Делаем закрытую сеть Wi-Fi с авторизацией по MAC-адресу на основе двух точек TP-Link TL-WR842ND.

Fix Me!

Цель: организовать закрытую сеть WiFi со списком доступа по MAC-адресу и централизованным управлением этим списком на точках доступа TP-Link TL-WR842ND.

Решаемые задачи:

1. Выбор методов авторизации и аутентификации с учетом ограничений используемого оборудования.
2. Выполнение настройки выбранного метода.

Требуемый уровень безопасности допускает организацию доступа к беспроводной сети на основе списка разрешенных MAC-адресов. В тоже время есть потребность ограничить любого желающего от доступа к сети. Хотя с другой стороны, любой желающий, просканировав эфир, может определить MAC-адреса разрешенных пользователей, и подключиться к сети скопировав полученный MAC на свой беспроводной интерфейс. Это противоречие и предстоит разрешить в работе: ограничить возможность доступа к сети «любому желающему» наиболее простыми методами с возможностью централизованного управления доступом (без надобности вносить изменения на каждой точке доступа в отдельности).

Введение

Для знакомства с теорией безопасности Wi-Fi, то неплохой обзор есть на Хабре: WPA2-Enterprise, или правильный подход к безопасности Wi-Fi сети

Добавим то, что автор при сравнении WPA2 Personal и WPA2 Enterprise основным отличием считает использование индивидуального ключа на каждого пользователя в WPA2 Enterprise.

А нам интересен WPA2 Enterprise для централизованного управления клиентами, а именно ведения списка MAC-адресов.

FreeRADIUS из коробки уже настроен на многие типы авторизации, конфиг достаточно большой и разбит на отдельные файлы. Понять что от чего зависит сходу достаточно непросто. Вот перевод статейки которая приблизительно описывает, чем занимается радиус сервер, чтобы разрешить или запретить доступ к сети.

RADIUSPicking Auth-TypeAuthenticating

Официальная WiKi FreeRADIUS настоятельно рекомендует новичкам прочитать эту страничку How things work in RADIUSPicking an Auth-TypeAuthenticating a userInsufficient information.

Вот мой вольный перевод этой статьи, за достоверность не ручаюсь.

И на всякий случай синхронизирую термины:

- Клиент в данном контексте (с точки зрения RADIUS-сервера) - это точка доступа.
- Пользователь (supplicant) - тот, кто хочет получить доступ в сеть используя точку доступа.
- Авторизация - с точки зрения настроек RADIUS (как я это понимаю), проверка наличия необходимых данных о пользователе для осуществления аутентификации. Хотя в широком смысле - процесс определения прав, предоставляемых пользователю.
- Аутентификация - процесс проверки пользователя, то есть является ли пользователь тем, кем о себе заявляет на этапе авторизации.

Большинство проблем с конфигурацией связаны с непониманием концепции FreeRADIUS. Редактирование конфигурационных файлов и перебор возможных опций не поможет понять концепцию.

Ни вы, ни радиус не знает и не решает, что клиент вам пришлет в запросе. Ответственность за то, что содержится в запросе лежит 100% на клиенте.

Радиус сервер смотрит на запрос и говорит:

```
Хмм... Могу ли я справится с этим запросом?
```

Ответ на этот вопрос зависит от того, какие типы аутентификации вы включили, что сервер может найти в базе данных, и что содержится в запросе.

Сервер начинает опрашивать модули этапа авторизации (в конфиге есть отдельная секция `authorize {}`):

```
Модуль Unix, можешь ли ты обработать это?
Модуль Pap, можешь ли ты обработать это?
Модуль Mschap, можешь ли ты обработать это?
```

В какой-то момент один из модулей скажет:

```
Да, я вижу в запросе нечто, что могу распознать. И я могу что-нибудь сделать!
```

Модули делают это на основании просмотра ключевых атрибутов пришедших в запросе, таких как `MS-CHAP-Challenge` (для `MS-CHAP`), или `CHAP-Challenge` (для `CHAP`), or EAP-Message (для `EAP`). Или же на основании предположения, что надо бы добавить что-то в каждый запрос.

Если модуль думает, что у него есть шанс чтобы ещё и аутентифицировать пользователя, он скажет:

```
Я не могу аутентифицировать этого пользователя сейчас ( Я просто говорю, чтобы авторизовать его)
Но мой приятель в секции Authenticate может!
Эй, добавьте  Auth-Type  для меня!
```

Если модуль ничего не распознаёт, или знает, что нет необходимости что-то искать, то он просто ничего не делает.

В конце авторизации, сервер проверит добавлен ли Auth-Type к запросу. Если нет, то немедленно отклонит запрос.

Давайте предположим, что клиент отправил запрос с атрибутом User-Password, и модуль pap включен в секции `autorize {}`. Тогда pap модуль установит Auth-Type = pap.

Таким образом, сервер снова обратится к модулю pap, но уже на стадии аутентификации (она также имеет свою секцию в конфиге `authenticate {}`), pap ответит:

```
Я вижу User-Password, который вводил пользователь.
Это замечательно, но мне нужно его с чем-то сравнить.
Ах! Другой модуль добавил «правильный» пароль для этого пользователя еще на стадии авторизации!
```

Итак, затем идет сравнение локального заранее известного правильного пароля с тем, который ввел пользователь (пришел в запросе). Это то, как работает аутентификация.

«правильный» пароль (заранее известный правильный пароль) был добавлен другим модулем. Например, модуль pap просто выполняет PAP аутентификацию и ничего больше. Преимущества такого подхода в том, что «правильный» пароль может быть добавлен в запрос из текстового файла `users`, `SQL`, `LDAP`, `/etc/passwd`, внешней программы и т.д. и т.п. в очень широких пределах.

Например, если подключен модуль `LDAP` в секции `authorize {}`, и сервер обрабатывает запрос, то если модуль найдет у себя запись с паролем (где-нибудь в каталоге LDAP), то он добавит этот пароль в запрос, чтобы другой модуль на этапе аутентификации мог использовать его.

Что, если клиент отправит `MSCHAP` запрос? Что будет с этим запросом делать радиус-сервер?

В этом случае, модуль mschap ищет в запросе атрибут `MS-CHAP` на этапе авторизации и когда находит устанавливает атрибут Auth-Type на себя (mschap), но уже для этапа аутентификации. Модуль базы данных (например, такой как LDAP, выше) получает информацию о «правильном» пароле и добавляет её в запрос. Затем модуль mschap вызывается уже для процесса аутентификации. Он просматривает запрос в поисках «правильного» пароля либо в виде текста, либо в виде NT-HASH (почему? Смотрите таблицу протоколов http://deployingradius.com/documents/protocols/compatibility.html). Если ни один требуемый вид пароля не будет

предоставлен хранилищем данных, то mschap скажет:

```
Извините, я не могу аутентифицировать пользователя, потому что не имею достаточно информации, мне нужно проверить MSCHAP.
```

Но сервер уже исчерпал варианты, его единственный вариант был mschap, так как это то, что клиент послал в запросе. Mschap модуль не смог ничего сделать, потому что ему не предоставили «правильный» пароль. Таким образом сервер за неимением вариантов запрос отклоняет. MSCHAP данные могли быть корректными, но у сервера не было способа убедиться в этом. Поэтому он ответил отказом.

Выводы

- FreeRADIUS организован по модульному принципу. Для того чтобы FreeRADIUS имел поддержку того или иного протокола, нужно подключить/отключить модуль в конфигурации. Многие, если не все модули имеют свою секцию конфигурации в общем конфиге.
- Поступивший запрос проходит процедуру авторизации, для настройки есть отдельная секция в конфигурационном файле authorize {}. Запрос обрабатывается попорядку перечисленными в секции модулями, модули пытаются распознать атрибуты и в случае успеха, авторизуют пользователя, указанного в запросе. По сути это означает, что в запрос добавляется атрибут Auth-Type, где указывается каким методом (модулем) производить аутентификацию. На этом же этапе модули реализующие хранилище данных ищут в своих базах данные, на указанного в запросе пользователя и также добавляют их к запросу. Если в процессе авторизации Auth-Type не установлен, то запрос откланяется. Иначе запрос переходит на этап аутентификации.
- Методы аутентификации настраиваются в секции authenticate {}. Тут запрос поступает в модуль, который задан в атрибуте Auth-Type. На основании данных вставленных в запрос из модуля-хранилища еще на предыдущем этапе происходит сравнение атрибутов пришедших от клиента с атрибутами из хранилища. На основании этого сравнения уже принимается решение о разрешении или отклонении доступа, либо об уточнении параметров у клиента.
- Вот такая концепция у радиус-сервера, в данном случае до безобразия упрощенная.

1. Обоснование

На имеющихся беспроводных маршрутизаторах TP-Link TL-WR842ND с заводской прошивкой у нас есть возможность использовать следующие типы защиты: WPA2–PSK, WPA2–Enterprise. Остальные варианты не рассматривались из-за их неактуальности: открытая сеть не соответствует поставленной цели; WEP – давно скомпрометирован; WPA-PSK/Enterprise – есть же WPA2 с более строгим подходом к шифрации.

WPA2-PSK – использует Pre-shared key, который в общем случае устраивал бы, но так как было желание управлять допущенными к сети MAC-адресами пользователей централизованно, а не на каждой точке в отдельности, доставляет некоторое неудобство.

WPA2-Enterprise – использует RADIUS-сервер для авторизации/аутентификации; учет пользователей (accounting) не поддерживается точкой доступа TP-Link TL-WR842ND. Отправляет в запросе к RADIUS MAC-адрес пользователя – следовательно можно вести список разрешенных MAC на RADIUS-сервере. WR842ND не умеет подставлять MAC-адрес вместо имени пользователя и пароля (такая возможность есть, например, в RouterOS от Mikrotik), поэтому придется заводить учетные данные (пользователь/пароль) либо на каждого пользователя можно с привязкой к MACу, либо одну общую учетку, о которой будут осведомлены все заинтересованные пользователи, а список MAC-адресов будет вестись отдельно.

Так как централизованно управлять списком доступа на основании MAC-адресов позволяет RADIUS-сервер, то выбор защиты Wi-Fi сети пал на WPA2-Enterprise.

WPA2-Enterprise поддерживает ряд методов аутентификации EAP. Для Window актуальны EAP-TLS и EAP-PEAPv0. EAP-TLS – поможет избавить пользователя от введения логина и пароля, но здесь встанет другая задача по разворачиванию инфраструктуры открытых ключей и распределению сертификатов. Причем встанет проблема установки сертификатов на мобильных устройствах. Хоть данный метод самый надёжный, но из-за сложности развёртывания и сопровождения для наших нужд он отпадает.

EAP-PEAPv0 с MS-CHAPv2 – от пользователя требуется доверие к точке доступа/серверу и знание связки логин/пароль. Доверие к точке доступа/серверу достигается проверкой предъявляемого сервером сертификата, либо отключением

этой проверки. Суть метода состоит в создании защищённого туннеля внутри которого осуществляется аутентификация пользователя по методу MS-CHAPv2. Простое развёртывание и сопровождение, особенно если слепо доверять точке доступа/серверу. Но страдает безопасность от «активных» злоумышленников.

Метод EAP-PEAPv0 с MS-CHAPv2 существенно проще в реализации, чем EAP-TLS и позволит организовать закрытую сеть, тем самым «любому желающему» будет гораздо сложнее, нежели просто определить разрешенный MAC-адрес, если конечно у него не будет альтернативного метода получения доступа к учетным данным.

Реализация метода EAP-PEAPv0 с MS-CHAPv2 во FreeRADIUS позволяет выдавать пользователю индивидуальную связку логин/пароль и привязывать их к MAC-адресу его оборудования. Хотя для наших целей это было признано нецелесообразным из-за дополнительной нагрузки на администраторов сети по сопровождению пользователей. Поэтому будет общий логин/пароль на всех, так как некоторые wi-fi клиенты не дают возможности оставлять данные поля пустыми.

Итог: беспроводной маршрутизатор TP-Link TL-WR842ND будет работать в режиме точки доступа с методом безопасности WPA2-Enterprise, который предполагает использование RADIUS-сервера. RADIUS-сервер на базе FreeRADIUS будет реализовывать список доступа на основе MAC-адресов и производить аутентификацию пользователей по методу EAP-PEAPv0 с MS-CHAPv2. При данном методе, пользователю необходимо доверять точеке доступа/серверу (или проверять сертификат для установки доверия) и знать учетные данные (логин/пароль) для прохождения MS-CHAPv2 аутентификации.

2. Реализация

Демон FreeRADIUSa после установки сразу же стартует, то есть после выполнения команды `apt-get install freeradius` мы имеем на выходе запущенный RADIUS-сервер. Официальная документация рекомендует вносить как можно меньше изменений в исходную конфигурацию, ибо она продумана для многих типичных задач RADIUS-сервера. FreeRADIUS «из коробки» уже умеет работать с WPA2-Enterprise, в том числе и с EAP-PEAPv0 с внутритуннельной аутентификацией MS-CHAPv2. Поэтому настройка FreeRADIUS заключается в том, чтобы создать собственный сертификат сервера, завести клиентов (точки доступа) , завести учетные данные для MS-CHAPv2 и добавить возможность фильтрации запросов по MAC-адресами из разрешенного списка.

Следует отметить, что приведенные команды и фрагменты конфигурации тестировались на Debian 7.5 с FreeRADIUS 2.1.12+dfsg-1.2

2.1 Создание production-сертификатов

Так как большинство методов корпоративного WPA2 требуют наличия сертификата и закрытого ключа как минимум на сервере, то придется его создать. Чтобы создать самоподписанный сертификат нам нужен сертификат и закрытый ключ собственного удостоверяющего центра, которые тоже придется создать.

Инструменты для создания сертификатов нужных freeradius у Debian лежат по пути /usr/share/doc/freeradius/examples/certs. Содержимое этого каталога следует скопировать в /etc/freeradius/certs. Необходимы файлы:

- Makefile
- ca.cnf
- server.cnf
- xpextensions,

А также можно скопировать README, в котором расписано что и зачем нужно и как этим пользоваться, поэтому дальше пойдут необходимые команды с краткими пояснениями, за более подробной инфой в README.

Для работы скриптов понадобится openssl и make.

Создаем файл с параметрами Диффи-Хеллмана, если он вдруг отсутствует в каталоге /etc/freeradius/certs:

```
# make dh
```

Результат работы: файл dh

Создаём корневой сертификат или **сертификат удостоверяющего центра**. Если до этого уже были какие-то

сертификаты (обычно тестовые), то удаляем:

```
# rm -f *.pem *.der *.csr *.crt *.key *.p12 serial* index.txt*
```

Редактируем под себя файл `ca.cnf`. Следует обратить внимание на `default_md = md5` (лучше поставить sha1), `default_days = 365` (возможно через год не захочется генерировать новый сертификат, тогда лучше ставить побольше), `input_password/output_password` и нужно отредактировать под себя всю секцию `[certificate_authority]`.

Далее выполняем:

```
# make ca.pem
# make ca.der
```

Результаты работы: `ca.pem`, `ca.key` и `ca.der` – сертификат удостоверяющего центра, его закрытый ключ и сертификат в формате понятном для Windows соответсвенно.

Создание серверного сертификата: Редактируем под себя файл `server.cnf`. Следует обратить внимание на `default_md = md5` (лучше поставить sha1), `default_days = 365` (возможно через год не захочется генерировать новый сертификат, тогда лучше ставить побольше), `input_password/output_password` и нужно отредактировать под себя всю секцию `[server]`. Важно чтобы значение `commonName` отличалась от соответствующего сертификата удостоверяющего центра.

Далее выполняем:

```
# make index.txt
# make serial
# make server.pem
```

Результат работы: помимо прочих файлы `server.pem` и `server.key` – сертификат и закрытый ключ сервера.

В конфигурации пути к сертификатам сгенерированным в папке /etc/freeradius/certs прописаны по умолчанию.

Если был изменён пароль на закрытый ключ сервера (`output_password`), то его придется указать в конфиге. А также нужно закомментировать путь к скрипту создания временных сертификатов. Это делается в файле /etc/freeradius/eap.conf в следующей секциии:

```
eap {
  ...
  tls {
    ...
    private_key_password=пароль
    ...
  # make_cert_command = "${certdir}/bootstrap"
    ...
  }
  ...
}
```

2.2 Описываем RADIUS-клиентов (точки доступа)

Добавляем в файл `/etc/freeradius/clietns.conf` следующее:

```
client TL-WR842ND_1 {
 ipaddr = 192.168.0.2
 secret = secret1
}

client TL-WR842ND_2 {
 ipaddr = 192.168.0.3
 secret = secret2
}
```

2.3 Заводим учетные данные для аутентификации по логину и паролю

В файл `/etc/freeradius/users` добавляем первую строку следующего содержания:

```
user Cleartext-Password := "pass"
```

2.4 Добавляем список доступа на основе MAC-адресов

(по мотивам http://wiki.freeradius.org/guide/Mac%20Auth)

Конфигурация FreeRADIUS не поддерживает списка MAC-адресов из коробки. Хотя MAC-адрес можно указать как дополнительный атрибут проверки в файле `/etc/freeradius/users`, например:

```
user Cleartext-Password := "pass", Calling-Station-Id == "1a-2b-3c-4d-5e-6f"
```

Для запроса с именем пользователя `user` будет проверяться пароль и сравниваться атрибут `Calling-Station-Id`. Чтобы это работало в случае EAP-PEAP следует в файле `/etc/freeradius/eap.conf` установить параметр `copy_request_to_tunnel = yes` в секции приведённой ниже:

```
eap {
  ...
  peap {
    ...
  copy_request_to_tunnel = yes
    ...
  }
  ...
}
```

Точка доступа в запросе шлет параметр `Calling-Station-Id =` “MAC-адрес«, причем формат представления MAC-адреса может быть различным на разных точках доступа. Чтобы избежать проблем из-за различного представления MAC-адреса, следует их приводить к одному виду. Для этого в фйле `/etc/freeradius/policy.conf` имеется уже готовая процедура - `rewrite.calling_station_id`. Чтобы ей воспользоваться её название нужно вставить в начало секции `authorize {}` после объявления `preprocess` (если он есть) в файлах `/etc/freeradius/site-available/default` и `/etc/freeradius/site-available/inner-tunnel` следующим образом:

```
authorize {
  preprocess
  rewrite.calling_station_id
  ...
}
```

Процедура нормализует формат MAC-адреса в атрибуте `Callin-Station-Id` из запроса, пришедшего на сервер, к формату `xx-xx-xx-xx-xx-xx` - нижний регистр и все через «минус». Поэтому в файле `/etc/freeradius/user` следует придерживаться данного формата. В принципе уже можно делать список доступа для фильтрации по MAC.

Вариант1.

Достаточно в файл `users` в самое его начало добавлять нужных пользователей с параметром `Calling-Station-Id`, если учетные данные нужны одинаковые, то можно их просто повторить, например:

```
user Cleartext-Password := "pass", Calling-Station-Id == "1a-2b-3c-4d-5e-6f"
user Cleartext-Password := "pass", Calling-Station-Id == "aa-bb-cc-dd-ee-ff"
user Cleartext-Password := "pass", Calling-Station-Id == "11-22-33-44-55-66"
```

Тем самым на одну пару логин/пароль вешаем три различных MAC-адреса.

Вариант2.

Можно пойти другим путём и создать для списка разрешенных MAC-адресов отдельный файл. Для этого создаем в каталоге `/etc/freeradius/` файл `authorized_macs` и объявляем его в модуле `files`. Для этого вставляем в конец файла `/etc/freeradius/modules/files` следующий кусок конфига:

```
#MAC Auth
files authorized_macs {
        key = "%{Calling-Station-ID}"
        usersfile = ${confdir}/authorized_macs
        compat = no
}
```

Чтобы проверка MAC-адресов из файла `/etc/freeradius/authorized_macs` выполнялась, нужно в файл `/etc/freeradius/sites-available/default` в секцию `authorize {}` после объявления `rewrite.calling_station_id` вставить прокомментированные строки:

```
authorize {
        preprocess
        rewrite.calling_station_id
        #
    # Здесь инициируем поиск по файлу authorized_macs
        authorized_macs
        #
    # А здесь отвергаем запрос, если поиск завершился неудачей
        if (!ok) {
          reject
        }
}
```

Теперь файл `authorized_macs` заполняем разрешенными для авторизации MAC-адресами. Каждый MAC на новой строке. Следует учитывать, что формат строго следующий: `xx-xx-xx-xx-xx-xx` , где x-либо цифра, либо буквы от a до f в нижнем регистре. Например:

```
1a-2b-3c-4d-5e-6f
aa-bb-cc-dd-ee-ff
11-22-33-44-55-66
```

Следует отметить, что атрибут `Calling-Station-Id` указанный в файле users для определённой пары логин/пароль также будет действовать, но его проверка будет осуществляться на этапе, который проходит позднее. Так что если поступит запрос со значением `Calling-Station-Id`, не указанном в файле authorized_macs, то этот запрос сразу отклонится, вне зависимости, что указано в файле users.

Итог: выше описаны необходимые измения в конфигурации по-умолчанию, чтобы получить требуемый функционал от freeradius. Еще нужно настроить точки доступа, куда нужно вписать IP адрес RADIUS-сервера и секретную фразу из секций `client {}`. Также в конфигурации описано много лишних методов аутентификации и модулей дополнительной обработки, которые можно удалить/закомментировать без ущерба функционалу. При выбранном типе аутентификации у FreeRADIUS нет возможности выдавать клиентам IP адреса и прочие сетевые настройки, это должен делать отдельный DHCP-сервер.

Советы и замечания

При изменении файлов конфигурации, в том числе файлов `users` и `authorized_macs` требуется принудительное перечитывание конфига сигналом `SIGHUP`:

```
# kill -HUP `cat /var/run/freeradius/freeradius.pid`
```

При изменениях в конфиге `radiusd.conf` лучше демон перезагрузить:

```
service freeradius restart
```

Чтобы проверить конфигурацию следует запускать демон в режиме отладки:

```
# freeradius –X
```

При таком запуске можно увидеть как freeradius распарсил конфиг и увидеть как он обрабатывает запросы. Очень помогает при отыскании различных проблем или при объяснении неожиданного поведения.

В комплекте идет полезная утилитка `radtest` – простой RADIUS-клиент. С ней можно проводить некоторые простые тесты для проверки конфигурации. Возможный пример использования (тестирование внутритуннельной аутентификации по порту 18120):

```
# radtest -t mschap -x user pass localhost:18120 4321 testing123
Sending Access-Request of id 230 to 127.0.0.1 port 18120
        User-Name = "user"
        NAS-IP-Address = 192.168.0.80
        NAS-Port = 4321
        Message-Authenticator = 0x00000000000000000000000000000000
        MS-CHAP-Challenge = 0xaf147766f1965e97
                                                                MS-CHAP-Response        =
0x000100000000000000000000000000000000000000000000000000000000fc1329e665dda81c87b76830a2344ae2f5388db64
73a2f3
rad_recv: Access-Accept packet from host 127.0.0.1 port 18120, id=230, length=84
        MS-CHAP-MPPE-Keys = 0x37653a880f5e2682eacb842922c8d34c1239e1a550a16fee0000000000000000
        MS-MPPE-Encryption-Policy = 0x00000001
        MS-MPPE-Encryption-Types = 0x00000006
```

Чтобы в логах отображалась информация о попытках аутентификации следует включить следующие параметры в `radiusd.conf`:

```
          auth = yes
          auth_badpass = yes
          auth_goodpass = yes
```

http://sysadminmosaic.ru/freeradius/freeradius_wifi

2019-05-11 00:50