

Snort для блокирования сайтов

PCAP и все-все-все

В качестве решения для блокировки сайтов были испробованы SCE8k URL filtering, SKAT и Snort.

Возможно, я не умею готовить URL-DB, но фильтрация средствами SCE давала сбои, к тому же, мы начали вылезать за пределы ее возможностей (обеих, имеющихся). Я решил попробовать Snort и получил эффективность выше, чем у SCE. Параллельно брали на тест [СКАТ](#), рассматривали как замену SCE по нарезке скорости. Со SKATом все вообще шикарно, однако по функционалу нарезки скорости он сильно неудобнее, чем SCE. Поэтому пока отложили.

Итак, возвращаемся к хрюшке. Snort – это IDS/IPS (в том числе inline), которая позволяет достаточно гибко настроить свои правила пакетной обработки. Для блокировки с редиректом клиента, можно использовать функционал Active Response (включается через ./configure, читать README.active, в пакетах для FreeBSD включен по умолчанию). Для мониторинга можно использовать preprocessor perfmonitord для наблюдения за качеством/эффективностью.

В определенных трудах в интернете утверждается, что с libpcap старых версий позволяет выдавить 200-500Мбит/с через фильтр. Но мой опыт говорит, что бутылочных горлышек больше чем одно, и идут они последовательно.

Первой печалью являются интерфейсы захвата пакета. Т.к. нам хочется не просто анализа, а еще и активного воздействия на трафик, то собирать нам придется конструкцию в inline-режиме.

Итерация первая — фильтрующий маршрутизатор.

Собираем конструкцию из BSD, поверх которой надеваем IPFW/ IPDIVERT / Quagga / SNORT и заправляем это все совершеннейшим безобразием на "питоне".

SNORT – запускается в режиме работы с DAQ: ipfw, слушает на 8000-м сокете пакеты от фаервола и возвращает их туда же. Правила берутся из файла rules/local.rules, который генерируется скриптом на "питоне".

Quagga - запускается единственный демон из пакета — bgpd на серой AS-ке, анонсирует префиксы (в основном - /32) для перехвата и анализа, все согласно рекомендаций надзора. Префиксы получает через телнет от того же "питона". "Квагга" была просто знакома, но также этот пакет не требует присутствия маршрута в таблице маршрутизации ядра для того, чтобы анонсировать его в сеть. Достаточно внести префикс в конфиг.

IPDIVERT – модуль ядра для добавления divert-сокетов, в конфиг ядра засовывать не надо, можно подгрузить в любой момент и начать пользоваться. Нужен для Inline-режима Snort-a.

IPFW – собственно, фаервол, изначально для облегчения жизни хрюшке, блокировал по таблице доступ по IP(HTTPS/Blocktype:IP/Domain), засовывал пакеты в divert-сокет, ловил их обратно и через fwd отправлял дальше по маршруту.

Итак, со всем этим мы пытаемся взлететь. Взлетаем, берегов не видим, но видим, что Ревизор регистрирует 600-800 неблокированных ссылок. Разбираемся и понимаем, что там большая часть — казино, скачущие по IP, как блохи по дворянке. Намазываем на "питон" сверху Libadns, негодуем, но делать нечего. В итоге, анонсируется чуть больше адресов, хороших и разных, время работы скрипта, генерирующего конфигов, вырастает с ~3 минут до 20 минут, число неблокированных сайтов падает до 100-120 штук.

Большую печаль вызывает при этом Google, равно как Amazon. На их облаках очень много разного и интересного, каждый раз даже ADNS возвращает разные наборы адресов. Добавляем префиксы "гугла" и AWS целиком в анонсы, уходим в штопор. До "ютуба" потери, отзывчивость AJAX-а потерялась. Два Snort-a в параллель не сильно помогают решению вопроса.

Первый кандидат на оптимизацию - интерфейс захвата пакетов. Было понимание, что путешествие пакета от интерфейса через ядро в юзерспейс и потом обратно, не может быть сильно быстрым. Поэтому, находим решение в виде пакета Netmap.

Netmap

Луиджи Риццо решил проблему быстрой обработки пакетов в юзерспейсе весьма элегантно. На момент, когда он взялся за решение, из вариантов были — пихать всю обработку в ядро,

реализовывать прямой доступ к оборудованию, или плакать в уголке. Экзотику в виде спец железа с FPGA не рассматриваем, на эти деньги можно собрать десяток тазиков, одной рукой их настраивать, другой — слезы счастья вытирать...

Обработка пакетов в ядре хороша тем, что нет переключения контекста, а плохо тем, что неудобно настраивать и любой косяк приводит к краху всей системы. Прямой доступ к оборудованию хорош тем, что получив в юзерспейсе железо, мы можем работать с ним напрямую, но это опять таки опасно тем, что от любого косяка мы теряем доступ к этому железу (а то и снова панику ловим). Луиджи решил эту проблему просто. Небольшая модификация драйвера/или эмуляция драйвера, и вот сетевая карта пишет уже в область shared memory, доступную и драйверу, и приложению пользователя. При этом, вполне можно реализовать транзит пакетов без копирования, и тратить при этом наносекунды на пакет. Такой подход также позволил генерировать или ловить до 14.8 миллионов пакетов в секунду одним ядром на частоте 900 МГц. Это внушает определенный оптимизм.

После обнаружения такого подарка судьбы, смотрим, как у нас собран DAQ, обнаруживаем, что поддержка netmap есть по умолчанию. Читаем, что и как запускать. Приходим ко второй итерации — фильтрующий мост. В режиме netmap у нас Snort является мостом, а хост-стек не участвует в форвардинге пакетов совсем. Это приводит нас к тому, что фильтрацию по IP придется отдать хрюшке, а значит — мы получим на несколько тысяч правил больше. Ну с 33 тысячами правил оно взлетело, значит и с 38 тысячами взлетит, решил я, и легким движением руки переделал.

Чтобы не разглагольствовать впустую, скажу сразу — взлетело, но полетело низко, примерно как спидгонщики по проселку. В ходе разборок задействовал два Dell R710 для стенда. Это крайне неэффективное решение с точки зрения энергопотребления и разбрасывания ресурсов. Однако, они были под рукой и мощности их хватило. На плате у этих серверов 4 бродкома, которые не имеют модифицированных драйверов netmap, они запускаются в режиме эмуляции. В этом режиме они имеют одну очередь на прием, одну очередь на отправку и кольцевой буфер регулируемой длины. Один из серверов был задействован как pkt-gen на прием/передачу. Он оказался способен генерировать от 700 до 980 тысяч пакетов в секунду размером 64 байта. С ростом размера пакета утилизация интерфейса в мегабитах в секунду доходит до 100%. В голом мостинге такой же сервер вполне способен с zegosuru гонять все пакеты без потерь и ошибок.

Snort/netmap запускается полностью, также как и другой вариант, меняется только в строке запуска имя интерфейса и название DAQ-модуля. С точки зрения ОС, в случае этого сервера большой разницы нет, будет Netmap собран модулем, или встроен в ядро. Для сервера с igb/em/ixl/ix разница, возможно, есть, т.к. при перекомпиляции ядра с device netmap также пересобируются драйвера с поддержкой этого фреймворка. Это даст возможность задействовать несколько очередей карты для многопоточной обработки пакетов. В случае Snort2 это не имеет особого смысла, но иметь в виду это стоит.

Итак, мы взлетели, но задевали любые выступа на поверхности. Что не так?

Оказывается, пока мы медленно и печально таскали пакеты из ядра в юзерспейс, нам было абсолютно по барабану, что пакет гоняется по 33 тысячам правил за 4.5 мс. В особо благоприятных случаях — 1.7 мс на пакет. Как только мы смогли доставить 700K пакетов в секунду снорту — он сдох. На наборе из 30K+ правил он умудрялся обрабатывать не больше 1023 пакетов в секунду, все остальное терялось. Если же запускать без правил вообще, то производительность именно моста там вполне достаточная, пакеты не теряются.

Правила и оптимизация работы фильтра

Поначалу, в ходе разборок на стенде я получал совершенно удручающие результаты. В моем представлении у этой системы производительность должна быть пакетной. Есть набор правил, по которому гоняется пакет, в зависимости от размера. Этим набором правил должна определяться производительность в пакетах в секунду. Если мы тратим 4.5 мс на пакет в Snort, то потолок у нас получится жалкие 222 пакета в секунду, т. к. система у нас однопоточная. На самом деле, все немножко лучше, и мы можем наблюдать производительность на высоком уровне. Несмотря на то, что средний размер пакета болтается от 70 до 150 байт.

Что можно оптимизировать?

1. Алгоритм pattern-matcher.

Есть несколько на выбор. Поскольку нам нужна быстрая обработка каждого пакета, то берем эффективный по CPU алгоритм AC, вместо алгоритма ac-bnfa-nq, который эффективно использует память, но при этом жрет процессор как не в себя.

2. Структура правил.

Это еще предстоит попробовать, но теоретически, правило должно быть максимально детерминированным, например:

drop any any → 192.168.0.0/24 \$HTTP_PORTS (dsize:>78; content:"casino.org"; fast_pattern; content: "/ruleтка.html"; nocase; rev:1; sid: 1000001; react;)

В этом правиле хорошо практически все. Я пока не уверен, что адрес получателя улучшит показатели (по идее должен, т. к. не задействует pattern-matcher, использует строгое сравнение). Однозначно важно иметь **dsize: > 78**, первым пунктом правила. Это опять таки строгая проверка: для пакета длиной 68 байт это правило будет отброшено оптимизатором сразу же. Почему 78? это минимальный размер сферического HTTP-GET в вакууме.

Дальше идет правило поиска по контенту, с модификатором fast_pattern, что опять-таки уменьшает количество правил, по которым пройдет пакет. Второе правило поиска по контенту уже работает с модификатором nocase, что дает регистронезависимый поиск. Возможно, стоит еще добавить regex, но пока не вижу в этом большой необходимости. Последняя инструкция react заставит Snort не просто сделать drop, но еще и отправить TCP RST серверу, а клиенту отправить пакет с содержимым, которое указано в спец. файле и добить сессию RST-пакетом клиенту. С реакцией опять возникли нюансы. Модуль включен по умолчанию в пакетах FreeBSD, однако последняя версия из пакетов отказывалась слать ответы. При этом, версия 2.9.5, собранная из исходников, шлет ответы просто изумительно.

3. Количество правил.

Можно иметь большой список правил, но путем различных ухищрений сокращать их число до привлечения pattern-matcher. Однако, на наборе в 30+ тысяч это работает слабо. Если даже сокращение списка правил и происходит, то сокращается он не сильно. В различных трудах, посвященных производительности Snort, авторы крутят пяткой у виска глядя на список из менее чем 10K правил. Их должно быть еще меньше.

Чисто для сравнения. Фильтруем список правил, оставляем только youtube, получаем 350 правил. На таком наборе Snort способен гонять все 500-600 Кппс без потерь.

Начинаем бомбардировать Snort пакетами по 500 Байт (максимум 260Кппс в гигабите), смотрим сколько пакетов в секунду пролазит на разном количестве правил. Получаем, что до 1000 правил — пролазит все. При 1000-10000 правилах начинаются потери. А к 10K правил дропы доходят до 50%. С большим же количеством правил, все еще хуже.

Также по опыту замечено, что падает Snort очень редко, только если сожрет всю память и своп - тогда его прибьет ОС. Также, с малым набором правил перезапуск "хрюшки" будет очень быстрым.

Определенный оптимизм есть по поводу Snort3, в котором пообещали устранить фатальный недостаток — однопоточность. Также (если смотреть малый размер списка правил), обещали в следующих версиях DAQ реализовать доступ к разным очередям сетевой карты в режиме netmap, что вместе с многопоточностью матчинга даст существенный прирост производительности.