

Как использовать возможности фильтров отображения Wireshark

Журнал «Хакер» Ноя 5, 2013

Для исследования поведения сетевых приложений и узлов, а также чтобы выявить неполадки в работе сети часто прибегают к анализаторам сетевых пакетов. Ключевые особенности подобного ПО — это, во-первых, возможности разносторонней аналитики, а во-вторых, многофункциональная фильтрация пакетов, позволяющая выудить крупницы интересующей информации в безбрежном потоке сетевого трафика. Именно последнему аспекту и посвящена эта статья.

Введение

Из всех методов изучения компьютерных сетей анализ трафика, пожалуй, самый кропотливый и трудоемкий. Интенсивные потоки современных сетей порождают очень много «сырого» материала, отыскать в котором крупницы полезной информации далеко не просто. За время своего существования стек TCP/IP оброс многочисленными приложениями и дополнениями, счет которым идет на сотни и тысячи. Это прикладные и служебные протоколы, протоколы аутентификации, туннелирования, доступа к сети и так далее. Кроме знания азов сетевых взаимодействий, исследователю трафика (то есть тебе) нужно свободно ориентироваться во всем этом протокольном многообразии и уметь работать со специфичными программными инструментами — снифферами, или, по-научному, анализаторами трафика (протоколов).

Функциональность сниффера — это не только возможность использования «неразборчивого» (promiscuous) режима работы сетевой карты для перехвата. Подобный софт должен уметь эффективно фильтровать трафик как на этапе сбора, так и во время изучения отдельных единиц передачи (фреймов, пакетов, сегментов, датаграмм, сообщений). Причем чем больше протоколов сниффер «знает», тем лучше.

Современные анализаторы протоколов много чего умеют: считать статистику трафика, рисовать графики хода сетевых взаимодействий, извлекать данные прикладных протоколов, экспортировать результаты работы в различные форматы... Поэтому подбор инструментария для анализа сетевого трафика — это тема для отдельного разговора. Если ты не знаешь, что выбрать, или же не хочешь тратить деньги на платное ПО, то воспользуйся простым советом: установи Wireshark.

Знакомимся с фильтрами

Wireshark поддерживает два вида фильтров:

- Фильтры перехвата трафика (capture filters);
- Фильтры отображения (display filters).

Первая подсистема досталась Wireshark в наследство от библиотеки Pcap, обеспечивающей низкоуровневый API для работы с сетевыми интерфейсами. Выборка трафика на лету во время перехвата позволяет экономить оперативную память и место на жестком диске. Фильтр представляет собой выражение, состоящее из группы примитивов, при необходимости объединенных логическими функциями (and, or, not). Записывается это выражение в поле

«Capture Filter» диалогового окна «Capture options». Наиболее употребляемые фильтры можно сохранять в профиле для повторного использования (рис. 1).

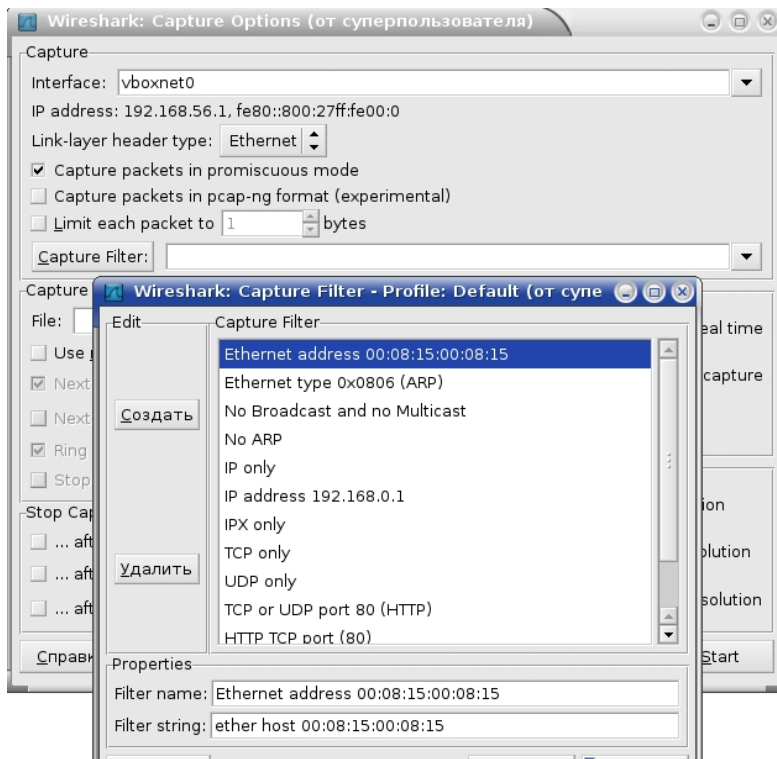


Рис. 1. Профиль фильтров перехвата

Язык фильтров перехвата стандартный для мира Open Source и используется многими Pcap-основанными продуктами (например, утилитой tcpdump или системой обнаружения/предотвращения вторжений Snort). Поэтому описывать синтаксис здесь нет особого смысла, так как он тебе, скорее всего, знаком. А детали можно посмотреть в документации, например в Linux на странице справочного руководства pcap-filter(7).

Фильтры отображения работают с уже перехваченным трафиком и являются «родными» для Wireshark. Отличия от Pcap — в формате записи (в частности, в качестве разделителя полей используется точка); также добавлены английская нотация в операциях сравнения и поддержка подстрок.

Вписать фильтр отображения можно прямо в соответствующее поле (внимание, работает выпадающий список-подсказка) главного окна программы после кнопки «Filter» (кстати, под этой кнопкой скрывается профиль для часто используемых выражений). А если кликнуть расположенную неподалеку кнопку «Expression...», то откроется многофункциональный конструктор выражений (рис. 2).

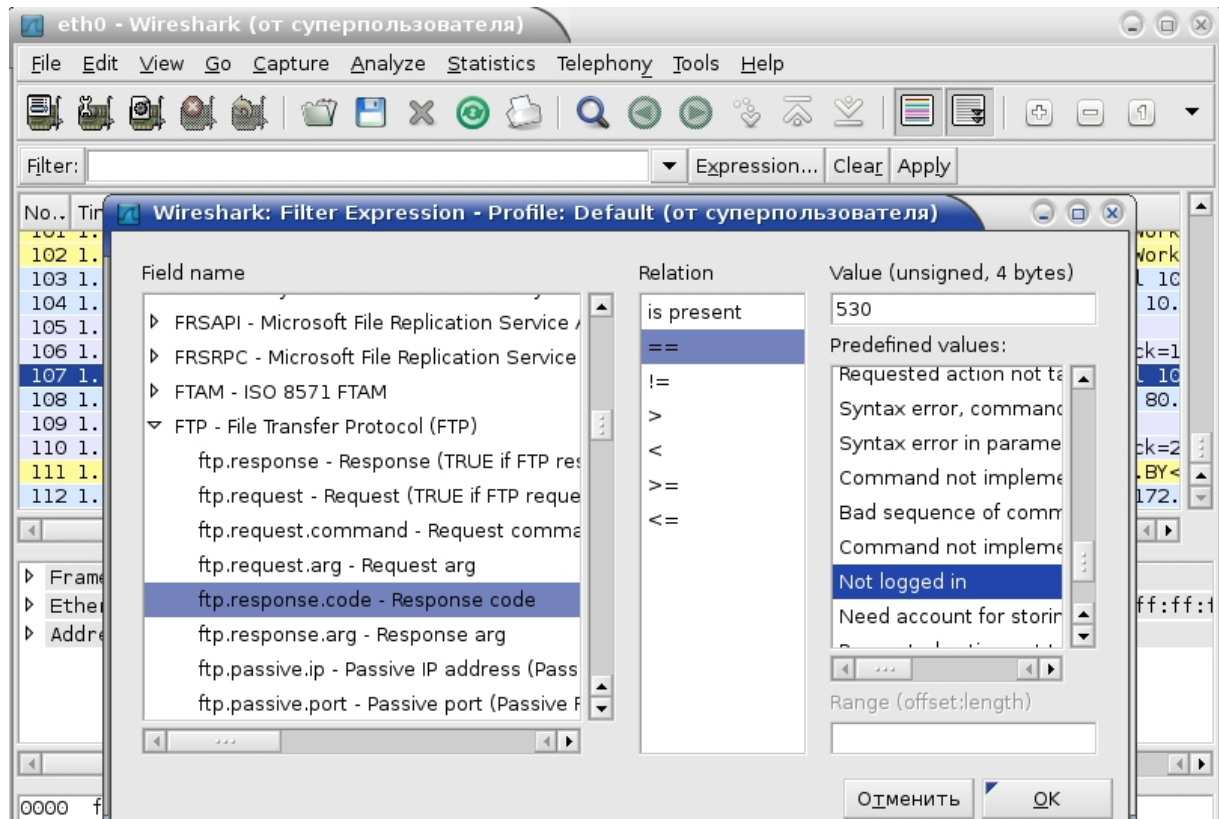


Рис. 2. Конструктор фильтров отображения

Слева (Field Name) представлено упорядоченное по алфавиту дерево полей сообщений протоколов, которые известны Wireshark. Для данного поля можно указать логический оператор (Relation), вписать значение (Value), указать диапазон (Range) или выбрать значение из списка (Predefined Value). В общем, полная сетевая энциклопедия в одном окошке.

Вот логические операторы, используемые в фильтрах отображения:

- `and (&&)` — «И»;
- `or (||)` — «ИЛИ»;
- `xor (^)` — исключающее «ИЛИ»;
- `not (!)` — отрицание;
- `[...]` — выборка подстроки. # Фильтруя по MAC-адресу своего сетевого адаптера, исключаем весь локальный трафик
 - `not (eth.addr eq aa:bb:cc:22:33:44)`
- # Отмечаем весь «служебный шум», чтобы сконцентрироваться на интересующем нас трафике
 - `!(arp or icmp or dns)`
-

Что касается выборки подстроки, то это не совсем логическая операция, но весьма полезная опция. Она позволяет получить определенную часть последовательности. Например, так можно использовать в выражении первые (первое число в квадратных скобках — смещение) три байта (число после двоеточия — длина подпоследовательности) поля MAC-адреса источника:

```
eth.src[0:3] == 00:19:5b
```

В выборках с двоеточием один из параметров можно опускать. Если пропустить смещение, то отсчет выборки начнется с нулевого байта. Если длину — то получим все байты от смещения до конца поля.

К слову, выборку подстроки удобно использовать для выявления малвари в случае, если известна последовательность байт, идущая после заголовка (например, «0x90, 0x90, 0x90, 0x04» в UDP-пакете):

```
udp[8:4] == 90:90:90:04
```

Операции сравнения, используемые в логических выражениях:

- eq (==) — равно;
 - ne (!=) — не равно;
 - gt (>) — больше;
 - lt (<) — меньше;
 - ge (>=) — больше или равно;
 - le (<=) — меньше или равно.
- ```
tcp.dstport ne 8080 && tcp.len gt 0 && data[0] eq A0
```

Собственно, теории для начала достаточно. Дальше используй здравый смысл и скобки по необходимости и без нее. Также не забывай, что фильтр по сути — логическое выражение: если оно истинно, то пакет отобразится на экране, если ложно — нет.

## Рсар-фильтр для выявления сканирования Netbios-портов

```
dst port 135 or dst port 445 or dst port 1433 and tcp[tcpflags] & (tcp-syn) != 0 and tcp[tcpflags] & (tcp-ack) = 0 and src net 192.168.56.0/24
```

## Ищем угонщика IP-адреса

В сегменте локальной сети случаются (по тем или иным причинам) совпадения IP-адресов у двух и более узлов. Методика «отлова» (определения MAC-адресов) конфликтующих систем общеизвестна: запускаем на третьем компьютере сниффер, чистим ARP-кеш и стимулируем запрос на разрешение MAC'а искомого IP, например пропинговав его:

```
arp -d 192.168.56.5
```

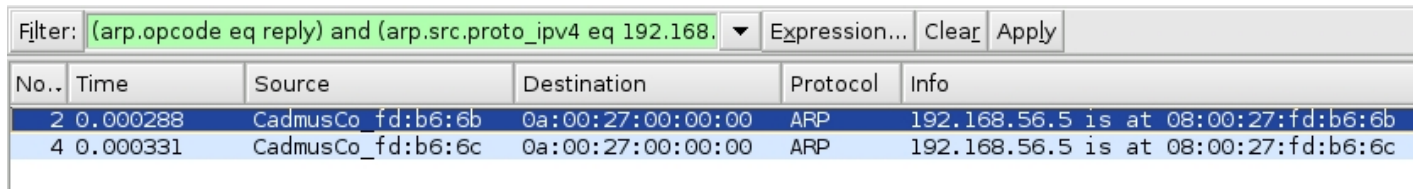
```
ping -n -c 1 192.168.56.5
```

А потом ищем в перехваченном трафике, с каких MAC'ов пришли ответы. Если Wireshark наловил слишком много пакетов, создаем фильтр отображения с помощью конструктора. В первой части выражения выбираем ARP-ответы, во второй — те сообщения, в которых исходный IP-адрес равен искомому. Примитивы объединяем оператором &&, так как нужно, чтобы оба условия выполнялись одновременно:



```
(arp.opcode == reply) && (arp.src.proto_ipv4 == 192.168.56.5)
```

Кстати, при выполнении этого сценария ни одна компьютерная сеть не пострадала, потому что были использованы две виртуальные машины Oracle VirtualBox и сетевое подключение типа «Виртуальный адаптер хоста».



| No. | Time     | Source            | Destination       | Protocol | Info                                 |
|-----|----------|-------------------|-------------------|----------|--------------------------------------|
| 2   | 0.000288 | CadmusCo fd:b6:6b | 0a:00:27:00:00:00 | ARP      | 192.168.56.5 is at 08:00:27:fd:b6:6b |
| 4   | 0.000331 | CadmusCo fd:b6:6c | 0a:00:27:00:00:00 | ARP      | 192.168.56.5 is at 08:00:27:fd:b6:6c |

Рис. 3. Результат работы фильтра протокола ARP

## Инспектируем сетевой и транспортный уровни

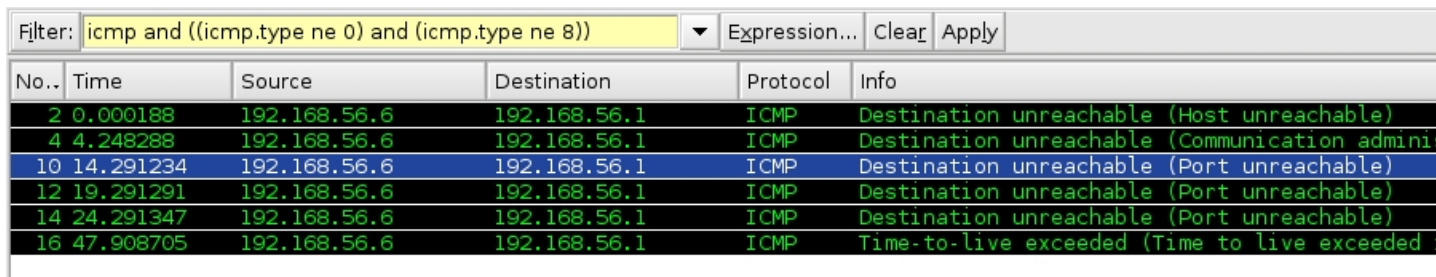
До сих пор достаточно эффективным средством диагностики сетевого стека остается протокол ICMP. Из сообщений этого протокола можно получить ценную информацию о проблемах в сети.

Как ты уже догадался, отфильтровать ICMP в Wireshark очень просто. Достаточно в строке фильтрации в главном окне программы написать: icmp. Кроме icmp, работают и многие другие ключевые слова, являющиеся именами протоколов, например arp, ip, tcp, udp, snmp, smb, http, ftp, ssh и другие.

Если ICMP-трафика много, то отображение можно детализировать, исключив, например, эхо-запросы (тип 0) и эхо-ответы (тип 8):

```
icmp and ((icmp.type ne 0) and (icmp.type ne 8))
```

На рис. 4 показан пример небольшой выборки ICMP-сообщений, созданных тестовым Linux-маршрутизатором. Сообщение «Port Unreachable» обычно используется по умолчанию. Оно же генерируется сетевым стеком при получении UDP-датаграммы на неиспользуемый порт. Чтобы виртуальный роутер на основе Debian начал отдавать сообщения «Host unreachable» и «Communication administratively filtered», пришлось с ним повозиться. Cisco же информирует об административной фильтрации обычно по умолчанию. Сообщение «Time-to-live exceeded» говорит о наличии петли на каком-то участке сети (ну и при трассировке маршрута такие пакеты также могут появляться).



| No.. | Time      | Source       | Destination  | Protocol | Info                                          |
|------|-----------|--------------|--------------|----------|-----------------------------------------------|
| 2    | 0.000188  | 192.168.56.6 | 192.168.56.1 | ICMP     | Destination unreachable (Host unreachable)    |
| 4    | 4.248288  | 192.168.56.6 | 192.168.56.1 | ICMP     | Destination unreachable (Communication admini |
| 10   | 14.291234 | 192.168.56.6 | 192.168.56.1 | ICMP     | Destination unreachable (Port unreachable)    |
| 12   | 19.291291 | 192.168.56.6 | 192.168.56.1 | ICMP     | Destination unreachable (Port unreachable)    |
| 14   | 24.291347 | 192.168.56.6 | 192.168.56.1 | ICMP     | Destination unreachable (Port unreachable)    |
| 16   | 47.908705 | 192.168.56.6 | 192.168.56.1 | ICMP     | Time-to-live exceeded (Time to live exceeded  |

Рис. 4. Некоторые ICMP-сообщения

Кстати, о межсетевых экранах. Создавать правила для популярных файеров можно прямо в Wireshark, используя пункт «Firewall ACL Rules» меню «Tools». Предварительно нужно выбрать в списке пакет, информация которого будет использована. Доступны стандартные и расширенные ACL Cisco, правила UNIX-like продуктов IP Filter, IPFirewall (ipfw), Netfilter (iptables), Packet Filter (pf), а также Windows Firewall (netsh).

И теперь кратко об азах фильтрации на сетевом уровне, основу которой составляют поля заголовка IP-пакета — адрес отправителя (ip.src) и адрес получателя (ip.dst):

```
(ip.src == 192.168.56.6) || (ip.dst == 192.168.56.6)
```

Так мы увидим все пакеты, которые получил или отправил данный IP-адрес. Фильтровать целые подсети можно, используя CIDR-нотацию записи маски. Для примера выявим инфицированный хост, осуществляющий спам-рассылку (здесь 192.168.56.251 — это IP-адрес нашего SMTP-сервера):

```
ip.src == 192.168.56.0/24 and tcp.dstport == 25 and !(ip.dst == 192.168.56.251)
```

К слову, для выборки по MAC-адресам следует использовать примитивы eth.src, eth.dst и eth.addr. Порой проблемы сетевого уровня куда теснее связаны с Ethernet-уровнем, чем об этом повествует теория. В частности, при настройке маршрутизации очень полезно бывает посмотреть, на MAC-адрес какого роутера упрямый узел отправляет пакеты. Впрочем, для такой простой задачи за глаза хватит утилиты tcpdump, практически штатной для UNIX-подобных систем.

С фильтрацией портов у Wireshark тоже никаких вопросов нет. Для TCP к твоим услугам ключевые слова tcp.srcport, tcp.dstport и tcp.port, для UDP — udp.srcport, udp.dstport и udp.port. Правда, у встроенного языка фильтров Wireshark не нашлось аналога примитива port в Pcap, обозначающего как порт UDP, так и TCP. Но это легко исправить с помощью логического выражения, например:

```
tcp.port == 53 || udp.port == 53
```

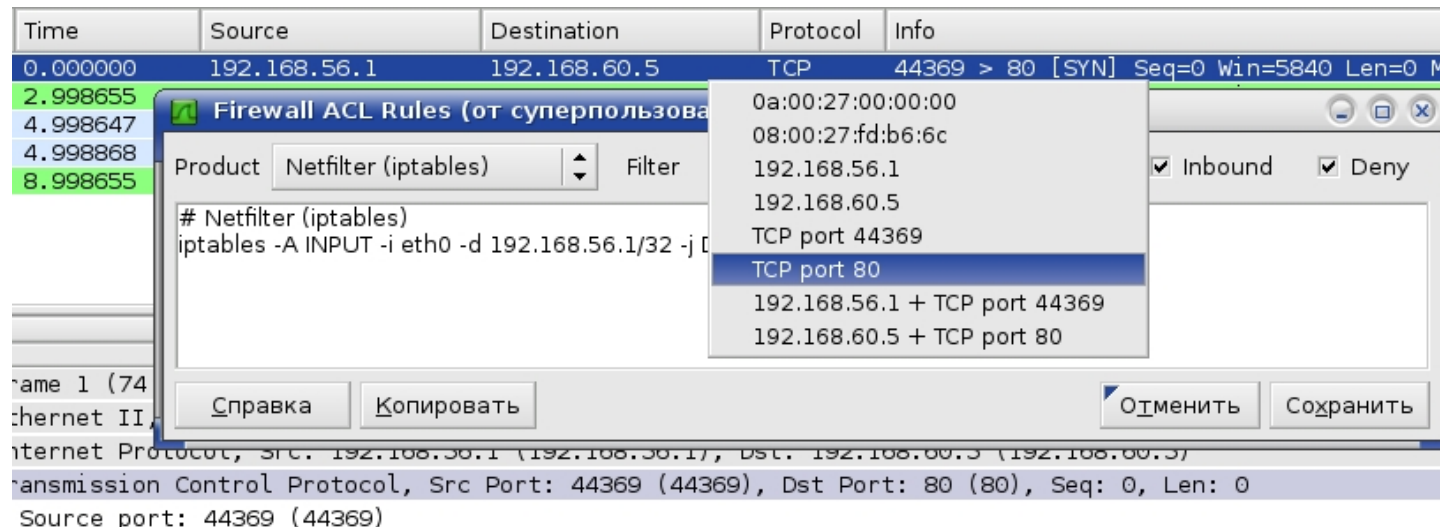


Рис. 5. Создание правил межсетевого экрана

## Импровизируем с HTTP-трафиком

Прикладные протоколы, в частности HTTP, — это «вечная» тема в разрезе sniffing. Справедливости ради нужно сказать, что для исследования веб-трафика создано немало специализированных программных средств. Но и такой универсальный инструмент, как Wireshark, с его гибкой системой фильтрации на этом поприще оказывается совсем не лишним.

Для начала соберем немного веб-трафика, сходяв на первый пришедший на ум сайт. Теперь поищем в сообщениях протокола TCP, служащего транспортом для HTTP, упоминания любимого интернет-ресурса:

```
tcp contains "xakep.ru"
```

Оператор `contains` проверяет наличие подстроки в данном поле. Есть еще оператор `matches`, в нем можно использовать Perl-совместимые регулярные выражения.

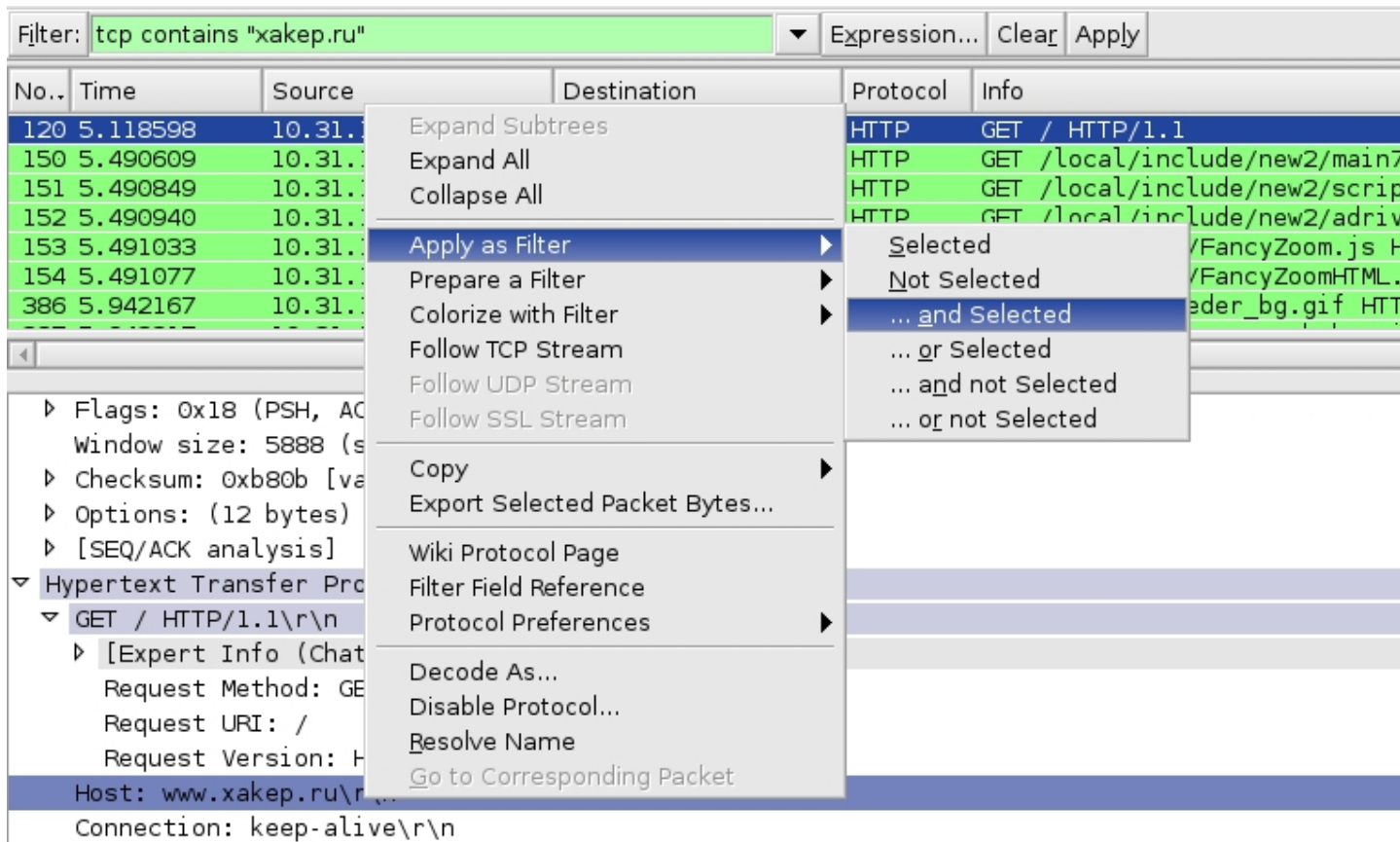


Рис. 6. Модификация фильтра отображения на лету

Окошко «Filter Expressions», конечно, хороший помощник, но порой перелистывать длинный список в поисках нужного поля весьма утомительно. Есть более простой способ создания/модификации фильтров отображения: с помощью контекстного меню при просмотре пакетов. Для этого нужно просто кликнуть правой клавишей мыши по интересующему полю и выбрать один из подпунктов пункта «Apply As Filter» или пункта «Prepare a Filter». В первом случае изменения тут же вступят в силу, а во втором — можно будет подкорректировать выражение. «Selected» означает, что значение поля станет новым фильтром, «Not Selected» — то же самое, только с отрицанием. Пункты, начинающиеся с «...», добавляют значение поля к существующему выражению с учетом логических операторов.

Комбинируя различные средства графического интерфейса Wireshark и знание особенностей протокола HTTP, можно легко детализировать до требуемого уровня отображение трафика в главном окне программы.

Например, чтобы посмотреть, какие изображения браузер запрашивал у веб-сервера при формировании страницы, сгодится фильтр, анализирующий содержимое передаваемого серверу URI:

```
(http.host eq "www.xakep.ru") and ((http.request.uri contains ".jpg") or (http.request.uri contains ".png"))
```

То же самое, но с использованием matches:

```
(http.host eq "www.xakep.ru") and (http.request.uri matches ".jpg|.png")
```

Разумеется, поля сообщений протоколов разных уровней можно смело смешивать в одном выражении. Например, чтобы узнать, какие картинки данный сервер передал клиенту, используем исходный адрес из IP-пакета и поле «Content-Type» HTTP-ответа:

```
(ip.src eq 178.248.232.27) and (http.content_type contains "image")
```

А с помощью поля HTTP-запроса «Referer» ты сможешь узнать, с каких еще серверов браузер берет контент при формировании страницы любимого сайта:

```
(http.referer eq "http://www.xakep.ru/") and (not (ip.dst eq 178.248.232.27))
```

Рассмотрим еще несколько фильтров-полезняшек. Для выборки из трафика HTTP-запросов, сделанных методом GET, можно использовать следующее выражение:

```
http.request.method == GET
```

Именно на прикладном уровне фильтры отображения проявляют себя во всей красе и простоте. Для сравнения: чтобы, например, решить эту задачу с помощью Pcap, пришлось бы применить вот такую трехэтажную конструкцию:

```
port 80 and tcp[((tcp[12:1] & 0xf0) >> 2):4] = 0x47455420
```

Чтобы выяснить, какие www-подключения совершал пользователь хоста 192.168.56.8 в определенный интервал времени (скажем, в обеденный перерыв), задействуем примитив frame.time:

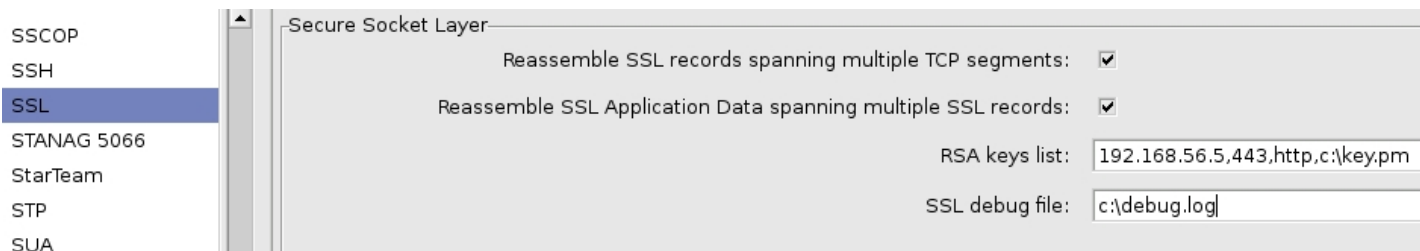


```
tcp.dstport == 80 && frame.time >= "Yan 9, 2013 13:00:00" && frame.time < "Yan 9, 2013 14:00:00"
&& ip.src == 192.168.56.8
```

Ну и отображение URI запросов, содержащих слова «login» и «user», плюс «напоминка» паролей:

```
http.request.uri matches "login.*=user"
```

```
(http contains "password") || (pop contains "PASS")
```



*Рис. 7. Настройка SSL-сертификата*

## Перехват SSL-контента

Настоящий бич исследователя сетевого трафика — шифрование. Но если у тебя есть заветный файл с сертификатом (кстати, беречь его нужно как зеницу ока), то ты легко сможешь узнать, что прячут пользователи данного ресурса в SSL-сессиях. Для этого нужно указать параметры сервера и файл сертификата в настройках протокола SSL (пункт «Preferences» меню «Edit», слева в списке протоколов выбрать SSL). Поддерживаются форматы PKCS12 и PEM. В последнем случае нужно убрать пароль с файла командами:

```
openssl pkcs12 -export -in server.pem -out aa.pfx
```

```
openssl pkcs12 -in aa.pfx -out serverNoPass.pem -nodes
```

## INFO

Извлечение трафика для мониторинга и отладки из сетевого трафика осуществляется пакетным фильтром. Пакетный фильтр входит в состав ядра операционной системы и получает сетевые пакеты от драйвера сетевой карты.

Примерами пакетных фильтров для UNIX-like ОС являются BPF (Berkeley Packet Filter) и LSF (Linux Socket Filter). В BPF фильтрация реализована на основе регистро-ориентированного примитивного машинного языка, интерпретатором которого и является BPF.

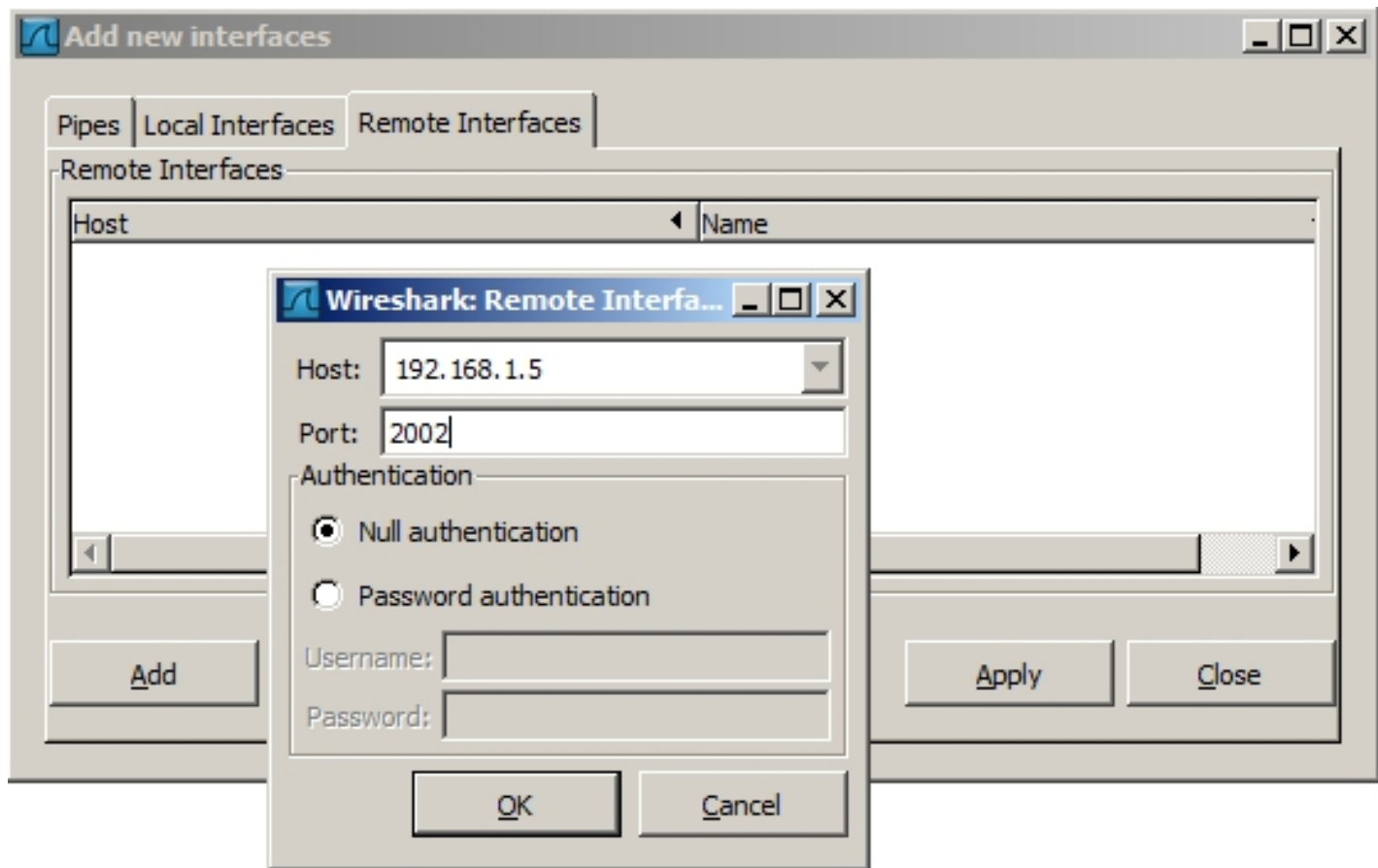


Рис. 8. Настройка удаленного перехвата в Wireshark

## Анализируем трафик с удаленных хостов

Пользователи Windows могут работать не только с интерфейсами того компьютера, на котором запущен Wireshark, но и снимать трафик с удаленных машин. Для этого существует специальная служба (Remote Packet Capture Protocol) в поставке библиотеки WinPcap. Ее нужно предварительно включить в оснастке управления службами (services.msc). Теперь, запустив Wireshark на удаленном компьютере, можно подключиться к тому узлу, на котором работает сервис удаленного перехвата трафика (по умолчанию использует порт 2002), и данные по протоколу RPCAP потекут к тебе рекой.

Также приведу варианты подключения к домашнему \*nix-роутеру «извне» для удаленного анализа трафика:

```
$ ssh root@home.gw.ip.addr 'tshark -f "port !22" -i any -w -' | wireshark -k -i -
```

```
$ ssh root@home.gw.ip.addr tcpdump -U -s0 -w - 'not port 22' | wireshark -k -i -
```

## Инструмент из разряда must have

Wireshark — широко известный инструмент перехвата и интерактивного анализа сетевого трафика, фактически стандарт для промышленности и образования. Распространяется под лицензией GNU GPLv2. Wireshark работает с большинством известных протоколов, имеет графический интерфейс пользователя на основе GTK+, мощную систему фильтров трафика и встроенный интерпретатор языка программирования Lua для создания декодеров и обработчиков событий.

## Извлечь полезный груз

В определенных кругах широко известны специализированные инструменты, позволяющие «вытаскивать» из трафика конечные информационные объекты: файлы, изображения, видео- и аудиоконтент и прочее. Благодаря мощной аналитической подсистеме, Wireshark эту функциональность с лихвой покрывает, поэтому ищи в соответствующих окнах анализа кнопку «Save Payload...».

## WWW

- Официальный сайт [Wireshark](#);
- wiki-страница, посвященная [фильтрам отображения](#);
- страница гида Wireshark, посвященная фильтрам отображения: [goo.gl](#);
- wiki-страница, посвященная фильтрам Pcap: [wiki.wireshark.org](#);
- библиотека Pcap и утилита [tcpdump](#);
- библиотека [WinPcap](#).

## Заключение

На фоне всеобщего увлечения компьютерного андеграунда вопросами безопасности сетевых приложений монументальные проблемы нижележащих уровней постепенно уходят на второй план. Понятно, что сетевой и транспортный уровни изучены и исследованы вдоль и поперек. Но беда в том, что специалисты, выросшие на SQL-инъекциях, межсайтовом скриптинге и

инклюдах, не подозревают об огромном пласте, скрытом под вершиной айсберга, и часто пасуют перед, казалось бы, элементарными проблемами.

Сниффер же, подобно отладчику и дизассемблеру, показывает детали функционирования системы в мельчайших подробностях. Установив Wireshark и проявив некоторую сноровку, ты сможешь увидеть сетевые взаимодействия, как они есть — в невинном, девственно обнаженном виде. И фильтры тебе в помощь!