонентой какого-то более общего средства.

Среди программных продуктов можно назвать очень много антивирусных программ, но выделить наиболее эффективные среди них довольно сложно, а тем более попытаться как-то выбрать лучшее средство. Наличие большего числа сигнатур не является доминирующим аргументом в пользу того или иного антивирусного продукта, так как нужного вируса в БД сигнатур может не оказаться, несмотря на то, что там содержится наибольшее число сигнатур. Поэтому защита не может основываться на одних антивирусных средствах. Среди наиболее часто используемого антивирусного ПО следует назвать: AVP Касперского, Dr.Web, Norton Antivirus, OpenAV и др. Многие продукты имеют версии для различных операционных систем, начиная от Linux и заканчивая Novell Netware. Существуют антивирусы и с открытым программным кодом.

Снифферы

Снифферы, как и оружие, могут применяться как с целью нападения (см. про пассивные атаки выше), так и с целью защиты. Для защиты они помещаются рядом с защищае-

мым местом для перехвата всего проходящего трафика в данном сегменте. Явным образом снифферы защищать не могут, они защищают «косвенным» образом и используются как составная часть других средств. При этом при необходимости физически специально создаются условия для перехвата всего трафика. Например, сниффер может быть подключён к дополнительному концентратору или к специальному порту коммутатора, на который передаются копии данных со всех портов. После получения данных снифферов производится их анализ. Анализ может быть как местным, так и внешним. Данные могут помещаться в БД, которая будет анализироваться уже другими средствами. Захватывать данные удобнее всего в сетях с общей средой передачи, как в Ethernet. Среди программных продуктов, осуществляющих перехват всего трафика в сетях Ethernet, можно назвать tcpdump, windump, ethereal и др. Вести перехват данных можно и в сетях, отличных от Ethernet, реализация в этом случае может немного усложниться, и потребуется наличие дополнительной аппаратной компоненты. Суть при этом не меняется.

По данным перехвата, при умелом их прочтении можно очень многое сказать о сети, из которой они были перехвачены. По количеству передаваемых пакетов можно судить о загруженности. По адресам источников и назначения можно судить об используемых сервисах и серверах, если таковые имеются в сети. Также снифферы отслеживают попытки взлома тех или иных сервисов, а их БД являются доказательной базой совершения тех или иных действий. Эти БД также могут служить источником информации для пополнения сигнатурных БД, используемых другими средствами защиты.

Часто снифферы используются совместно с ловушками. Порой количество проходящих данных настолько велико, что не имеется физической возможности производить постоянную запись всего трафика хотя бы за последние сутки. Даже при наличии всего перехваченного трафика существует проблема его анализа и поиска в нём нужной информации. Администратор-человек бессилен. Для автоматизации процесса поиска в своё время были написаны программы, которые в дальнейшем приобрели ряд дополнительных свойств и стали называться системами обнаружения атак/вторжений, о которых пойдёт речь в следующем параграфе.

Системы обнаружения атак (СОА)

Данный класс средств защиты есть историческое развитие других классов. Поэтому СОА обладают различными характеристиками других классов, которые в совокупности могут предоставить дополнительную информацию. СОА собирают данные с различных источников. Основное – это перехват данных снифферами, однако, как было замечено выше, не всегда имеется возможность и требуется перехватывать все данные целиком. Поэтому СОА осуществляют анализ данных «на лету», не обращая внимания на менее значимые события. Обычно анализ проводится сигнатурным методом. В этот момент работа СОА практически ничем не отличается от работы антивирусных средств за исключением специфики БД.

Большинство злоумышленников изначально пытаются атаковать узлы уже известными атаками, так как вероятность существования уязвимостей для этих атак больше, но и вероятность присутствия данной атаки в сигнатурной БД тоже велика, поэтому СОА обнаруживают такие атаки. Так, любое сканирование распознаётся практически безошибочно. В качестве реакции на какое-то событие СОА может передавать управление любому заранее написанному скрипту, который может инициировать закрытие соединения с атакующим узлом либо изменять политику фильтрации пакетов. Из последнего следует, что СОА очень тесно используются совместно с межсетевыми экранами. Отсутствие в трафике данных, коррелирующих с теми или иными сигнатурами, не говорит об отсутствии нарушителей, поэтому в данных средствах используется сбор информации с различных мест. Сбор данных в сети осуществляется посредством сенсоров – небольших программ или приспособлений, расположенных вблизи прослушиваемых мест и выдающих различную информацию о состоянии прослушиваемого объекта. Объектом может быть как соединение, так и лог-файл работы той или иной программы. Средства, анализирующие лог-файлы, исторически не принято называть сенсорами. Фактически, сенсор – это маленькая копия СОА, отправляющая данные в некий общий центр – ядро анализа СОА. Всё вышеописанное может располагаться как на одном узле, и тогда сложно отделить одни функции от других, так и в различных местах сети.

СОА исторически делятся на два типа: сеть- и хост-ориентированные СОА. Хост-ориентированные СОА как раз и занимаются анализом различных файлов на хосте, в то время как сеть-ориентированные занимаются перехватом и анализом трафика в сети. Большее развитие в последнее время получило сеть-ориентированное направление. Отчасти это объясняется платформенной независимостью СОА от используемых ОС на компьютерах в сети.

Правильно настроенная СОА обнаруживает большой процент атак, при маленьком проценте ложных срабатываний. Понятие «СОА» часто трактуется очень широко и включает в себя множество различных компонентов, в результате чего довольно сложно определить границы. Например, это может быть обычная БД. Те же антивирусные средства также могут являться одним из компонентов СОА. Не следует исключать ошибки первого рода, которые уходят из внимания СОА. Среди программных продуктов можно перечислить: NIDS, BRO, Snort и др., о последнем продукте как раз пойдёт речь дальше в статье.

Средства контроля целостности файловых систем

СОА не могут гарантировать обнаружение всех атак, поэтому существует некоторый процент атак, не обнаруживаемый СОА. Пропущенную на первый взгляд атаку можно достоверно обнаружить другими средствами. Основывается данное утверждение на том, что цель практически любой атаки – реализоваться, поэтому любая атака будет себя как-то проявлять. Она или запустит/остановит какой-то процесс, или изменит какой-то файл или несколько файлов на диске. В любом случае, чтобы после перезагрузки компьютера его не пришлось повторно атаковать на его жёстком диске либо другом носителе, скорее всего будут сделаны изменения. В результате реализации атаки могут быть изменены загрузочные файлы либо

часто используемые утилиты, например команда ls в *nix. Для борьбы с подменой или искажением файлов можно выделить группы важных неизменяемых файлов или областей диска и периодически проверять их неизменность. Некоторый аналог вспаханной полосы на границе.

Средства проверки могут быть от самых простых – проверка по размеру и времени файла – до сложных, когда используется хэш не только с содержимого файла, но и от его месторасположения на диске. Реагирование на изменение того или иного файла может быть различным от запуска заранее определённого сценария до замены файла «новым». Например, Windows XP, если обнаруживает замену некоторых своих файлов чужими, может восстановить их без ведома пользователя. Это не всегда удобно, например, если по ошибке поменять папку с пользовательскими документами как системную. Поэтому чаще всего такие средства не обходятся без контроля человеком – они высылают предупреждения администратору, а тот уже сам решает, как поступать.

В качестве примера, демонстрирующего удобство и наглядность данных средств защиты, можно рассмотреть случай организации веб-сервера, защищаемого таким образом от взлома. Для этого устанавливаются два компьютера. Один с доступом во внешний мир, на котором запущен потенциально взламываемый http-сервер, а другой для обновления, на нём нет службы http-сервера, возможно, и доступа ко внешней сети у него тоже нет. Периодически второй компьютер по внутреннему каналу связи проверяет содержимое первого и загружает туда изменения. Таким образом, однажды заменённые файлы просуществует до первой проверки, после которой изменения будут восстановлены и будет выслано сообщение администратору о необходимости его дальнейшего вмешательства. Данная схема недостаточно надёжная по причине того, что уязвимость не закрывается и сервер могут сломать повторно, но всё же реализуемая в ряде случаев.

Следует отметить, что большой процент инцидентов не регистрируется, поэтому многие атаки оказываются неучтёнными. Общее число атак день ото дня растёт с завидным постоянством. Подробнее см. ниже данные CERT. Появление нежелательного трафика к хосту является уже нормальным явлением, а различные сканирования уже не рассматриваются как инциденты, требующие немедленного вмешательства по причине того, что их очень большое число, а найти и доказать злой умысел того или иного нарушителя является непростой задачей. Возникает вопрос выбора средств по отсеву более опасных событий от менее опасных. Хост-ориентированные СОА путём исследования лог-файлов и средства контроля целостности файловых систем не всегда могут своевременно сделать выводы о начавшейся атаке до того, как станут ощутимыми её последствия. Чтобы не подвергать существующие системы большому риску, следует выбирать другие средства. Такими средствами могут быть виртуальные ловушки.

Ловушки

Использоваться виртуальные ловушки стали только в последнее время. И их появление вызвано насущной необходимостью. В связи с увеличением числа атак стало очень сложно классифицировать атаки между собой по степени их опасности. Например, какой-нибудь школьник, взламывая гипотетический НИИ, может не знать, чей узел он взламывает. Он может добраться до каких-то научных расчётов, которые он тут же сотрёт, так как более ценной информацией для него будут, например, пароли для Интернета. Поэтому утечки информации в данном случае не будет, как если бы эти расчёты попали в руки спецслужб других стран. СОА не в состоянии классифицировать случаи взлома по-разному. Создав ложные службы, узлы или сети, мы можем спровоцировать нападающего на их взлом, тем самым получить больше информации о взломах, методиках и пр., из которых соответственно можно будет принять правильное решение на основе собранных данных, которые могут служить доказательной базой наличия злого умысла у атакующего в суде. В случае использования ловушек также решается и другая проблема – опасность вывода из строя реальных служб. Если атаки производить на реальные службы и узлы, то последствия могут быть более значительными.

На сегодняшний день существует огромное число виртуальных ловушек, большинство из которых могут не только собирать данные, но и предпринимать какие-то активные действия по отношению к нарушителю. В зависимости от способов установки и работы ловушки бывают следующих видов. См. рисунок 2.

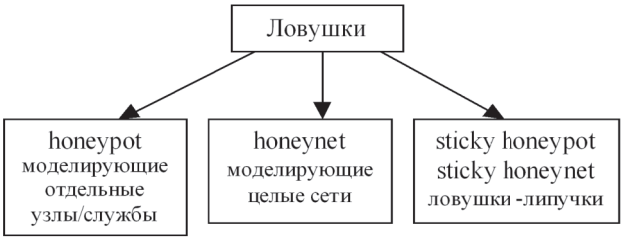


Рисунок 2. Виды ловушек

Рассмотрим преимущества тех или иных видов ловушек с точки зрения поимки нарушителей.

Ловушки, моделирующие отдельные службы, хороши тогда, когда имеется некоторый известный узел, выполняющий функции какого-либо сервера, например, веб-сервера или шлюза. С большой вероятностью данный сервер будет атаковаться, и будут попытки взлома. Если нет уязвимостей или их наличие маловероятно, то все атаки и попытки взлома будут просто-напросто растрачивать ресурсы сервера. При этом разделение случаев атак от легального использования будет осложнено. Создание на этом же сервере (логически) других фиктивных сервисов заставит атакующих распараллелить свои действия. Так как потенциально возможных уязвимых сервисов будет больше, то и вероятность успешной атаки с точки зрения атакующих будет больше. Эту вероятность можно будет заведомо повысить, выдав ложные сведения о плохой защищённости подставных виртуальных ловушек. Между тем легальные пользователи не будут работать с ловушками или реальными службами нештатными способами, пытаясь вывести те из строя, основываясь на ложной информации, полученной прямым или косвенным способом от ловушек о наличии тех или иных уязвимостей. Таким образом, можно понизить вероятность ошибок первого и второго рода. Однако при данной реализации есть некоторая

опасность нарушения функционирования реальных служб. Также нарушитель не всегда может впоследствии воспользоваться ложной информацией об уязвимостях и атаковать предоставленные ему ловушки. Безопаснее ловушки, моделирующие узлы, ставить виртуально или использовать, ловушки, моделирующие работу отдельных узлов.

Ловушки, моделирующие отдельные узлы, хороши с точки зрения безопасности, так как они ставятся отдельно. В случае их компрометации ущерб будет меньше, чем если бы они стояли на реальных узлах. Моделирование целых узлов даёт больше возможностей для реализации тех или иных моделируемых конфигураций.

Для реализации одних и тех же служб могут использоваться различные платформы. При этом атакующий, если ничего не знает об уязвимостях какой-то платформы, может не воспользоваться предоставляемым ему «куском сыра в мышеловке» и тем самым не быть пойманным. Поэтому для увеличения вероятности поимки атакующего следует предоставить ему возможность выбора, тем более использование различных платформ в одной сети одновременно – не редкость. Попав в подобный клондайк, нарушитель, возможно, захочет пойти по наиболее лёгкому пути и атаковать более известную ему платформу. На данном этапе уже можно составлять некоторые «портреты», характеризующие нарушителей. Если нарушитель окажется в состоянии сломать все предоставленные ему ловушки, то опять же это охарактеризует его уникальным образом и позволит выделить среди других производимых атак. Нельзя не сказать, что подобные характеристики о взломщиках могут дать много полезной информации в дальнейшем, при доказательстве злого умысла нарушителей и при использовании обработки информации с распределённых телекоммуникационных систем.

Создание отдельных виртуальных ловушек, объединённых в сеть, может обойтись довольно дорого, поэтому дешевле наряду с отдельно моделируемыми хостами моделировать целые сети с их замысловатой топологией. Ловушки, моделирующие целые сети, позволяют моделировать случайным образом задержки между узлами, потерю пакетов, создавая полную иллюзию реальности происходящего.

Организациям обычно выдаются диапазоны адресов, но не все и не всегда используются, размещение ловушек на неиспользуемых адресах может дать необходимый результат. Любое обращение к неиспользуемому адресу есть вероятная атака. Сканирующий снаружи нарушитель не может знать, что к какому-то узлу или узлам не следует обращаться. Поэтому нарушитель может быть сразу же замечен и взят под контроль какой-либо из систем обнаружения атак.

Ловушки-липучки призваны затруднять действия нарушителей, заставляя их большую часть времени проводить в бессмысленном ожидании. Несмотря на то, что взлом любой из ловушек есть бесполезная, с точки зрения нарушителя, трата времени, количество потраченного времени можно увеличить. Например, если поставить ловушку-липучку, проверяющую работающие узлы в сети и отвечающую вместо неработающих на попытки извне установить соединения, то снаружи при проведении сканирования может создаться иллюзия работы всех узлов, в данном случае вопрос взлома того или иного реального узла будет осложнён, так как атакующему будет необходимо найти реально работающие узлы либо попытаться атаковать все подряд, на что, несомненно, уйдёт время. Также при потенциальном подозрении на атаку можно увеличивать фрагментацию и задержки для пакетов, которыми ведётся обмен с нарушителем, имитируя загруженность сети. При этом время нарушителя будет растрачиваться понапрасну в ожиданиях. Это время может быть использовано при проведении оперативных мероприятий, где каждая лишняя минута может быть решающей.

В силу специфики, варианты размещение ловушек могут очень сильно отличаться друг от друга. См. рисунок 3.

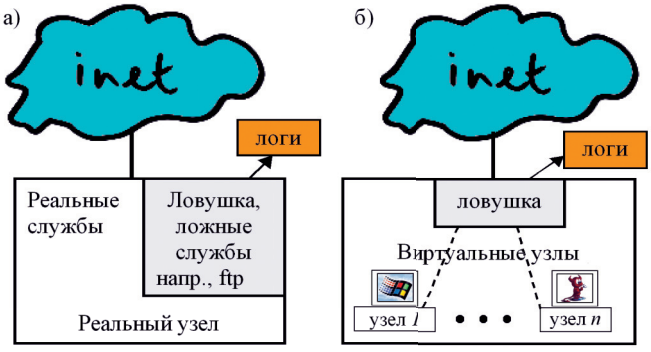


Рисунок 3. Варианты размещения ловушек:
а) ловушка на отдельном хосте; б) ловушка-сеть

Важным моментом работы ловушек есть сбор логов их работы и, по возможности, всего трафика обмена с ними. Данная информация является очень важной с целью её анализа после совершения той или иной атаки или только попытки. В случае реализации новой атаки, для которой нет записи в сигнатурной БД, по этим данным можно будет составить сигнатуры, которые в дальнейшем можно включить в БД СОА реальных систем, повысив их эффективность.

В случае использования нескольких ловушек или ловушек-сетей, информация об их работе может собираться в едином центре для анализа с целью обнаружения распределённых телекоммуникационных атак.

IDS Snort

Введение

После небольшого введения в теорию я расскажу о практике, а именно, с чего собственно и хотел начать статью – про систему обнаружения атак IDS Snort. Введение появилось не случайно. Связано это с тем, что очень мало людей занимается вплотную вопросами обнаружения атак, поэтому наблюдается информационный вакуум в этой области. Читателю должно стать чётко ясно, где именно используется Snort и каким маленьким винтиком он является в большой теме обнаружения телекоммуникационных атак.

Несмотря на то, что IDS расшифровывается как Intrusion Detection System и дословно переводится как Система Обнаружения Вторжений (СОВ) использование этого термина не совсем правильно и в силу возможностей великого и могучего русского языка мы будем говорить о Системе Обнаружения Атак (СОА) Snort. Простой пример в правильности выбранного термина: атака может быть совершена, но

безуспешно, то есть вторжения не было. Система, отвечающая первому названию, не должна ничего обнаружить, в то время как вторая, и тот же Snort, обнаруживают именно атаки независимо от их последствий. Конечно, тут есть и доля философская, читатель может спросить, а какой мне смысл от того, что мы обнаружили атаку, которая уже сделала своё «чёрное дело»? И вопрос будет кстати, так как он подтвердит нужность всего вышенаписанного введения, что есть способы и программы защиты, а есть программы, обнаруживающие, например, атаки. Полная аналогия тому, если вы захотите поставить видеокамеру над входной дверью. От взломщиков она никак не спасёт, дверь будет взломана одинаково, вне зависимости от наличия или отсутствия видеокамеры, надо ставить хорошую дверь и хороший замок, но камера может дать сигнал хозяину о необходимости принятия дополнительных мер, записать действия взломщиков, что может служить доказательной базой в суде по факту взлома. Однако если хозяин не предпринимает никаких мер, то и польза от такой камеры минимальна. Всё это говорит о том, что обнаружением атак надо заниматься серьёзно и постоянно уделять этому внимание. Всё описанное будет реально полезно тем, кто будет целенаправленно и практически ежедневно уделять время и работать по вопросам обнаружения атак. Статья должна заложить базу для развития в этом направлении. Для всех же остальных, кто поленится или сочтёт тему скучной, статья может быть полезной только с точки зрения расширения кругозора.

История

Данный продукт, а вместе с ним и выпускающий его одноимённый проект является уже немолодым, первые упоминания датируются январём 1999 года, собственно, когда его основателю (Martin Roesch) и пришла в голову мысль о создании некоторой логической структуры правил по выборочному перехвату трафика. С этого момента и начало зарождаться некоторое подобие простой системы обнаружения атак. Что можно отметить последующим появлением версии 1.0. Уже к концу января появляется законченный продукт, который в полной мере может «ловить» атаки и называться системой обнаружения атак. Система может на основании заданных правил ловить атаки и структурированно вести лог-файлы. Правила можно задавать в любом удобном для пользователя порядке, и система толерантно относится к этому. Однако созданный продукт не устраивает разработчика в полной мере, и есть ещё ряд вещей, которые хочет внедрить Мартин в свой продукт с целью его улучшения. А именно: ставятся задачи создания возможности задания правил и обработки TCP-пакетов на основании выставленных в них флагах. Обработка фрагментированных пакетов и др. В это время Мартин меняет работу и привыкает к своей новой эргономичной клавиатуре, что сказывается на замедлении темпов процесса разработки. Основные силы переключаются на исправление ошибок, и появляется идея улучшить читаемость правил, в ухудшение удобности их набирания пользователями. Правила должны стать более длинными, менее удобными, но зато более понятными. Переписывается заново ядро обработки пакетов, за счёт чего увеличивается скорость их обработки, по просьбам «трудящихся» вводится более точное определение времени прихода пакета в систему, с учётом миллисекунд. В конце марта – начале апреля добавляется ранее задуманная обработка фрагментированных пакетов и другие вещи. В конце апреля добавляется возможность разбора PPP- и SLIP-пакетов, что позволило модемным пользователям применять Snort в полной мере. К проекту постепенно начинают подключаться всё больше и больше людей: Chris Sylvain, Damien Daspit и другие. Благодаря им у Snort появляется ряд новых полезных свойств. Проект идёт по миру и развивается. К началу лета выходит уже версия 1.2 с частично переписанным кодом. Однако обнаруживаются ошибки в работе Snort на платформах, отличных от Linux, в результате чего в августе появляется версия 1.2.1 с исправлениями, которая без проблем может запускаться как на Sparc, так и на OpenBSD. Чуть позже, в конце сентября (1999), ставится точка на активном развитии версии серии 1.x, и выдвигается идея о создании улучшенной с радикально переписанными кусками кода второй серии версий с номером 2.0. Всё самое лучшее – во вторую версию. Работа над второй версией идёт медленно, в то же время и не прекращается улучшение первой. В конце декабря выходит версия 1.5. Новый год знаменуется выходом версии 1.6. К середине года выходит версия 1.6.1, и ставится задача по выпуску версии 1.7 к октябрьской конференции «SANS Network Security 2000» [31]. Поспешный выход 1.6.1 приводит к выходу 1.6.2.2, и далее 1.6.3. Версию 1.7 удаётся выпустить только в начале 2001 года. Возможно, это было сделано специально, с целью ознаменования больших трудов на протяжении полугода. Версия 1.7 уже без проблем компилируется и работает на таких платформах, как:

- Linux 2.0.X, Linux 2.1.X, Linux 2.2.X (i386)
- FreeBSD 3.x, 4.x (i386)
- SunOS/gcc 5.5, 5.5.1, 5.6, 5.7, 5.8 (sparc)
- OpenBSD 2.7, 2.8
- Tru64/gcc
- HPUX 11.0/gcc

Ещё через полгода выходит новая версия, идея по выходу второй версии вот-вот осуществится, на этот раз в разрабатываемую версию уже включено всё самое лучшее, чего удалось добиться в версии 1.7. Далее, во второй половине лета у Мартина рождается сын, и поэтому его усилия более переключаются на него. Несмотря на это актуальность выпуска второй версии не уменьшается. До конца года выходят версии 1.8.1 с множеством дополнений и новых функций и сразу же после выходят исправления к новым ошибкам в 1.8.2. В новом, 2002 году выходит версия 1.8.3, и продолжает улучшаться вторая версия. В 2003 году выходят завершающие версии 1.9.x и долгожданная вторая версия, туда уже включены следующие функции, многие из которых даже и не принадлежат руке Мартина:

- Улучшенная производительность (благодаря новым образцам для фильтров и переписанному блоку сравнения).
- Лучшее декодирование.
- Расширенные возможности дефрагментирования и повторной сборки пакетов.
- Исправлены тысячи ошибок.
- Обновлены правила для обнаружения атак.
- Обновлён файл конфигурации Snort.conf.

- Введены новые ключевые слова и соответствие образцам.
- Новый анализатор HTTP-потоков.
- Расширены возможности аномального детектирования (для HTTP, RPC, TCP, IP и др. протоколов).
- Лучшая защищённость программы.
- Исправлена ошибка Xrefs.
- Flexresp работает быстрее и эффективнее.
- Лучшие возможности chroot().
- Исправлено декодирование 802.1q.

Количество постоянных разработчиков Snort уже превысило 13, а количество людей, как-то оказавших помощь проекту, уже исчисляется сотнями. И вот летом выходит улучшенная «вторая» версия – 2.0.1, а уже в начале осени, 17 сентября 2003 года, выходит версия 2.0.2, установкой и настройкой которой мы и займёмся чуть ниже. Что же касается разработчиков, то они ни на минуту не останавливаются и продолжают совершенствовать данный продукт. Одним из направлений развития предполагается создание систем из 5/10/20/100 машин с установленным Snort в виде сенсоров с целью централизованного обнаружения и мониторинга больших сетей, в том числе и с целью обнаружения распределённых телекоммуникационных атак. По крайней мере Мартин просит всех заинтересованных в данном проекте дать о себе знать [32]. Возможно, удастся обсудить тему нескольких сенсоров в будущих публикациях, а пока непосредственно перейдём к установке самой простой конфигурации Snort с введением логов в файл.

Начальная установка IDS Snort

Для начала нам необходимо зайти на сайт [15] и скачать оттуда последнюю версию. Сейчас это версия 2.0.2 http://www.Snort.org/dl/Snort-2.0.2.tar.gz. Далее необходимо удостовериться в том, что мы скачали что надо, для этого мы скачиваем контрольную сумму http://www.Snort.org/dl/Snort-2.0.2.tar.gz.md5. (Также можно проверить подлинность с помощью имеющейся PGP-подписи.) Подсчитываем контрольную сумму от скачанного файла:

```
# md5sum snort-2.0.2.tar.gz
```

В результате запуска получим:

```
21b14d90e2a323831d85f3d845d64b23   snort-2.0.2.tar.gz
```

Сверив это значение с тем, что записано в файле Snort-2.0.2.tar.gz.md5, и убедившись, что они идентичны, мы можем приступать к распаковке и установке пакета. Не следует пренебрегать этим, казалось бы, излишним шагом. В 2002 году уже имелись инциденты по взлому различных веб-сайтов, в том числе и различных официальных зеркал, с которых скачивались «модифицированные» программы [24].

Все возникающие вопросы мы будем решать по мере их появления. Содержимое полученного архива следует куда-либо записать, удобнее это сделать в mc. Мы создадим для этого директорию /progi, куда и запишем содержимое скачанного архива.

Рисунок 4. Screenshot скопированного содержимого архива

В директории /progi/Snort-2.0.2/contrib мы можем найти множество полезных программ и утилит, которые не являются частью Snort, но значительно могут помочь в работе со Snort. Многие вещи нам в первый раз не понадобятся, они скорее нужны для продвинутых пользователей. Среди представленных программ вы найдёте готовые скрипты с командами по созданию таблиц в различных БД: MSSQL, MySQL, PostgreSQL, Oracle с целью ведения логов в БД. (Организация ведения логов в другие БД открыта и не представляет сложностей, разве что придётся написать скрипты самостоятельно.) Делать выборки и различные операции с БД более удобно, чем с обычными текстовыми лог-файлами. Для более-менее простого просмотра БД с логами мы там же можем найти ACID – систему визуального отображения содержимого БД через php и веб-сервер.

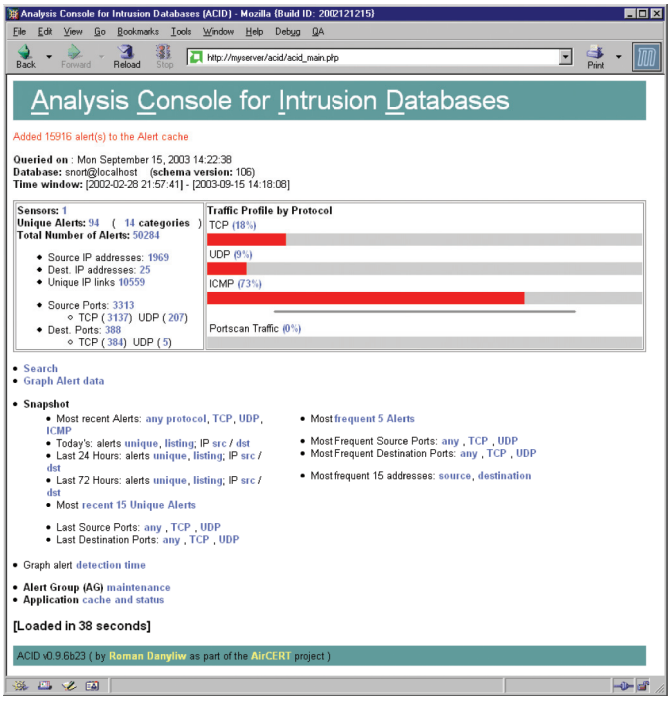


Рисунок.5. Вид ACID через веб-браузер mozilla

Подробнее с ACID мы познакомимся в следующих статьях.

Если вас чем-то не устраивает ведение логов в БД, а просматривать лог-файлы вручную всё же вам кажется не очень удобным, то для удобства просмотра имеется

модуль SnortLog.pm для Perl в Net-SnortLog-0.1.tar.gz. Помимо этого имеется:

- Guardian – скрипт на Perl, позволяющий изменять правила пакетного фильтра (пока для ipchains) с файлом конфигурации и небольшим README.
- PassiveOS – скрипт для Perl, пытающийся по имеющимся логам определить, какую операционную систему имел атаковавший вас хост.
- mysql.php3 – php-скрипт для просмотра статистики работы Snort в online (10 наиболее частых TCP-опробований, 10 наиболее частых UDP-опробований и пр.), данные берутся из БД mysql.
- snort2html, скрипт на Perl, создающий отчёты в html.
- S99snort – стартовый скрипт, чтобы запускать Snort при загрузке системы из /etc/rc.d/init.d – он вам понадобится, если вас не устроит изложенный вариант решения ниже.
- snortdb-extra.gz – файл с правилами, создающими дерево таблиц в любой SQL92-совместимой БД, с целью более удобного изучения данных Snort пользователями.
- snortlog – ещё один небольшой скрипт на Perl, ведущий анализ логов.
- snortnet.tar.gz – набор программ, используемых проектом по обнаружению распределённых телекоммуникационных атак, когда Snort используется в качестве сетевого сенсора.
- snortwatch – программка на python, позволяющая отслеживать предупреждения, генерируемые Snort.
- Даже имеется система обнаружения аномальных событий на основе их вероятности – SPADE (Statistical Packet Anomaly Detection Engine).

Создаётся впечатление, что для Snort пишут свои добавления все, кому не лень это делать, что даже очень хорошо, так как это говорит о реальной популярности программы и гибкости её использования.

Пробежавшись далее быстро по всем встречающимся директориям, мы можем встретить документацию в doc, файлы конфигурации в etc, правила в rules, исходники в src и образцы препроцессоров для обработки данных в templates.

Для успешности последующих действий в вашем компьютере должны быть установлены средства разработки приложений, остальное всё, как обычно, перед компиляцией Snort необходимо запустить конфигурирование:

```
# ./configure
```

Для самого простого случая, как у нас, не надо передавать никаких параметров. (Забегая вперёд, для продвинутых пользователей скажу, что для конфигурации Snort для работы с БД MySQL следует указывать параметр --with-mysql, со всеми остальными рассмотрим этот режим работы Snort позже, возможно, даже в следующей статье). В процессе конфигурации осуществляется проверка на наличие всех необходимых компонент. Так как Snort работает в режиме сниффера, когда перехватывает пакеты, то он использует библиотеку libpcap. Если вы никогда не ставили её и не пользовались пакетом tcpdump, то у вас её может не оказаться, и тогда вы через несколько строчек после запуска увидите сообщение о произошедшей ошибке и необходимости установки библиотеки.

```
[root@natali snort-2.0.2]# ./configure
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for gawk... gawk
checking whether make sets ${MAKE}... yes
checking for style of include used by make... GNU
checking for gcc... gcc
checking for C compiler default output... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables...
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking dependency style of gcc... none
checking for gcc option to accept ANSI C... none needed
checking for ranlib... ranlib
checking for gcc... (cached) gcc
checking whether we are using the GNU C compiler... (cached) yes
checking whether gcc accepts -g... (cached) yes
checking dependency style of gcc... (cached) none
checking build system type... i686-pc-linux-gnu
checking host system type... i686-pc-linux-gnu
checking whether byte ordering is bigendian... no
checking for sparc alignment... no
checking how to run the C preprocessor... gcc -E
checking for ANSI C header files... yes
checking for sys/types.h... yes
checking for sys/stat.h... yes
checking for stdlib.h... yes
checking for string.h... yes
checking for memory.h... yes
checking for strings.h... yes
checking for inttypes.h... yes
checking for stdint.h... yes
checking for unistd.h... yes
checking for strings.h... (cached) yes
checking for string.h... (cached) yes
checking for stdlib.h... (cached) yes
checking for unistd.h... (cached) yes
checking sys/sockio.h usability... no
checking sys/sockio.h presence... no
checking for sys/sockio.h... no
checking paths.h usability... yes
checking paths.h presence... yes
checking for paths.h... yes
checking for inet_ntoa in -lnsl... yes
checking for socket in -lsocket... no
checking whether printf must be declared... no
checking whether fprintf must be declared... no
checking whether syslog must be declared... no
checking whether puts must be declared... no
checking whether fputs must be declared... no
checking whether fputc must be declared... no
checking whether fopen must be declared... no
checking whether fclose must be declared... no
checking whether fwrite must be declared... no
checking whether fflush must be declared... no
checking whether getopt must be declared... no
checking whether bzero must be declared... no
checking whether bcopy must be declared... no
checking whether memset must be declared... no
checking whether strtol must be declared... no
checking whether strcasecmp must be declared... no
checking whether strncasecmp must be declared... no
checking whether strerror must be declared... no
checking whether perror must be declared... no
checking whether socket must be declared... no
checking whether sendto must be declared... no
checking whether vsnprintf must be declared... no
checking whether snprintf must be declared... no
checking whether strtoul must be declared... no
checking for snprintf... yes
checking for strlcpy... no
checking for strlcat... no
checking for strerror... yes
checking for __FUNCTION__... yes
checking for floor in -lm... yes
checking for pcap_datalink in -lpcap... no

   ERROR!  Libpcap library/headers not found, go get it from
   http://www.tcpdump.org
   or use the --with-libpcap-* options, if you have it installed
   in unusual place
[root@natali snort-2.0.2]#
```

Рисунок 6. Сообщение с ошибкой во время запуска конфигурирования о необходимости установки библиотеки libpcap

В RedHat 7.3 libpcap-0.6.2-12.i386.rpm находится на втором диске в директории /RedHat/RPMS. (В RedHat 9.0 файл уже называется libpcap-0.7.2-1.i386.rpm и находится тоже на втором диске в той же директории).

Устанавливаем библиотеку (с поправкой на вашу версию):

```
# rpm -i libpcap-0.6.2-12.i386.rpm
```

и запускаем конфигурирование повторно.

В случае успешного завершения, то есть отсутствия сообщений об ошибках, можем запустить компиляцию командой:

```
# make
```

из директории /progi/snort-2.0.2, куда мы распаковали пакет. После завершения компиляции у нас в директории /progi/snort-2.0.2/src должен появиться запускаемый файл – snort. Это и есть готовая программа. (У меня её размер 1224986). Если её просто запустить, то она выдаст информацию о ключах и пожалуется на то, что мы, «глупые» пользователи, не сообщили, что ей после запуска нужно делать: «Uh, you need to tell me to do something...» Программа готова к работе, задавая нужные ключи, можно запустить её работать, однако более разумным будет предварительное копирование программы в систему, то есть «установка» на своё место. С помощью команды:

```
# make install
```

программа копируется в директорию /usr/local/bin, устанавливается man. После того как программа установлена, наступает самый интересный, а может, и самый муторный момент – конфигурация и создание правил, от корректности задания которых будет зависеть – будут ли ловиться те или иные атаки или нет. Цитата из раздела про sendmail в [33, стр. 718] гласит: «Если вы увидите человека, который начинает создание конфигурационного файла с пустой страницы в текстовом редакторе, можете с чистой совестью звонить в 03... Не имеет никакого смысла создавать этот файл с нуля (особенно если учесть его сложность) – гораздо проще изменить существующий. Только не забудьте сделать перед этим его копию!» То же самое можно сказать и про конфигурационный файл Snort – snort.conf. Конечно, он несколько проще его вышеупомянутого собрата, но незначительно. Готовый файл имеется в директории etc (/progi/snort-2.0.2/etc/snort.conf). Имя и размещение для файла конфигурации нигде не определены, однако де-факто файл называется snort.conf и размещать мы его будем в директории /etc/snort, которую необходимо создать. Далее в директории /etc/snort создадим поддиректорию rules, позже мы туда поместим правила. Правила мы также возьмём готовые и рассмотрим парочку правил, чтобы было ясно, как они создаются, на случай, если нам придётся их править и писать свои правила. Готовые правила можно взять из дистрибутива, однако они могут несколько отставать от жизни. Поэтому лучше брать ежедневно обновляемые правила на сайте Snort в разделе download: http://www.snort.org/dl/rules/snortrules-stable.tar.gz. Они подходят как для версий 1.9.x, так и для всех версий 2.0.x на момент написания статьи. Вопросы автоматического скачивания и обновления правил в этой статье мы рассматривать не будем. Также во всех файлах с правилами имеется шаблон файла конфигурации. Из файла с последними правилами или из дистрибутива переписываем все файлы с расширением .rules в директорию /etc/snort/rules, а оставшиеся несколько файлов в /etc/snort:

- classification.config
- generators
- gen-msg.map
- Makefile.am
- reference.config
- sid
- sid-msg.map
- snort.conf

Не все файлы нам нужны, и необязательно правила писать в отдельную директорию, просто мне кажется, что так будет удобнее, когда «мухи отдельно, котлеты отдельно».

Язык написания правил довольно специфичный, но в то же время простой и понятный. Большинство правил настолько просты, что умещаются в одну строчку. В старых версиях до 1.8 это даже было необходимо. Сейчас же правила могут переноситься с помощью обратного слеша «\» в конце строки. Для удобства правила состоят из двух логических частей: заголовка и опций.

Заголовок определяет, какие пакеты следует смотреть, ***опции*** определяют, на что следует смотреть в перехваченных пакетах. Так, правило по «обнаружению водителя с пассажиром на жёлтых «Жигулях» будет выглядеть так: «Сообщить в центр, если мимо поста будут проезжать жёлтые «Жигули», едущие из центра в область по Рублёвскому шоссе с номером «22-33» днём с включёнными фарами, если водитель негр, а пассажиры – стройная блондинка с собачкой, о том, что проехала машина А.». Если применять это всё сразу к правилам и проводить аналогию, то первая часть правила – это «сообщить в центр, если мимо поста будут проезжать жёлтые «Жигули», едущие из центра в область по Рублёвскому шоссе», а вторая – «с номером «22-33» днём с включёнными фарами, если водитель негр, а пассажиры – стройная блондинка с собачкой, о том, что проехала машина А.». «Сообщить в центр» – это будет действие. «Жёлтые «Жигули» – это протокол. «Едущие из центра в область» – это направление следования, порты и IP-адреса отправителя и получателя. «По Рублёвскому шоссе» – это будет интерфейс, который будет слушать Snort, к правилам это отношения не имеет. А всё остальное будет опцией с условием. Если выразить это коротко и без комментариев, то получится, что заголовок содержит информацию о принимаемом действии по данному правилу, протокол, порты и IP-адреса отправителя и получателя с сетевыми масками. Опции же определяют, какую часть пакета стоит просматривать на соответствие чему-то и какой текст выдавать. Часть пакета – это будет салон автомобиля – смотрим на пассажира и водителя, а какой текст выдавать – «проехала машина А.». Простой пример:

```
alert tcp any any -> 192.168.1.0/24 111 ↵
    (content:"|00 01 86 a5|"; msg:"mountd access";)
```

До скобок это заголовок правила, в скобках – опции. Пояснять правило, думаю, не имеет смысла, и так должно быть понятно.

Теперь пример посложнее, давайте возьмём файл /etc/snort/rules/ftp.rules и рассмотрим его первую непустую строчку с правилом:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21 ↵
    (msg:"FTP CEL overflow attempt"; ↵
    flow:to_seserver,established; ↵
    content:"CEL "; ↵
    nocase; ↵
    content:!"|0a|"; ↵
    within:100; ↵
```

```
reference: bugtraq,679; ↵
reference:cve,CVE-1999-0789; ↵
reference:arachnids,257; ↵
classtype: attempted-admin; ↵
sid:337; ↵
rev:5;)
```

Правило выглядит более «навороченным», но несмотря на это, его можно по кусочкам изучить, и думаю, что читатель, знающий английский язык, без труда поймёт содержание правила. Добавлены комментарии, чтобы было проще найти информацию об уязвимостях, на которые направлено это правило, чтобы точно знать, когда указанная атака действительно может повлиять на систему. Если для данной атаки ваша система неуязвима, тогда и беспокоиться не стоит.

Внимательный читатель заметит, что вместо адресов в последнем правиле используются переменные $EXTERNAL_NET и $HOME_NET. Введение переменных очень удобно, и оно предусмотрено синтаксисом файла конфигурации, где могут писаться правила. Можно один раз задать постоянные и часто неменяемые параметры сети, и не придётся после менять их во многих местах. Настройка Snort по большей части как раз и сводится к заданию подобных уже выбранных переменных в файле конфигурации. Помимо этого к каждой такой переменной идут комментарии.

Чтобы файл конфигурации не «распухал», используется опция include с именем включаемого файла, аналогичная языку C. Разве что задаётся она без знака решётки в начале строки. Знак решётки, как обычно, используется для комментирования ненужных строк.

Думаю, что не имеет смысла расписывать все возможные опции для правил вроде TOS и TTL, так как это есть в обширной и очень понятной документации и любой читатель при необходимости в написании своих правил может заглянуть на страничку с документацией к Snort в разделе Documentation и выяснить для себя детали синтаксиса.

А пока давайте рассмотрим основные шаги внесения изменений в конфигурационный файл snort.conf, чтобы в конечном итоге наша система обнаружения атак худобедно заработала.

Первое, что мы видим в конфигурационном файле, – это инструкция о необходимости выполнения конфигурации в 4 этапа:

1. Установка сетевых переменных;
2. Конфигурирование препроцессоров;
3. Конфигурирование и определение, куда будет вестись вывод;
4. Выбор необходимых правил.

1-й этап

Переменная $HOME_NET определяет IP-адреса, считаемые адресами нашей домашней сети. Это нужно, чтобы Snort знал, что считать своим, а что чужим. Например, если мы имеем одну машину с адресом 123.45.45.45, то следует написать:

```
var HOME_NET 123.45.45.45
```

В случае маскируемых сетей и если их несколько, пишем без пробелов в скобках:

```
var HOME_NET [10.1.1.0/24,192.168.1.0/24]
```

Можно использовать ключевое слово «any» – оно означает любой адрес. Далее следует определить адрес внешней сети. Обычно это все адреса:

```
var EXTERNAL_NET any
```

Далее идёт секция с выбором серверов, используемых в сети. Нужно это для того, чтобы не распылять ресурсы напрасно и повысить эффективность работы. Вопрос, конечно, открытый: какой смысл ловить атаки на веб-сервера, если у вас в сети нет ни одного веб-сервера? Но большая часть ответит, что никакого, поэтому в этой секции следует убрать лишнее либо задать более конкретно. Спешу вас предупредить, что бездумное комментирование этих строчек может привести к не запуску Snort и сообщениям об ошибках, так как некоторые включаемые ниже файлы с правилами, возможно, будут иметь ссылки на несуществующие переменные. Далее определяются порты для некоторых служб и прочие переменные. Для первого запуска лучше бегло пробежать, пропустив некоторые сложные моменты, и запуститься, а после уже заниматься более детальным изучением тех или иных правил и строчек в конфигурационном файле с целью выбора оптимальных настроек. Переменную $RULE_PATH следует определить как rules:

```
var RULE_PATH rules
```

а не:

```
var RULE_PATH ../rules
```

что написано по умолчанию, либо необходимо переписать правила в другую директорию.

2-й этап

Конфигурирование препроцессоров. На первый взгляд может показаться, что препроцессоры – это лишняя и ненужная вещь, и, наверное, это будет по большей части правдой, только если не одно «но», что в сети одно и то же сообщение может быть разбито на две части и на много маленьких пакетов вплоть до содержания в 1 байт. Такие пакеты, не обнаруживая себя, будут проходить через любую СОА без особых трудностей; в зарубежной литературе, в том числе и электронной, эта классическая ситуация очень хорошо и с рисунками расписана. Поэтому разумнее накапливать все маленькие пакеты в буфере и пересобирать до нужного размера, а после уже проверять на наличие признаков той или иной атаки. Пересобирать можно и обычные, среднестатистические пакеты при необходимости. Также, если необходимо, препроцессоры могут выполнять предварительные преобразования веб-адресов. Не секрет, что через процент и шестнадцатеричное число в строке адреса можно передавать лю-

бые символы. Так, в истории есть один интересный факт, ставший примером классической глупости компании Microsoft, связанной с этим преобразованием, когда оная, обнаружив одну ошибку, исправила её другой. Некоторое время назад в серверах IIS была найдена ошибка, которая заключалась в том, что если в конце URL-адреса какого-либо asp-сценария дописать точку, то вместо запуска сценария на выполнение выводился его исходный текст. (Подобная ситуация была и со знаком пробела). На сообщения об этом известная фирма выпустила тут же заплатку, заключавшуюся в установке простой проверки синтаксиса запроса на наличие в конце знака «.» и удалении его перед передачей запроса непосредственно обрабатывающему его серверу. На что специалисты только посмеялись, а взломы серверов продолжились как ни в чём не бывало, с тем лишь различием, что в конце запроса вместо точки стал использоваться её шестнадцатеричный код со знаком процента, который беспрепятственно пропускался всеми фильтрами. Этот обходной манёвр несколько позже тоже прикрыли, но факт остался фактом. Поэтому лучше 7 раз подумать и один раз сделать, чем один раз подумать и 7 раз переделывать. Если, прочитав комментарии к препроцессорам, вы поняли, что всё ещё не сильны в их настройке или даже не поняли, зачем они вообще нужны, то лучше оставьте пока эту секцию без изменений.

3-й этап

Конфигурирование параметров вывода. Здесь мы указываем, куда у нас будет осуществляться вывод, в БД или в обычный лог-файл. Если в БД, то можно задать различные параметры, если в файл, то можно задать имя файла. Подробнее о ведении логов в БД и настройках, я думаю, я напишу в следующих статьях, а пока мы попытаемся настроить вывод в обычный файл, как и планировалось. (Замечание: напоминаю, что для вывода в БД необходима компиляция с поддержкой той или иной БД). Для вывода в файл надо оставить всё как есть по умолчанию и создать директорию, куда будут вестись логи – это /var/log/snort. Необходимо проследить, чтобы у Snort было достаточно прав на запись в неё.

4-й этап

Выбор тех или иных уже готовых правил обычно происходит с появлением некоторого опыта, поэтому мы оставим всё как есть по умолчанию.

После внесения изменений мы сохраняем конфигурационный файл и запускаем Snort от имени суперпользователя командой:

```
# /usr/local/bin/snort -o -i eth0 -d -c /etc/snort/snort.conf
```

(вместо eth0 можно указать любой другой прослушиваемый интерфейс), использованные опции означают:

- -o – сменить порядок применения правил с Alert → Pass → Log order на Pass → Alert → Log order, это ускоряет несколько работу.
- -i eth0 – слушать указанный интерфейс. Можно опустить, если интерфейс один в системе.
- -d – выводить содержимое уровня приложения в пакетах, если стоит режим избыточности вывода или ведения учёта пакетов (дополнительная информация нам не помешает).
- -c /etc/snort/snort.conf – использовать указанный конфигурационный файл.

Имеется ещё одна полезная опция «-l имя_директории» – она позволяет задавать ведение логов в какую-либо конкретную директорию. По умолчанию это /var/log/snort, поэтому мы не стали задавать этот излишний параметр.

После этого, если программу не прерывать, то она висит на консоли и ловит атаки – это не самый удобный способ запуска, но для понимания происходящего и отладки – самый лучший. При этом на каждый IP-адрес, с которого идут атаки, создаётся поддиректория в директории, куда ведутся логи, и там создаётся файл с коротким названием сокращённо от протокола и портов зарегистрированной атаки, например: TCP:3739-1080 или TCP:21-21, или ICMP_ECHO. В частности, пока я отлаживал запуск в процессе написания этой статьи, за несколько секунд успело зарегистрироваться несколько реальных атак.

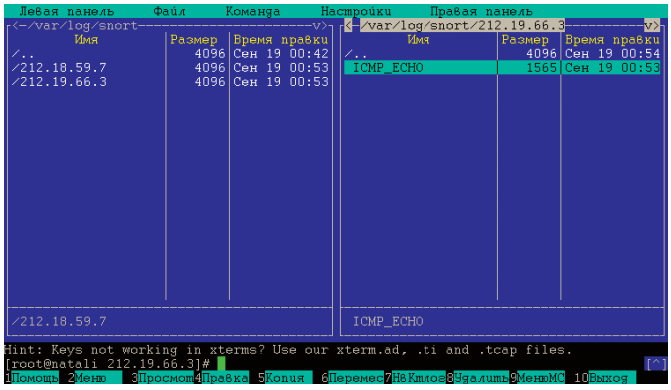


Рисунок 7. Вид директории, куда ведутся логи

Внутри файлы имеет более подробное содержание произошедшего, например, просмотрев файл ICMP_ECHO, мы получаем информацию, что были атаки на узлы xx.xx.xx.7, xx.xx.xx.15, xx.xx.xx.16, что показано на рис. 8.

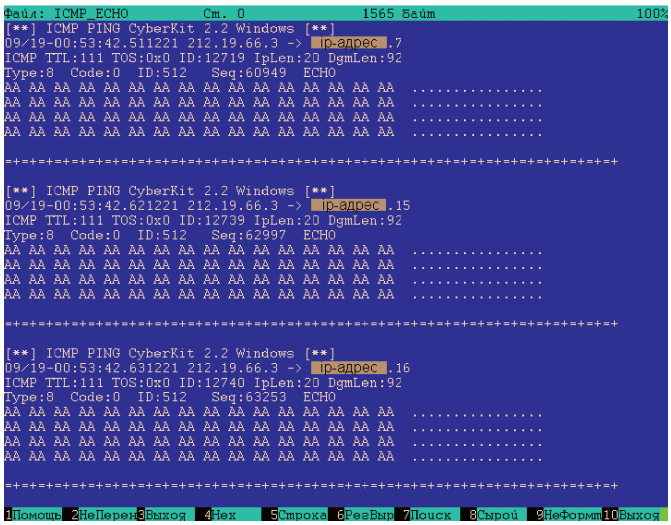


Рисунок 8. Пример содержимого файла с информацией об атаках

Замечание. Из-за нашествия червя Blaster в последнее время (начиная с августа), которое стало возможным благодаря ошибкам в Windows NT/2000/XP и Windows 2003 Server, «холостой» трафик на все узлы в сети увеличился практически в десять и более раз за счёт подобных пакетов. Поэтому не удивляет тот факт, что за малый промежуток времени были зарегистрированы атаки.

Запускать Snort руками хорошо только в отладочных целях, когда же мы убедились, что он работает, необходимо подумать об автоматическом его запуске.

Автоматизация процесса запуска Snort

Изначально вопрос автоматического запуска никак не был продуман, и каждый запускал его как хотел из стартовых скриптов. Потом кто-то выложил свои скрипты в сети, а потом стартовый скрипт для помещения в директорию /etc/rc.d/init.d стал поставляться среди дополнительных программ (см.выше). Объяснить сиё можно только тем, что Snort может запускаться на различных платформах, а поддерживается он многими (см. таблицу 1, в которой запуск реализуется по-разному).

Один из возможных способов автоматического запуска представлен ниже. Для Linux следует создать файл /etc/rc.d/init.d/snortd следующего содержания:

```
#!/bin/bash
#
# snortd          Start/Stop the snort IDS daemon.
#
# chkconfig: 2345 79 11
# description:    snort is a lightweight network intrusion
#                 detection tool that currently detects more
#                 than 1100 host and network vulnerabilities,
#                 portscans, backdoors, and more.
#
# June 10, 2000 -- Dave Wreski <dave@linuxsecurity.com>
#   - initial version
#
# July 08, 2000 Dave Wreski <dave@guardiandigital.com>
#   - added snort user/group
#   - support for 1.6.2

# Source function library.
. /etc/rc.d/init.d/functions

# Specify your network interface here
INTERFACE=eth0
# See how we were called.
case "$1" in
  start)
        echo -n "Starting snort: "
        # ifconfig eth0 up
        daemon /usr/local/bin/snort -o -i $INTERFACE -d -D \
                -c /etc/snort/snort.conf
        touch /var/lock/subsys/snort
        sleep 3
        if [ -f /var/log/snort/alert ]
         then
         rm /var/log/snort/alert
         fi
        echo
        ;;
  stop)
        echo -n "Stopping snort: "
        killproc snort
        rm -f /var/lock/subsys/snort
        echo
        ;;
  restart)
        $0 stop
        $0 start
        ;;
  status)
        status snort
        ;;
  *)
        echo "Usage: $0 {start|stop|restart|status}"
        exit 1
esac

exit 0
```

Далее следует дать команду:

```
# chkconfig snortd on
```

чтобы в директориях соответствующих уровней появились символические ссылки на этот файл.

Закомментированная строчка «ifconfig eth0 up» имеет следующий смысл: если при каких-то условиях (смена номера, запуска или какой-то сбой при запуске network) у нас Snort будет запускаться до запуска сети и поднятия интерфейса, то при её наличии ошибки не будет. Повторное поднятие уже поднятого интерфейса ошибок давать не должно. Строчку «rm /var/log/snort/alert» и несколько соседних можно закомментировать, если вы не хотите, чтобы у вас при запуске этот файл начинался с нуля.

В заключение хотел бы привести таблицу возможных платформ, на которых может работать Snort.

Таблица 1. Платформы, на которых работает Snort

i386	Sparc	M68k/PPC	Alpha	Other	
X	X	X	X	X	Linux
X	X	X			OpenBSD
X			X		FreeBSD
X		X			NetBSD
X	X				Solaris
	X				SunOS 4.1.X
				X	HP-UX
				X	AIX
				X	IRIX
			X		Tru64
		X			MacOS X Server
X					Win32 - (Win9x/NT/2000)

Планы на будущее

Если эта статья вам понравилась и вы заинтересовались COA Snort, значит, статья удалась. В дальнейшем я планирую продолжить тему обнаружения атак и написать про введение логов в БД, просмотр БД с помощью ACID, возможно, рассказав ещё про Snortcenter и другие вещи. Далее я планирую переключить внимание на обнаружение распределённых телекоммуникационных атак с помощью нескольких сенсоров Snort.

В эксперименте по установке и настройке IDS Snort разных версий принимал участие RedHat Linux v.7.3.

Литература:

1. Статистика CERT http://www.cert.org/stats/cert_stats.html
2. С.Яремчук. L.I.D.S. //Системный администратор, №4(5), 2003.
3. В.Мешков. Система крипрографической защиты информации //Системный администратор, №4(5), 2003.
4. С.Яремчук. Контрольная сумма на защите Linux/FreeBSD //Системный администратор, №6(7), 2003.
5. С.Яремчук. SELinux //Системный администратор, №5(6), 2003.
6. А.Даниленко. Технологии протоколирования Honeypot в обеспечении безопасности сетевых Unix-систем //Системный администратор, №5(6), 2003.

7. В.Мешков. Брандмауэр //Системный администратор, №1(2), 2003.
8. В.Мешков Брандмауэр ч.2. //Системный администратор, №2(3), 2003.
9. Лукацкий А.В., Обнаружение атак. – СПб: БХВ-Петербург, 2001. – 624 с.
10. Милославская Н.Г., Толстой А.И. Инстрасети: обнаружение вторжений: Учеб. пособие для вузов. – М: ЮНИТИ-ДАНА, 2001.
11. Н.Польман, Т.Краверс. Архитектура брандмауэров для сетей предприятия.: Пер. с англ. – М.: Издательский дом «Вильямс», 2003.
12. С.Норткат, М.Купер и др. Анализ типовых нарушений безопасности в сетях.: Пер. с англ. – М.: Издательский дом «Вильямс», 2001.
13. П.Н.Девянин, О.О.Михальский, Д.И.Правиков, А.Ю.Щербаков. Теоретические основы компьютерной безопасности: Учеб. пособие для вузов. – М: Радио и связь, 2000.
14. Анонимный автор, Максимальная безопасность в Linux.: Пер. с англ./Автор анонимный – К.: Издательство «ДиаСофт», 2000.
15. Сайт системы обнаружения атак IDS «Snort» http://www.snort.org/
16. А.Потёмкин. Общий обзор наиболее часто применяемых техник компьютерных атак и защиты от них //Системный администратор, №1(2), 2003.
17. Форум журнала «Системный администратор» http://www.samag.ru/cgi-bin/yabb/YaBB.pl
18. Мак-Клар, Стюарт, Скембрей, Джоел, Курц, Джордж. Секреты хакеров. Безопасность сетей – готовые решения, 2-е изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2001.
19. Сайт компании Internet Security Systems, Inc. http://www.iss.net/
20. Правиков Д.И. Закляков П.В. Использование виртуальных ловушек для обнаружения телекоммуникационных атак. //Проблемы управления безопасностью сложных систем: Труды международной конференции. Москва, декабрь 2002 г./ Под ред. Н.И.Архиповой и В.В.Кульбы. Часть 1. – М.: РГГУ – Издательский дом МПА-Пресс. 342 с., с 310-314;
21. Роберт Л. Зиглер., Брандмауэры в Linux. : Пер. с англ.: Уч.пос. – М.: Издательский дом «Вильямс», 2000. – 386 с.
22. Request For Comments ? 793
23. Request For Comments ? 3168
24. Отчёт фирмы Symantec по угрозам безопасности в Интернете. http://www.symantec.com/region/ru/ruresc/download/SymantecInternetSecurityThreatReport2003.pdf
25. Руководящий документ Государственной Технической Комиссии России при Президенте РФ «Средства вычислительной техники. Межсетевые экраны»
26. П.Закляков «Разводной мост на Linux» //Системный администратор, №4(5), 2003.
27. Козлов Д.А. Энциклопедия компьютерных вирусов.- М.: «Солон», 2001. – 464 с.
28. Большая вирусная энциклопедия http://www.viruslist.com/index.html
29. С.Манн, М.Крелл Linux. Администрирование сетей TCP/IP. Пер. с англ. – М.:ООО «Бином-Пресс», 2003.
30. В.Г.Олифер, Н.А.Олифер. Компьютерные сети. Принципы, технологии, протоколы – СПб: Питер, 2001.
31. Сайт SANS Institute – Computer Security Education and Information Security Training http://www.sans.org/
32. Файл NEWS из Snort-2.0.2.tar.gz
33. Дж.Такет, С. Барнет. Использование Linux. Специальное издание.: 5-е изд.:Пер с англ.: Уч. пос. – М.: Издательский дом «Вильямс», 2000.
34. Медведковский И.Д., Семьянов П.В., Платонов В.В. Атака через INTERNET. Под научной редакцией проф. Зегжды П.Д. - СПб: «Мир и семья-95», 1997.