# Wi-Fi сети: векторы атаки и Kali Linux 02

WPA и WPA2 (первый основан на черновике IEEE, второй — на финальной версии, но в нашем случае оба можно считать синонимами) используют довольно хитрую схему обмена ключами. Вернее, их *необмена*. Как мы помним по первой части, самое слабое место в любой защите — это передача ключей, даже зашифрованных — ведь хакер может перехватить эти значения и попытаться подобрать по ним изначальный ключ в offline-режиме, то есть никак более не общаясь с точкой доступа или клиентом. Разработчики WPA устранили всякий обмен и шифрованными, и открытыми паролями. Как это всё работает как раз и будет описано ниже.

Начнём с конца. Допустим, мы хотим передать зашифрованный пакет данных. Для этого нам нужны сами данные и ключ, по которому алгоритм вроде AES (а именно он используется в CCMP) преобразует их в ~~нечто неудобоваримое~~ шифрованную форму. Дешифровать данные обратно можно передав этому же алгоритму шифрованную строку и ключ, который использовался для зашифровки.

Самый простой способ — использовать в виде ключа сам пароль от беспроводной сети. Однако это чревато серьёзными проблемами:

- **Слишком быстрые вычисления** — если мы знаем начальные байты исходных данных (а почти всегда это так, ведь в начале идёт какой-нибудь стандартный заголовок), то простым перебором мы сможем попробовать дешифровать перехваченный пакет, подставляя разные пароли и смотря, что получилось на выходе — если он содержит похожий на корректный заголовок, то скорее всего мы нашли исходный пароль.
- **Гарантия на всю жизнь** — если каким-то образом мы узнали пароль, то сможем расшифровать всё, что передавалось ранее и будет передано позднее, пока администратор не сменит ключ сети. А как показывает практика, даже при смене роутера люди обычно ставят на него старый пароль, так что надеяться на их благоразумие не, *гм*, благоразумно.
- **Отсутствие защиты от любопытного соседа** — если мы можем авторизоваться в сети, то ничто не помешает нам читать пакеты других клиентов и даже произвольно подменять их, так как ~~тапки одни на всех~~ пароль, используемый для шифрования наших данных, используется и для всех остальных клиентов в рамках этой сети

Это похоже на то, как если бы между нами и конечной целью стояло много-много дверей, каждая из которых ведёт внутрь. Двери — клиенты сети, «нутрь» — сама сеть. Если ко всем дверям подходит одинаковый ключ, то мы можем пройти через соседнюю дверь и посмотреть, что там происходит.

Вторую и третью проблемы легко устранить, добавив к ключу случайное значение, которое меняется каждый раз при запуске защищённого соединения и даже в процессе его работы. Первую — очень длинным ключом. Но тогда встаёт вопрос: ~~кто будет его помнить~~ если его будет вводить не человек, то как получить этот длинный ключ из короткого (от пользователя), да ещё и не подверженному атаке на перебор?

Для этого есть два стандартных решения: либо введение ограничений на число попыток входа, либо многократное шифрование, чтобы замедлить процесс получения конечной строки. Первый вариант не подходит, так как он годится для online-атак (обычно на формы входа на веб-сайты) и бесполезен, когда сам хэш уже «утёк». А вот второй — как раз наш случай.

Однако и тут есть проблема: если мы просто прогоняем некий алгоритм 10 000 раз на нашей исходной строке с паролем, то возможен вариант, когда хакер создаст словарь, предварительно прогнав этот же алгоритм на всех возможных комбинациях паролей, и затем всё, что ему останется — перехватить рукопожатие и посмотреть хэш в нём по таблице, которую он уже успел вычислить. Если в таблице такой хэш есть — значит, есть и исходная строка, из которой он был вычислен, и она ему известна. А затем ту же самую таблицу можно использовать для расшифровки других данных. Такая таблица называется *радужной* (rainbow table). Особую популярность техника получила при подборе паролей в украденных базах старых форумных и прочих движков на PHP.

Решается это добавлением «соли» — случайной или более-менее уникальной строки, которая даже при совпадении самих паролей сделает вычисленные хэши разными.

Надеюсь, мой краткий обзор пролил кое-какой свет на то, какую чёрную магию используют инструменты вроде **aircrack-ng** и **hashcat** при попытке определить пароль в рукопожатии. Как мы теперь знаем, им нужно раскрутить всю цепочку назад, имея на руках всего-навсего хэш-сумму одного из пакетов (MIC). А именно, сделать следующее:

1. Первым делом вычислить главный ключ сети — **PMK**. Для этого берётся пароль и имя сети. Последнее берётся из перехваченных пакетов рукопожатия (об этом ниже), а первый можно брать из словаря по одному или перебирать всё доступное пространство ключей «в лоб», ~~но только если вы джедай и обладаете особой уличной магией в виде пары GPU~~.
2. Дальше вычисляется **PTK** — хэш-сумма из полученного PMK (выше), MAC-адресов клиента и ТД и случайных nonce-строк от них же (берутся из перехваченных пакетов).
3. Наконец, для пакета, переданного с MIC, вычисляется **MIC** на основании PTK, полученного выше, при этом переданный MIC игнорируется (так как он зависит от всего сообщения, в том числе самого MIC, то перед вычислениями это поле устанавливается в 0, иначе невозможно вычислить сумму, не зная MIC, для вычисления которого нужно знать эту сумму).
4. Оба MIC **сравниваются** — если совпали — пароль найден (PTK верен > PMK верен > пароль верен), если нет — *go to line 1*

Таким образом, каждая итерация требует как минимум 8192 вычислений SHA-1, который в 3 раза медленнее MD5. Это очень затратный процесс. А что мы получим в итоге?

А в итоге — только исходный пароль и PTK того незадачливого клиента, аутентификацию которого мы перехватили. Это значит, что мы не сможем прочитать потоки других клиентов — у них другие PTK. Мы не сможем прочитать данные, которые этот клиент передал до того, как подключился — у него тоже был другой PTK.

Мы даже не сможем прочитать то, что он передаст после того, как подключится в следующий раз — ведь PTK снова изменится!

Это очень важный вывод, который в четвёртой статье цикла нам будет очень кстати при перехвате пакетов в Wireshark. Там нам придётся не только получить пароль от сети, как описано в этой статье, но и перехватить рукопожатия всех клиентов, которых мы хотим прослушать (либо использовать ARP-спуфинг, но это атака на другом уровне). Та ещё работёнка.

Итак, ещё раз: наша задача — перехватить первые 4 пакета, которыми обмениваются клиент и точка доступа (по 2 с каждой стороны) при установлении соединения. В сумме они называются рукопожатием (handshake). После них начинается уже шифрованная передача данных, из которой нам ничего не вытащить. Кстати, эти пакеты — часть протокола EAP (или EAPOL), и под таким именем они фигурируют в Wireshark (см. следующую статью цикла).

Перехватив их мы можем сохранить их к себе и затем провести offline-атаку — то есть попытаться подобрать оригинальный пароль к сети, поочерёдно пробуя разные пароли для генерации PMK > PTK > MIC и сравнивая последний с тем, который был передан на самом деле, как это было описано выше.

Перехват делается с помощью **airodump-ng**. С параметрами можете поиграть, о них писалось там же, но в общем случае вызов выглядит так:

```
airodump-ng mon0 -c 5 --bssid AP_BSSID -w caps
```

Перед этим вам нужно перевести свою карту в «хакерский режим» (monitor mode) и проделать все прочие манипуляции (смена MAC, txpower и т.д.).

В команде выше мы используем интерфейс под идентификатором `mon0` для сбора пакетов атакуемой сети на канале 5, которая имеет MAC, указанный после `--bssid`, сохраняя пакеты в файл `caps-NN.cap` (по умолчанию используется стандартный формат **libpcap**, который поддерживается очень многими библиотеками на всех ОС, в том числе и Wireshark). **NN** будет заменено на уникальное число, таким образом при повторных запусках **airodump-ng** с теми же параметрами старые файлы не будут перезаписаны, а будут иметь имена вида `cap-01.cap` и дальше.

Допустим, что атакуемая нами сеть имеет BSSID `4F:B1:A4:05:5C:21` и находится на канале 11. Тогда делаем так:

```
airodump-ng mon0 -c 11 --bssid 4F:B1:A4:05:5C:21 -w caps
```

После запуска откроется уже знакомое консольное окно с двумя таблицами. Оставим его висеть, пока кто-нибудь не подключится к нашей сети…

Но ведь мы можем ускорить этот процесс! Читатель помнит про то, что мы можем отключить имеющихся клиентов и заставить их передать данные аутентификации повторно — очень полезно для таких непоседливых хакеров, как мы.

**aireplay-ng** с радостью поможет нам:

```
aireplay-ng mon0 -0 5 -a 4F:B1:A4:05:5C:21 -c 5B:23:15:00:C8:57
```

`5B:23:15:00:C8:57` — MAC-адрес клиента, который мы почерпнули из таблицы запущенного ранее **airodump-ng**.

Если всё сделано правильно, **aireplay-ng** выведет 5 строк вида `Sending directed deauth`, а в окне с **airodump-ng** мы должны наблюдать быстро увеличивающееся число «потерянных» пакетов (в графе **Lost**). Оно может пойти на тысячи.

После этого наш дорогой клиент, если он был в радиусе действия нашего передатчика и если он был активный (иногда устройство остаётся подключенным к сети, но не использует её, и отключение не заставляет его переподключиться), тут же начнёт авторизацию заново и мы поймаем эти пакеты, о чём в правом верхнем углу **airodump-ng** победно сообщит надписью `[ WPA handshake: 4F:B1:A4:05:5C:21 ]` (MAC-адрес сети, рукопожатие с которой было перехвачено).

Если так и случилось — атака проведена успешно, **airodump-ng** можно закрывать, копировать полученный `caps-01.cap` на флешку ~~и сматываться~~.

#### aircrack-ng

Самый простой способ перебора. **aircrack-ng** использует только ЦП, зато отлично поддерживает многопоточность. Она перебирает значения по словарю для WPA-сетей (также умеет взламывать WEP).

Kali идёт с набором словарей в `/usr/share/wordlists/`, но их при желании можно легко найти на просторах Интернета любого размера — от мегабайт до десятков гигабайт. Довольно хорошая компиляция — `WPA-PSK Wordlist 3 Final`, а также сгенерированный словарь из всех комбинаций 8-значных числовых паролей, который получается с помощью `crunch 8 8 1234567890`

```
aircrack-ng -w /usr/share/wordlists/fasttrack.txt caps-01.cap
```

`/usr/share/wordlists/fasttrack.txt` (идёт в комплекте с Kali) — путь к словарному файлу с паролями, один пароль на строку. Строки короче 8 символов будут проигнорированы, так как это минимальная длина для WPA.

На моём i7 3840QM 4x3.8 GHz **aircrack-ng** выжимает ~4700 паролей в секунду. Таким образом, мы можем подсчитать, сколько времени птребуется для полного перебора всех возможных комбинаций из 8 цифр:

```
(10^8) / (4700 * 3600) = 5,91 часов
```

Проверить скорость подбора без собственно подбора (бенчмарк) можно так:

```
aircrack-ng -S
# 4713 k/s
```

А так можно посмотреть, сколько при подборе будет использоваться ядер:

```
aircrack-ng -u

# No CPU detected: 8  (SSE2 available)
```

Итого, за 6 часов можно перебрать 10 миллионов паролей на high-end мобильном ЦП. Цифровой пароль такой длинны не стоит рассматривать в качестве серьёзной защиты, и даже не поэтому — применив ГП (GPU), который даёт в десятки и сотни раз большую скорость, такой пароль сломается за минуты. Об этом ниже.

Если атака прошла успешно, то есть **aircrack-ng** нашла пароль, то она завершит работу и на экране будет выведено радостное `KEY FOUND! [ ... ]` — запишите его и используйте для входа в сеть. Также можно записать найденный пароль в файл через **-l pass.txt**, что полезно при запуске перебора в фоне как `aircrack-ng ... -l pass.txt &`

Пояснения по формуле выше:

- **10^8** — 10 в степени 8, число возможных комбинаций; вычисляется как `число_возможных_символов ^ длина_строки`. К примеру, для 6-значного пароля из латинских символов в нижнем регистре будет $26^6$ = 308 915 776 комбинаций. К слову, это наглядно показывает, что длина пароля имеет куда большее значение, чем возможное число символов в нём: запомнить "`this weirdo voodoo`" куда проще, чем "`0.o@&z%_`" — тогда как комбинаций первого на $10^{25}$ больше (десять и 24 нуля).
- **4700** — число сравнений паролей в секунду (скорость перебора)
- **3600** — приведение результата деления от паролей в секунду к паролям в час (60 секунд * 60 минут)

Ещё подсчёты для сравнения:

```
 (26^8) / (4700 * 3600 * 24)        = 514 суток для перебора 8-значного пароля с
a-z
(10^10) / (4700 * 3600 * 24)        = 24,6 суток для 10-значного цифрового
пароля
(26^10) / (4700 * 3600 * 24 * 365)  = 952 года для 10-значного пароля с a-z
(10^12) / (4700 * 3600 * 24 * 365)  = 6,7 лет для 12-значного цифрового пароля
(10^14) / (4700 * 3600 * 24 * 365)  = 674,6 лет для 14-значного цифрового пароля
```

Статистику для более длинных строк нет смысла вычислять, если речь идёт только о ЦП.