Защита доступа к беспроводной сети предприятия с помощью Heimdal Kerberos и freeRadius.

mkondrin
из *hppi.troitsk.ru*

31 января 2012 г.

Аннотация

Как разрешить доступ сотрудников к беспроводной сети предприятия по учётной информации из единой базы данных Heimdal-Kerberos?

1 Зачем нужен WPA-Enterprise для беспроводных сетей?

Поскольку беспроводные сети уже широко используются на практике, то постепенно возникает вопрос уже не столько защиты доступа к ним, сколько унификации этого доступа. Одно дело, если вы у себя дома устанавливаете беспроводную точку доступа в интернет и, чтобы помешать своим соседям безнаказанно таскать у вас трафик, вводите на ней общий пароль WPA-PSK (Wireless Protected Access with Pre-Shared Key), каковой пароль вы и сообщаете всем своим близким и вбиваете его на все электронные устройства, которым может понадобиться доступ в сеть. В тоже время при развёртывании беспроводной сети на предприятии ввиду принципиальной физической открытости WiFi-сети основная проблема – это шифрование каналов беспроводной передачи данных. Как в таком случае подсказывает житейский здравый смысл – секрет известный более, чем двум сторонам, уже не является секретом. Стало быть, защита средствами общего ключа там неприемлема, и необходимый минимум защиты достигается при наличии индивидуального пароля/ключа у каждого из сотрудников. Для беспроводного доступа такой уровень безопасности можно обеспечить средствами WPA-Enterprise (название говорит само за себя). При этом стандарт WPA-Enterprise (или его расширение WPA2-Enterprise) требует наличия RADIUS-сервера, который как раз хранит базу данных пользователей, и к которому должны обращаться все точки беспроводного доступа для обслуживания запросов пользователей на предоставление доступа к сети.

> Название протокола RADIUS является акронимом и расшифровывается как Remote Authentication Dial-In User Service (Служба удалённой аутентификации dial-in пользователей), и уже намекает на его почтенный возраст, отсылая к тем далёким временам, когда телефонная линия (dial-in) являлась основным средством доступа в сеть. Но на самом деле в последнее время этот протокол, можно сказать, испытал второе рождение, что связано не только с развитием беспроводных сетей и стандарта WPA/WPA2-Enterprise, но и с распространением VPN и интернет-телефонии (и протокола SIP), где RADIUS также широко используется.

Однако, как правило, когда на предприятии возникает необходимость во внедрении беспроводного доступа и развёртывании RADIUS-сервера, там уже имеется единая база пользователей с уже отлаженным циклом ротации и проверки сложности паролей. Внедрение и поддержание в рабочем состоянии дублирующей службы всегда является дополнительной нагрузкой для сетевого администратора, и поэтому у него появляется мысль – а нельзя ли

как нибудь настроить RADIUS, чтобы он использовал уже имеющуюся базу учётных данных пользователей? В данном тексте как раз и будет рассмотрен один из способов решения этой проблемы в случае, если в качестве системы единого доступа используется Heimdal-Kerberos. Вообще говоря, нет принципиальных ограничений, чтобы этот метод (после минимальных изменений) не работал бы и для других типов серверов Kerberos – MIT, Shishi или AD. Хотя сам по себе Radius является не только (вопреки своему названию) службой проверки подлинности пользователей, но и службой авторизации и контроля действий пользователей (что называется AAA – Authentication, Authorization and Accounting), но в данном тексте он будет рассматриваться только с точки зрения первого A – как прокладка между точкой беспроводного доступа и средством удалённой аутентификации пользователей Heimdal-Kerberos.

2 Программное обеспечение

Для наладки и тестирования потребуется работающая система с операционной системой Линукс. Разумеется, рассматриваемое решение не ограничивается только одной операционной системой и в конце статьи будут предложены рецепты переноса на гетерогенное окружение. Разумеется, также необходим работающий сервер Heimdal Kerberos. Его запуск и настройка рассматривались ранее в статьях [1, 2], так что на этом я останавливаться не буду, единственное требование, чтобы в базу данных пользователей был вписан тестовый пользователь idiot с паролем DeBiLL (`kadmin -l add idiot -p DeBiLL`). Такой логин и пароль выбран не случайно, а чтобы вы знали, кто вы есть такой, если забудете удалить этого пользователя после тестировании системы (`kadmin -l del idiot`), поскольку эти учётные данные будут широко использоваться в тестовых скриптах, приведённых в этой статье.

С выбором сервера Radius есть несколько возможностей. Я остановился на FreeRadius [3], поскольку он открыт, доступен и достаточно подробно документирован. Также хорошим вариантом будет коммерческий сервер Radiator[4], но скорее всего, если вы используете его, то все решения предложенные в этом тексте вас вряд ли заинтересуют, поскольку разработчики Radiator работают в тесном контакте с разработчиками Heimdal-Kerberos (о чем свидетельствует, например, вот эта запись в блоге его основного разработчика Love Hörnquist Åstrand [5]). Так что в той или иной степени интеграция Radiator и Heimdal там должна работать, что называется, “из-коробки”. Также в качестве самостоятельного сервера Radius может работать демон hostapd [6]. Но поскольку основная его функциональность – это обслуживание со стороны user-space беспроводных устройств в режиме точек доступа (Master Mode), то возможности его встроенного Radius сервера мне показались несколько урезанными. Можно также упомянуть (для любителей экстрима и bleeding-edge) про протокол Diameter, который представляет собой расширение протокола RADIUS и реализуется например сервером freeDiameter [7]. В названии протокола подразумевается, что Diameter=Radius*2.0, но насколько он совместим с имеющимися на текущий момент в продаже беспроводными маршрутизаторами и стандартом WPA2-Enterprise я судить не берусь.

> Кстати, о маршрутизаторах и точках доступа. Хотя практически все производители точек доступа и беспроводных роутеров представленных на рынке, утверждают, что их продукция полностью поддерживает протокол WPA/WPA2, но зачастую это ограничивается поддержкой только PSK-режима (несмотря на заверения в технической документации об обратном). Такова оказалась сравнительно новая модель DIR-320/NRU версии B1 с прошивкой 1.2.94. В отличии от популярной модели DIR-320 (версий A1/A2) эта модель с новым чипсетом Ralink5350 на рынке появилась только в начале 2011 года. Хотя интерфейс этого роутера и позволял выбирать WPA режим с полями для ввода адреса и порта RADIUS-сервера, но фактически он такой функциональностью не обладал, поскольку все мои попытки подключения он игнорировал и, соответственно, в логах не фиксировалось

никаких попыток подключения от этого роутера к моему RADIUS-серверу. Сходным образом повела себя и служба поддержки фирмы-разработчика, проигнорировав мои письма с просьбами о помощи. Что ж, спишем это на проблемы, связанные с переходом на новую версию прошивки, и что в дальнейшем замеченные недостатки будут исправлены. Пока же мне пришлось ограничиться проверенной временем моделью – беспроводным маршрутизатором ASUS-WL500gP.

Установить FreeRADIUS можно на тот же сервер, где уже работает KDC от Heimdal. Лучше всего воспользоваться стандартными пакетными менеджерами от того дистрибутива Линукс, что работает на этом сервере, однако собрать FreeRadius из исходного кода совсем несложно. Следует только убедиться, что при этом будет собран модуль `rlm_krb5` и что к нему подключены имеющиеся библиотеки от Heimdal. В остальном проблем не должно возникнуть, но, на всякий случай, я приведу свой скрипт для сборки последней на текущей момент версии freeradius 2.1.12, который выглядит примерно так:

```
tar xvf freeradius-server-2.1.12.tar.bz2
cd freeradius-server-2.1.12
./configure --prefix=/usr --sysconfdir=/etc --enable-heimdal-krb5
make
make install
```

Если вы захотите установить freeRadius во временный каталог, чтобы впоследствии собрать из него пакет под свой дистрибутив, вам следует помнить, что в данном случае команда `make install` игнорирует стандартную для таких целей опцию `DESTDIR=/tmp/foo/`, а вместо этого использует переменную окружения `R`.

В операционной системе Линукс программное обеспечение для подключения к беспроводной сети с клиентской стороны состоит из двух независимых частей: собственно ядерных модулей/драйверов, обслуживающих как имеющееся беспроводное устройство на клиентском компьютере, так и криптографическую защиту канала беспроводной связи с одной стороны, и пользовательские утилиты с другой, называемые supplicant-ами, которые предоставляют интерфейс для выбора пользователем какой-нибудь из беспроводной сети, ввода пароля и т.д. Для Линукса в настоящий момент имеется два варианта supplicant-ов, примерно одинаковой степени неудобства: Xsupplicant [8] и wpa-supplicant [9]. Как раз последний для нас будет интересен тем, что в своём составе имеет консольную программу `eapol_test`, которая позволяет протестировать работоспособность Radius сервера в определённых режимах криптографически защищённых каналов. Так что этот пакет тоже будет полезно установить на том же сервере, пусть даже на нём и отсутствует беспроводное оборудование, вместе с freeRadius, поскольку утилита `eapol_test` позволит изрядно сэкономить время при его отладке. Единственная проблема, что `eapol_test` не собирается по-умолчанию, и возможно поэтому отсутствует в пакете для вашего дистрибутива. Но это не беда, поскольку пакет wpa-supplicant можно будет собрать и из исходного кода. Процедура сборки, однако, несколько отличается от привычной `./configure; make; make install`. Распаковав последний на текущий момент пакет `wpa_supplicant-0.7.3.tar.gz`, нужно будет в директории `wpa_supplicant-0.7.3/wpa_supplicant` отредактировать файл `.config`, раскомментировав (т.е. удалив символ # в начале строки) директиву:

```
#CONFIG_EAPOL_TEST=y
```

и может быть другие, какие сочтёте нужным после прочтения комментариев в этом файле. После этого запуск команды `make` в той же самой директории запустит процесс сборки нужных утилит, но в итоге перемещение команды `eapol_test` в какую-нибудь из стандартных директорий в списке `PATH` нужно будет проделать вручную.

Последнее замечание по программному обеспечению касается уже драйверов беспроводных устройств. При тестировании я использовал довольно старый ноутбук с предустановленной операционной системой Линукс, где имелось беспроводное

встроенное USB-устройство RTL-8187B. Проблемы Линукс с драйверами достаточно широко известны, так что я особо не удивился, что в режиме WPA этот ядерный драйвер работать не смог (хотя ни в открытом режиме, ни в режиме WPA-PSK я до этого сложностей не испытывал). Очевидным симптомом такого рода проблем является появление в консоли, где запущен `wpa_supplicant` в тестовом режиме, сообщений напоминающих вот это:

```
ioctl[IPW_IOCTL_WPA_SUPPLICANT]: Operation not supported
```

что и означает отсутствие необходимого ядерного вызова (ioctl) у драйвера устройства. Эту проблему мне удалось обойти использованием пакета ndiswrapper [10], который является оболочкой, позволяющем использовать под Линукс драйверы беспроводных устройств от Windows. В моем случае драйвер от производителя запущенный из-под ndiswrapper позволил работать в режиме WPA2-Enterprise, пусть и в ущерб скорости. В настоящее время под Linux появилось новое поколение драйверов беспроводных устройств, так называемых mac80211-based, которые совместимы даже со старыми ядрами, начиная с 2.6.27 [11]. Но на практике я их не тестировал.

3 Стандарт IEEE802.11 и криптографические алгоритмы

Как уже упоминалось ранее, при работе с беспроводными сетями, поскольку их физически достаточно легко прослушать, следует уделять особое внимание проблемам шифрования беспроводного канала. Более того это шифрование должно быть произведено до того, как по этому каналу будут переданы идентификационные данные пользователя: пароли, ключи или даже хэши паролей и ключей, поскольку все они могут быть перехвачены злоумышленниками и подвергнуты криптографическому взлому уже в оффлайне. Эти проблемы не вполне тривиальны и стандарт на беспроводные системы (который принято обозначать как IEEE802.11) в процессе своего развития достаточно серьёзно на этом поскользнулся, поскольку первоначальный алгоритм шифрования канала WEP (Wired Equivalent Privacy – приватность эквивалентная проводной сети), предложенный в 1999 году, удалось взломать в течении того же года. Последующие модификации стандарта (IEEE802.11i является эквивалентом WPA2), эту проблему в значительной мере разрешили, путём замены алгоритма WEP на TKIP(Temporal Key Integrity Protocol) или ещё более стойким шифром CCMP [1]. Поскольку вычислительные ресурсы беспроводного оборудования в значительной мере ограничены, то из соображений быстродействия оба эти алгоритма предусматривают симметричное шифрование с периодически обновляемыми общими ключами, т.е. при создании беспроводного соединения обе стороны должны быть каким-то образом обеспечены общим ключом. Несложно сообразить, что этим ключ может быть тем или иным способом сгенерирован при аутентификации пользователя, на основе его пароля, логина и т.д. При этом авторы стандарта не стали настаивать на каком-то определённом алгоритме, а предложили достаточно абстрактный метод EAP (Extensible Authentication Protocol – расширяемый протокол аутентификации) [12], переложив задачу по его детализации на конечных разработчиков программного обеспечения и беспроводного оборудования. В настоящее насчитывается уже несколько десятков EAP протоколов. Не все они одинаково распространены, поэтому в этой статье будет рассмотрены только пара наиболее широко используемых.

4 Настраиваем freeRadius для работы в режиме EAP-TTLS/PAP

Этот метод EAP задокументирован в RFC5281 [13]. Его идея достаточно проста: средствами TTLS создаётся зашифрованный канал, по которому в открытом виде передаётся

[1] Counter with CBC-MAC mode Protocol, иногда в настройках беспроводного оборудования он обозначается как AES – Advanced Encription Standard, т.к. он используется этим стандартом

пароль пользователя (PAP – password authentication protocol, протокол проверки подлинности пароля) или же используется более продвинутые способы аутентификации. Этот способ удобен в развёртывании и обслуживании по многим причинам. Он достаточно криптографически стоек и безопасен, поскольку пароль в открытом виде не передаётся по сети. При этом в отличии от сходного по названию и реализации метода EAP-TLS [14] вам не нужно иметь на клиентских машинах ключи, подписанные корневым сертификатом, так что нет необходимости их менять на всех машинах при устаревании корневого сертификата. Немаловажно и то, что алгоритм EAP-TTLS/PAP было бы достаточно легко реализовать на базе freeRadius в связке с Heimdal, если б не одно но – необходимость правки конфигурационных файлов самого freeRadius.

Эти файлы находятся в директории `/etc/raddb`. Предполагается, что конфигурация, созданная по умолчанию при установке freeRadius, приспособлена для работы при условии, что база данных пользователей и их пароли содержатся в файле `/etc/raddb/users`. Замена же на любую другую базу данных, типа SQL, LDAP или как в нашем случае Kerberos, уже требует модификации конфигурационных файлов. Собственно, проблема связана даже не столько с объёмом и количеством этих файлов, сколько с тем фактом, что все они представляют из себя скрипты, написанные на языке программирования unlang, придуманным разработчиками freeRadius специально для этой цели. Пусть этот язык и ограничен по своей функциональности (“неполон по Тьюрингу” и является domain-specific языком, если выражаться научным терминами), но можно сказать, что значительная часть функциональности freeRadius реализована на этом языке. Причём файлы из директории `/etc/raddb` – это ещё только малая видимая часть айсберга, поскольку огромная часть кода, состоящая из так называемых статических словарей/dictionary, вынесена в системную директорию `/usr/share/freeradius`. Так что не сильно разбираясь в самом языке программирования пытаться править программу, написанную на этом языке, выглядит не очень хорошей идеей, которая может плохо закончится. Но выручает то, что все эти файлы снабжены подробными комментариями, и сам язык выглядит достаточно наглядно, поэтому при желании разобраться в настройке freeRadius можно. Я же приведу свои “минимально-функциональные” конфигурационные файлы (т.е. они с одной стороны минимальны по объёму и при этом обеспечивают требуемую функциональность), которые я использовал для решения задачи аутентификации пользователей беспроводной сети по протоколу WPA. Упрощения в логике работы RADIUS-сервера не столько даже оптимизируют его работу, сколько помогают администратору разобраться в отладочной информации при настройке freeRadius. Во многом при редактировании этих файлов я воспользовался идеями с сайтов [15, 16].

Отправной точкой является файл `/etc/raddb/radiusd.conf`. При старте freeRadius сначала читает этот файл, а уже затем подверстывает остальные куски указанные в директивах `$INCLUDE`.

```
prefix = /usr
exec_prefix = ${prefix}
sysconfdir = /etc
localstatedir = ${prefix}/var
sbindir = ${exec_prefix}/sbin
logdir = ${localstatedir}/log/radius
raddbdir = ${sysconfdir}/raddb
radacctdir = ${logdir}/radacct
name = radiusd
confdir = ${raddbdir}
run_dir = ${localstatedir}/run/radiusd
db_dir = ${raddbdir}
libdir = ${exec_prefix}/lib
pidfile = ${run_dir}/${name}.pid
max_request_time = 30
cleanup_delay = 5
max_requests = 1024

listen {
        type = auth
```

```
        ipaddr = *
        port = 0
}

hostname_lookups = no
allow_core_dumps = no
regular_expressions     = yes
extended_expressions    = yes

log {
        destination = files
        file = ${logdir}/radius.log
        stripped_names = no
        auth = yes
}

checkrad = ${sbindir}/checkrad

security {
        max_attributes = 200
        reject_delay = 1
        status_server = yes
}

proxy_requests  = no

$INCLUDE clients.conf
$INCLUDE policy.conf

thread pool {
start_servers = 5
max_servers = 32
min_spare_servers = 3
max_spare_servers = 10
max_requests_per_server = 0
}

modules {
$INCLUDE ${confdir}/modules/
$INCLUDE eap.conf
}

$INCLUDE sites-enabled/
```

По сравнению с исходным файлом, основная правка проделанная мной, заключается в удалении комментариев, удалении обработки proxy-запросов и закрытии порта, ответсвенного за регистрацию действий пользователей (accounting). Проксирование отключено, поскольку наш RADIUS-сервер будет единственным ответсвенным за авторизацию пользователей, находящихся в радиусе действия беспроводной сети предприятия, а особого смысла держать открытым UDP-порт 1813 нет, поскольку этот сервис (`radacct`) все равно большинством беспроводных роутеров не используется. В моём случае, RADIUS-сервер настроен только на обработку запросов на аутентификацию и авторизацию по стандартному UDP-порту 1812 (блок `listen {...}`)

> За протоколом RADIUS зарезервированы два UDP-порта, один для accounting, а второй для authentication/authorization, причём раньше им использовался другой диапазон портов, о чём сохранились свидетельства в файле `/etc/services`:
>
> ```
> $cat /etc/services | grep RADIUS
> #radius         1645/udp   #RADIUS authentication protocol (old)
> #radacct        1646/udp   #RADIUS accounting protocol (old)
> radius          1812/udp   #RADIUS authentication protocol (IANA sanctioned)
> radacct         1813/udp   #RADIUS accounting protocol (IANA sanctioned)
> ```

Всего в этом конфигурационном файле содержится 5 директив $INCLUDE. Первая подгружает файл clients.conf, где перечислены адреса, с которых разрешено подключение к нашему серверу. Можно ограничится двумя записями – отдельный адрес для локального хоста и для всей остальной локальной сети. В каждом случае нужно указать секретный ключ, который затем будет широко использоваться, так что можно особо не стараться, выдумывая сложные комбинации символов.

```
client localhost {
ipaddr = 127.0.0.1
secret = testinpass
}

client 192.168.0.0/16 {
secret = testinpass
}
```

Следующий подгружаемый файл policy.conf содержит своего рода подпрограммы на языке unlang, которые можно будет вызывать из других конфигурационных файлов. Поскольку ни одна из этих подпрограмм в моих настройках не используется, то я ограничился одним пустым блоком:

```
policy {
}
```

Следующие две директивы $INCLUDE содержатся внутри блока modules {...} и предназначены для считывания конфигурационных файлов из директории /etc/raddb/modules с настройками подгружаемых модулей freeRadius. Сами по себе модули – это обычные разделяемые библиотеки с приставкойrlm_ (расшифровывается как Radius loadable module), и должны поэтому располагаться в стандартном для библиотек месте, известном или из переменной окружения LD_LIBRARY_PATH или из файла /etc/ld.so.conf, например, /usr/lib. Настройки же модуля rlm_eap, ответственного за обработку EAP процедур, вынесены в файл подгружаемый отдельно $INCLUDE eap.conf, не столько потому, что этот модуль настолько важен (хотя и поэтому тоже), сколько из-за того, что там во вложенных блоках содержатся настройки других модулей связанных с EAP, например, rlm_eap_ttls, rlm_eap_peap и т.д. Упрощённую версию этого файла я позаимствовал с сайта [15], и с его помощью настраиваются только те EAP алгоритмы, которые нам в дальнейшем действительно понадобятся – TTLS и PEAP (о нём речь пойдёт ниже).

```
eap {
    default_eap_type = peap
    timer_expire = 60
    ignore_unknown_eap_types = no
    cisco_accounting_username_bug = no
    max_sessions = 2048
    tls {
        certdir = ${confdir}/certs
        cadir = ${confdir}/certs
        private_key_password = whatever
        private_key_file = ${certdir}/server.pem
        certificate_file = ${certdir}/server.pem
        CA_file = ${cadir}/ca.pem
        dh_file = ${certdir}/dh
        random_file = ${certdir}/random
        fragment_size = 1024
        include_length = yes
        check_crl = no
        cipher_list = "DEFAULT"
        make_cert_command = "${certdir}/bootstrap"
    }
    ttls {
        default_eap_type = mschapv2
```

```
        copy_request_to_tunnel = yes
        use_tunneled_reply = yes
        virtual_server = "inner-tunnel"
    }
    peap {
        default_eap_type = mschapv2
        copy_request_to_tunnel = yes
        use_tunneled_reply = yes
        virtual_server = "inner-tunnel"
        }
    mschapv2 {
    }
}
```

К присутствию в этом блоке двух других вложенных секций `tls{...}` и `mschapv2{...}`, да и вообще к содержимому всего этого файла, следует относится как своего рода чёрной магии. Ясно, по крайней мере, что блок `tls{...}` (хотя сам метод EAP-TLS нам не понадобится) содержит важную информацию о расположении серверных сертификатов, без которых, разумеется, ни о каких TTLS каналах речи быть не может. Эти сертификаты (вместе с корневым CA и одним клиентским) будут автоматически сгенерированы скриптом `bootstrap` при первом старте сервера freeRadius в отладочном режиме (т.е. с флагом `-X`). Если же вы захотите подписать эти сертификаты своим собственным CA, то вам (как рекомендуют сами разработчики) необходимо удалить каталог `/etc/raddb/certs` и вместо него создать новый, куда поместить свой корневой сертификат и снова запустить freeRadius с флагом `-X`. Впрочем, для наших целей будет достаточно автоматически сгенерированных сертификатов.

Сама загрузка библиотечных модулей происходит в “ленивом” режиме, т.е. подгружаются только те модули, которые реально используются в работе freeRadius и только тогда, когда это необходимо. Так что, хотя при установке freeRadius, директория `modules` заселена достаточно плотно, на вычислительных ресурсах это никак не отражается. Тем не менее, чтобы просто проиллюстрировать, какие именно модули и конфигурационные файлы действительно используются, я её несколько подчистил, и привожу её листинг на Рис. 1.

И наконец с последним `$INCLUDE` начинается самое интересное – подгружается директория `/etc/raddb/sites-enabled`, где содержатся конфигурационные файлы ”виртуальных серверов которые как раз и отвечают за логику обработки клиентских запросов. Это относительно свежая функциональность, появившаяся только в версии 2.0. С её помощью внутри одного сервера freeRadius можно организовать несколько виртуальных серверов (в традиционном понимании), т.е. с независимой друг от друга конфигурацией, прослушивающих каждый свой UDP-порт и т.д. Но на практике ”виртуальные сервера”в freeRadius применяются, чтобы обрабатывать различные типы запросов с клиентов различным образом. Например, можно заметить, что в файле `eap.conf`, пакеты, полученные по TTLS (или PEAP) туннелю, должны передавастся виртуальному серверу `inner-tunnel`. Каждому виртуальному серверу соответствует одноименный файл в директории `/etc/raddb/sites-enabled`, хотя на самом деле там расположены только символьные ссылки на файлы в директории `/etc/raddb/sites-available` – предполагается, что так несколько проще переконфигурировать freeRadius, просто переадресуя ссылки. В обязательном порядке должен присутствовать по крайней мере файл `default`, который служит обработчиком по-умолчанию. Для простоты оба этих виртуальных сервера (`default` и `inner-tunnel`) у меня сконфигурированы практически идентично. При этом файл `default` я значительно сократил, удалив все блоки, отвечающие за проксирование запросов и обработку accounting информации (поскольку наш сервер всё равно эти пакеты игнорирует), оставив только то, что имеет отношение к авторизации и аутентификации пользователей, так что в итоге этот файл выглядит таким образом:

```
authorize {
        preprocess
        mschap
        eap
```

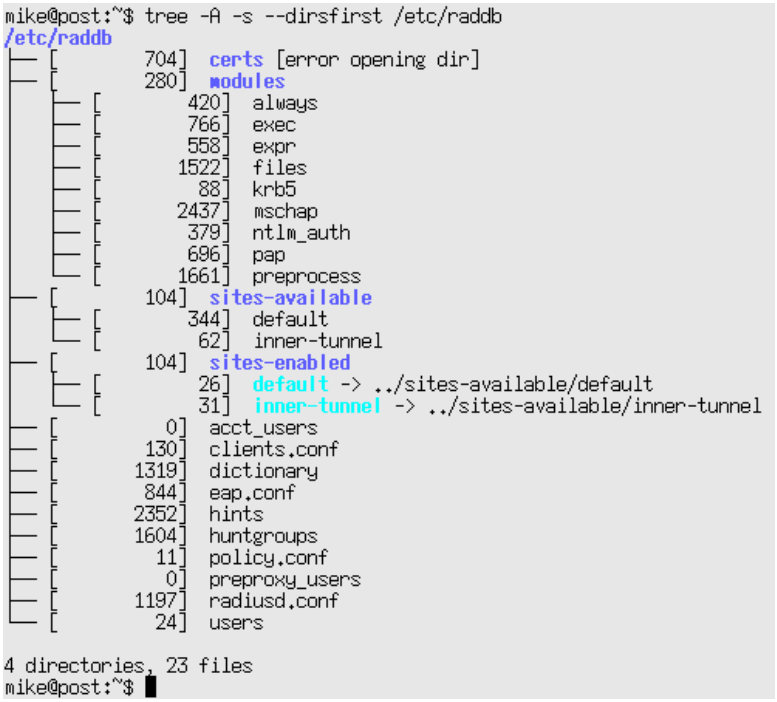

Рис. 1: Листинг директории с конфигурационными файлами freeRadius. Секретная информация содержится только в подкаталоге certs, поэтому его просмотр запрещён простому пользователю.

```
        files
}

authenticate {
        Auth-Type PAP {
                krb5
        }
        Auth-Type MS-CHAP {
                mschap
        }
        eap
}

post-auth {
        noop
}
```

Затем этот текст целиком вставляется в файл `inner-tunnel` с помощью всё той-же директивы `$INCLUDE`:

```
server inner-tunnel {
       $INCLUDE sites-enabled/default
}
```

Осталось совсем немного – понять, как эта штука работает? Последовательность обработки протоколом RADIUS запросов на вход в сеть не совсем тривиальна (отличается от традиционной логики "сначала аутентифицируем, а потом раздаем права"). Вначале запрос полученный каким-то из виртуальных серверов проверятся последовательно всеми модулями, перечисленными в блоке `authorize`, пока какой-то из этих модулей не заявит, что он должен обслуживать этого пользователя, и не выставит ему необходимые атрибуты, включая атрибут `Auth-Type`. Потом уже, в зависимости от выставленного значения, для аутентификации этого пользователя выбирается какой-то один подходящий модуль из перечисленных в блоке `authenticate{...}`, а весь процесс аутентификации завершается пустым блоком `post_auth`. Т.е. процесс перебора начинается с модуля `rlm_preprocess`, который как раз

подгружает статические словари, считывая конфигурационные файлы /etc/raddb/hints и /etc/raddb/huntgroups, и с их помощью преобразует клиентский запрос, полученный с сети, в пригодный для дальнейшей обработки вид, а последним рубежом является модуль rlm_files. Как раз этот модуль проверяет логин пользователя по текстовым файлам данных /etc/raddb/users, /etc/raddb/acct_users и /etc/raddb/preproxy_users. Два последних файла нас не интересуют, так что их можно оставить пустыми, а вот файл /etc/raddb/users как раз потребует доработки. Исторически, предки freeRadius использовали этот файл для хранения паролей пользователей, но для нас важно, чтобы он состоял из единственной записи:

```
DEFAULT Auth-Type = PAP
```

Эта древняя магическая руна ..., ой, простите, эта строка означает, что всем пользователям, для которых атрибут Auth-Type не был установлен ранее, назначается механизм аутентификации PAP, т.е. простая проверка текстового пароля. Если же посмотреть в секцию authenticate, то можно догадаться, что проверка пароля будет проделана модулем rlm_krb5. Фактически всё это эквивалентно выполнению процедуры kinit для выбранного пользователя с полученным от него текстовым паролем.

Т.е. теперь можно будет попытаться взлететь. Но для работы с Heimdal-Kerberos, нужно будет ещё создать специального сервисного принципала для freeRadius и экспортировать его ключ в keytab-файл.

```
kadmin -l add -r radius/server.example.ru
ktutil get radius/server.example.ru
```

Пара полезных советов:

- Имя сервисного принципала и используемый keytab-файл следовало бы внести в конфигурационный файл /etc/raddb/modules/krb5, но в принципе, это не обязательно, его можно сделать пустым:

  ```
  krb5 {
  }
  ```

  Всё равно в качестве последнего резерва (если все введённые там параметры не помогли) модуль rlm_krb5 будет использовать системный keytab-файл /etc/krb5.keytab и имя сервисного принципала составленного из слова radius и имени сервера.
- После того, как сервер RADIUS будет приведён в рабочее состояние, лучше всё же создать под него выделенного системного пользователя и группу (тем более, что порт 1812 не является привилегированным) и сообщить эти значения freeRadius, добавив куда-нибудь в начало файла /etc/raddb/radiusd.conf записи вида:

  ```
  user = radius
  group = radius
  ```

  Следует позаботиться при этом, чтобы все задействованные в настройках freeRadius директории и файлы были доступны на чтение (включая файл /etc/krb5.keytab!) и запись этому пользователю или группе.

Для тестирования системы, запуск будем осуществлять от имени суперпользователя, это упростит отладку.

```
#radiusd -X
```

Флаг командной строки -X помимо того, что запускает процесс в консольном (foreground) режиме, переключает выдачу информационных и отладочных сообщений с системного лог-файла, сконфигурированного выше, на консоль. В процессе первого запуска некоторое время будут генерироваться TLS-сертификаты, потом после перечисления загруженных модулей и листинга их конфигурационных файлов (информация, безусловно, полезная для

системного администратора, но весьма объёмная, поэтому я её не буду приводить), должна появиться надпись:

```
Listening on authentication address * port 1812
Ready to process requests.
```

После этого всё готово, чтобы приступить к испытаниям нашего RADIUS-сервера!

5 Тестируем FreeRADIUS совместно с Heimdal

Всё же, вначале лучше проверить логику работы RADIUS-сервера в локальном режиме с помощью тестовых утилит, а только потом браться за подключение беспроводных роутеров. Т.е. для начала все проверки можно будет проделать на том же сервере, где работает freeRadius, если только там имеются подходящие утилиты.

Таких утилит будет две – `radtest` из комплекта freeRadius, и `eapol_test` из комплекта `wpa_supplicant`. Так получилось, что каждая из них по-отдельности позволяет тестировать функциональность одного из двух сконфигурированных нами виртуальных сервера – `default` и `inner_tunnel` соответственно.

Начнём, разумеется, с `default`. Лучше иметь под рукой два терминала – один, где работает freeRadius в отладочном режиме, и второй, с которого производится тестирование.

```
# radtest -t pap idiot DeBiLL localhost 10 testinpass
Sending Access-Request of id 110 to 127.0.0.1 port 1812
        User-Name = "idiot"
        User-Password = "DeBiLL"
        NAS-IP-Address = 192.168.1.252
        NAS-Port = 10
        Message-Authenticator = 0x00000000000000000000000000000000
rad_recv: Access-Accept packet from host 127.0.0.1 port 1812, id=110, length=20
```

Т.е. нами запрошен алгоритм аутентификации PAP (флаг `-t pap`), для пользователя с таким-то логином и паролем (мы договаривались, что после тестирования вы его удалите), на локальном хосте, с NAS-портом 10 (годится любое численное значение) и локальным паролем `testinpass` (он был сконфигурирован ранее в файле `/etc/raddb/clients.conf`). `Access-Accept` означает, что аутентификация прошла успешно, а `Access-Reject` – нет.

В окне сервера информация будет более многословной, но в итоге они должны закончиться вот этим:

```
Sending Access-Accept of id 110 to 127.0.0.1 port 34683
Finished request 0.
Going to the next request
Waking up in 4.9 seconds.
Cleaning up request 0 ID 110 with timestamp +1546
Ready to process requests.
```

Заметим, что потребовался всего один запрос на аутентификацию пользователя. По крайней мере это означает, что виртуальный сервер `default` работает нормально и соединяется с сервером Kerberos для аутентификации пользователей. После этого можно будет переходить к более практически важному тесту.

Этим тестом будет утилита `eapol_test`, которая, как подсказывает её название, позволяет проверить аутентификацию пользователей в рамках EAP-методов. Нам потребуется сочинить для неё текстовый конфигурационный файл, который потом (с небольшими доработками) можно будет использовать и для тестирования подключений в беспроводном режиме с помощью `wpa_supplicant`. Сам конфигурационный файл (назовём его `eapol.conf`) выглядит таким образом:

```
network={
        ssid="example"
        key_mgmt=WPA-EAP
        proto=WPA2
```

```
        identity="idiot"
        password="DeBiLL"
        eap=TTLS
        phase2="auth=PAP"
}
```

Беспроводной идентификатор `ssid` можно пока оставить любым, параметры `key_mgmt` и `proto` совместно устанавливают режим WPA2-Enterprise, а параметры `eap` и `phase2` позволяют выбрать конкретный метод EAP и механизм проверки пароля внутри EAP (замечу, что согласно [13] внутри TTLS-канала помимо PAP разрешено выбирать CHAP, MSCHAP или даже рекурсивно вызывать другой EAP механизм). После этого запустим саму проверку:

```
#eapol_test -c eapol.conf -s testinpass
...
MPPE keys OK: 1  mismatch: 0
SUCCESS
```

При этом вывод будет довольно многословным, главное чтобы всё завершилось фразой `SUCCESS`, а не `FAILURE`. Сам же процесс установки защищённого соединения и проверки методом EAP оказывается многоступенчатым и требует 7 шагов, как можно судить по отладочной информации freeRadius (её я тоже подсократил, оставив только последние строки):

```
Sending Access-Accept of id 6 to 127.0.0.1 port 52685
        MS-MPPE-Recv-Key = 0x8be20c5c3ef447d72091252816b2b20c3d587b95755d21b10d9f3fd80118319f
        MS-MPPE-Send-Key = 0x6f4f11ad271fe42c3b7b43b5087457a5b919e4955b85192a72c8b3b2f6fa3a29
        EAP-Message = 0x03060004
        Message-Authenticator = 0x00000000000000000000000000000000
        User-Name = "idiot"
Finished request 9.
Going to the next request
Waking up in 4.9 seconds.
Cleaning up request 3 ID 0 with timestamp +5399
Cleaning up request 4 ID 1 with timestamp +5399
Cleaning up request 5 ID 2 with timestamp +5399
Cleaning up request 6 ID 3 with timestamp +5399
Cleaning up request 7 ID 4 with timestamp +5399
Cleaning up request 8 ID 5 with timestamp +5399
Cleaning up request 9 ID 6 with timestamp +5399
Ready to process requests.
```

Как только вы будете удовлетворены результатом этого теста, можно будет вводить в эту цепочку ещё одно звено – беспроводной маршрутизатор.

6 Испытания на практике с беспроводным маршрутизатором WL-500gP V2

Я уже писал, почему мною была выбрана модель WL-500gP V2. Но, на самом деле, требуемые модификации однотипны и применимы для любого роутера (лишь бы там имелась необходимая функциональность). Проще всего эти настройки проиллюстрировать рисунком со снимком экрана, где изображён web-интерфейс маршрутизатора WL-500gP V2 (Рис. 2).

Для дальнейшего тестирования нам понадобится уже устройство с беспроводным оборудованием. Я для этой цели использовал ноутбук с операционной системой Линукс, просто потому что с его помощью мне удалось собрать гораздо больше отладочной информации, чем, скажем, с мобильного телефона. Но по крайней мере, это устройство должно быть настроено на беспроводную работу и быть в физической досягаемости от беспроводной точки. Т.е. команда `iwlist` должна видеть беспроводную точку (у меня она, как видно из Рис. 2, называется `suslik`):

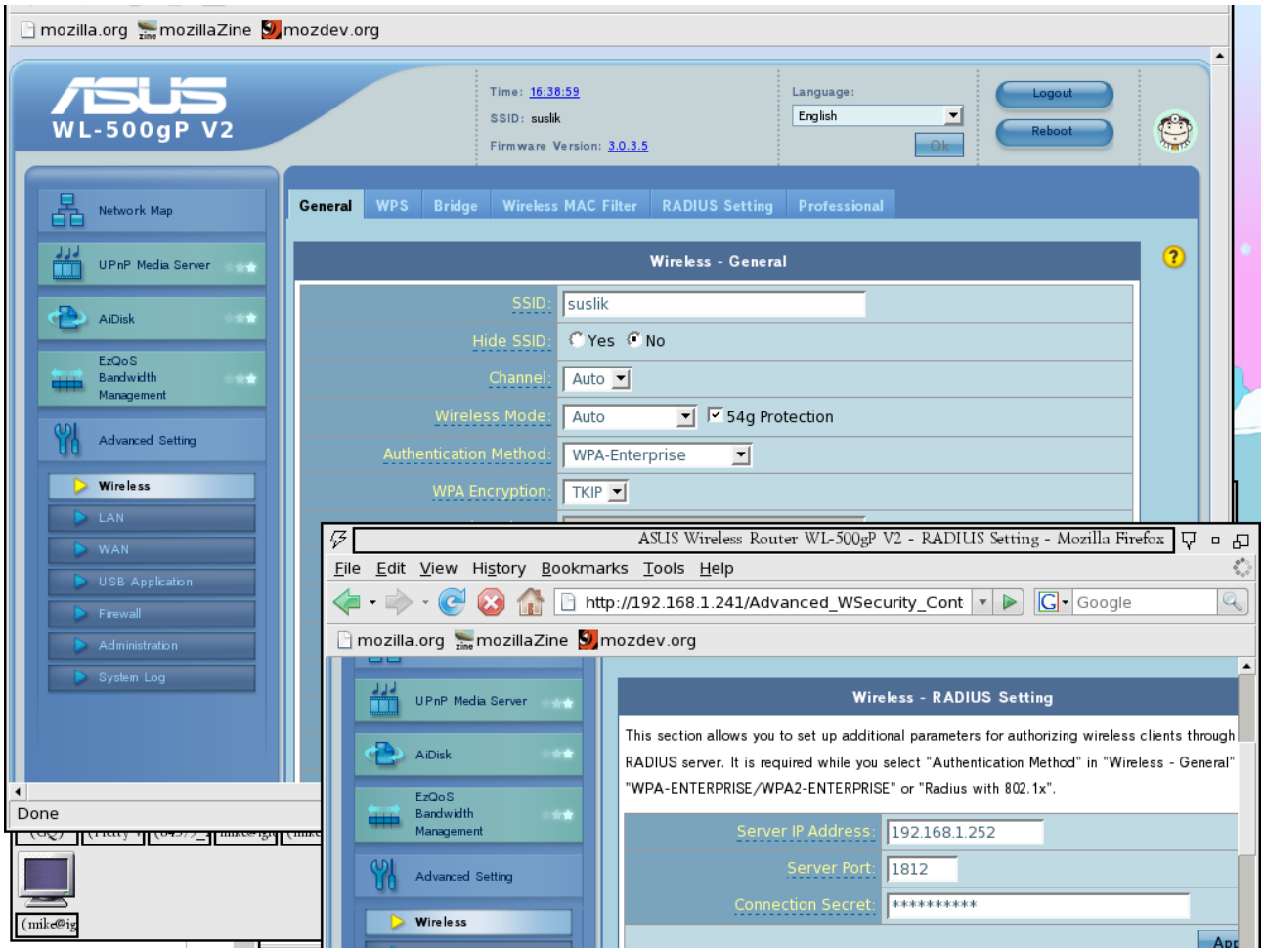

Рис. 2: Настройка маршрутизатора WL-500gP V2. Требуется выбрать метод WPA аутентификации, адрес RADIUS-сервера и секретную фразу для подключения к нему.

```
#iwlist wlan2 scan
wlan2     Scan completed :
          Cell 01 - Address: 48:5B:39:E7:DF:88
                    ESSID:"suslik"
                    Protocol:IEEE 802.11g
                    Mode:Managed
                    Frequency:2.412 GHz (Channel 1)
                    Quality:70/100  Signal level:-51 dBm  Noise level:-96 dBm
                    Encryption key:on
                    Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 11 Mb/s; 18 Mb/s
                              24 Mb/s; 36 Mb/s; 54 Mb/s; 6 Mb/s; 9 Mb/s
                              12 Mb/s; 48 Mb/s
                    Extra:bcn_int=100
                    Extra:atim=0
                    IE: WPA Version 1
                        Group Cipher : TKIP
                        Pairwise Ciphers (1) : TKIP
                        Authentication Suites (1) : 802.1X
```

По крайней мере информация совпадает, с той что была введена на роутере. После этого нужно создать конфигурационный файл `/etc/wpa_supplicnt.conf`, взяв за основу тестовый файл `eapol.conf` и включив туда настройки беспроводной точки доступа.

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=100
eapol_version=1
ap_scan=1
fast_reauth=1

network={
  scan_ssid=0
  ssid="suslik"
  proto=WPA
  key_mgmt=WPA-EAP
  pairwise=CCMP TKIP
  group=CCMP TKIP
  eap=TTLS
  phase2="auth=PAP"
}
```

Предполагается, что сам `wpa_supplicant` запускается при старте операционной системы системы командой, типа:

```
wpa_supplicant -D wext -c /etc/wpa_supplicant.conf -i wlan2
```

а непосредственный выбор сети (из перечисленных в конфигурационном файле) и подключение к ней будет производится непривелегированным пользователем, для чего доступ к контролирующему интерфейсу `wpa_supplicant` предоставляется всем пользователям из группы users (gid=100). Заметьте, что информация о пароле пользователя из конфигурационного файла удалена. Эти данные пользователь может потом ввести самостоятельно при подключении к сети, используя какой-то из имеющихся пользовательских интерфейсов – консольный `wpa_cli` или графический `wpa_gui` (см. Рис. 3). Свидетельством успешного завершения процесса подключения к сети будет появление в окне `wpa_gui` надписи `COMPLETED`. Если же этого почему-то не происходит, то нужно ещё раз убедится в совпадении конфигурации `wpa_supplicant.conf` и настроек беспроводной точки доступа. Также может иметь значение выбранный при старте `wpa_supplicant` драйвер беспроводной сети (параметр командной строки `-D wext`) и интерфейс (`-i wlan2`). На самом деле среди всех драйверов, поддерживаемых программой `wpa_supplicant`, драйвер `wext` практически единственный работоспособный в режиме WPA-Enterprise. Его и нужно выбирать, если вы работаете с современным ядром Linux и с драйверами беспроводного оборудования, основанных на модуле `mac80211`, а так же с последними версиями `ndiswrapper`.

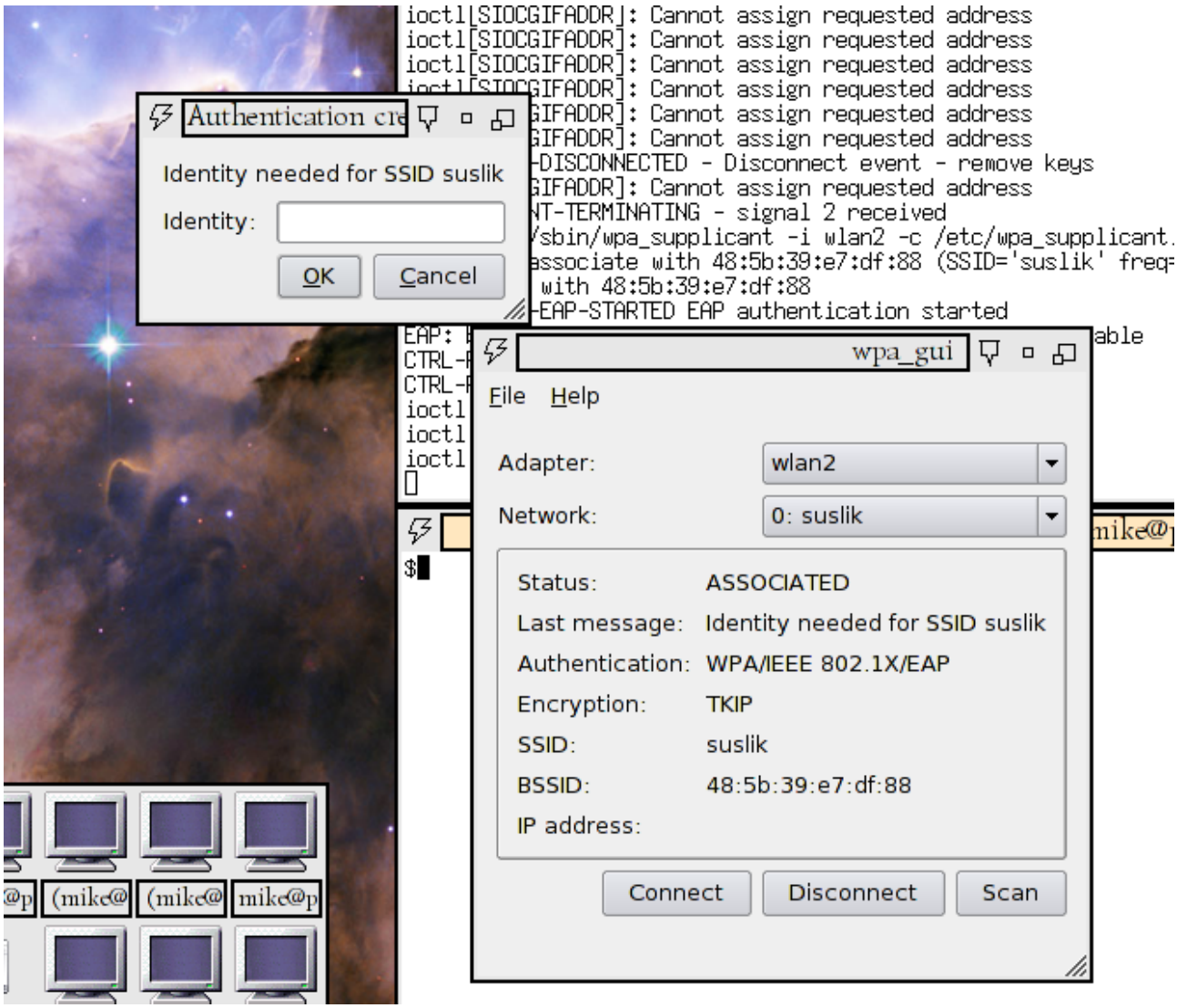

Рис. 3: Подключение к беспроводной сети используя wpa_gui. Интерфейс выглядит не слишком привлекательно, но контакт есть!

7 Настройка проверки подлинности пароля по протоколу MSCHAPv2

У метода EAP-TTLS, рассмотренного выше, несмотря на его безопасность и криптографическую стойкость есть один, но очень существенный недостаток – он не поддерживается продукцией компании Майкрософт. Т.е. вы можете, конечно, заняться настройкой клиентских компьютеров, установив там windows-версию того же `wpa_supplicant`, а также его графического тулкита QT, и заняться обучением пользователей обращаться с ними. Задача эта, как вы сами понимаете, не из лёгких. Проще будет расширить функциональность нашего freeRadius сервера, чтобы Windows-машины могли работать с ним, используя только установленное по умолчанию программное обеспечение.

А установленное по умолчанию программное обеспечение, в смысле ОС Windows, поддерживает всего два EAP механизма. Один из них – упоминавшийся ранее EAP-TLS. Но он нам не подходит, поскольку требует наличия клиентских сертификатов. Т.е. это все равно сводится к развёртыванию дублирующей системы аутентификации пользователей, теперь уже средствами PKI. А вот второй механизм – PEAP/MSCHAPv2 можно вписать в нашу связку, состоящую из freeRadius и Heimdal-Kerberos, хотя это может потребовать обновления паролей пользователей.

Не претендуя на академическую строгость, я составил небольшой неформальный справочник с переводом птичьего языка, который будет применяться в дальнейшем.

Хэш – результат действия хэш-функции, т.е. функции, которая для блока данных произвольной длины выдаёт короткую последовательность байт с фиксированной длиной (можно считать его целым числом). Понятно, что такое соответствие неоднозначно, один и тот же хэш соответствует множеству блоков данных, но с другой стороны требуется, чтобы вычислительная сложность нахождения любого из этих обратных значений (коллизий) была велика. Можно представлять, что хэш – это своего рода контрольная сумма, дайджест, выжимка или резюме исходных данных.

MD4 – один из вариантов хэш-функций семейства алгоритмов MD (расшифровывается, что интересно, как Message Digest – дайджест сообщения). Длина полученного значения всего 16 байт (128 бит). Это весьма криптографически слабый способ, поскольку эта длина, по меркам современных вычислительных мощностей, весьма короткая, и для его взлома (нахождения коллизии) требуется немного времени – порядка часа.

NT-Hash или NT-Key – это хэш (разумеется, длиною 16-байт), полученный из пароля пользователя, записанного в Unicode, применением к нему функции MD4. Используется для хранения пользовательских паролей в семействе операционных систем Windows, начиная с версии NT4.

CHAP или CHallenge Authentication Protocol (протокол проверки пароля вызовом) – протокол удалённой проверки пользовательского пароля. В том или ином виде эта идея присутствует во всех способах проверки идентичности пользователя по сети. В самом примитивном варианте, сервер шлёт клиенту вызов (challenge, nonce) – случайный блок данных. Клиент, имея введённый от пользователя пароль, каким-то условленным способом шифрует этот блок данных паролем пользователя и отсылает результат (responce, отзыв) назад. Сервер, зная пароль пользователя по своим базам данных, проделывает ту же самую процедуру и сравнивает свой результат с полученным от клиента. Совпадение результатов подтверждает правильность введённого пользователем пароля. При этом отпадает необходимость в передаче пароля пользователя в открытом виде по возможно небезопасной сети.

MSCHAP , версий v1/v2 – варианты CHAP, где в качестве ключа используется NT-Hash, откуда и приставка MS в названии. Если вариант v1 напоминает примитивный способ, описанный ваше, то v2 более криптографически защищён, за счёт увеличения шагов, требуемых для проверки пароля, при этом challenge генерируется как сервером, так и клиентом.

DES (digital encryption standard) – "стандарт" алгоритма шифрования, блочный шифр с длиною ключа 7-байт (56 бит). Что интересно, был принят в качестве официального стандарта американским правительством ещё в 1976 году, и только после этого такой тип шифровальных алгоритмов и их криптографическая стойкость стала предметом интереса со стороны научного сообщества. В настоящее время, опять же ввиду небольшой длины ключа, считается небезопасным, и в значительной степени вытеснен своим преемником – AES, но его разновидность TripleDES (3DES, последовательное шифрование-дешифрование тремя разными 7-байтными ключами) по-прежнему актуален.

NTLM (NT LAN Manager) – можно считать конкретной реализацией CHAP, используемой в основном для собственных нужд внутри операционных систем семейства Windows. В качестве вызова используется случайная 8-байтная последовательность данных. Ответ получается путём параллельного шифрования этой последовательности тремя 7-байтными ключами с помощью алгоритма DES и объединением результата в одну строку (т.е. responce будет иметь длину 3*8=24 байта). В версиях операционных систем Windows начиная с NT4, в качестве трёх шифровальных ключей используется NT-Hash (16 байт), дополненный 5 нулями до 21 байта и поделённый на 3 части, т.е. как раз получается три ключа длиной по 7 байт. Впоследствии алгоритм работы NTLM усложнялся, в настоящее время придуманы алгоритмы NTLMv2 и NTLM-Session, но и в современных системах Windows описанный выше алгоритм (NTLMv1) всё ещё используется.

PEAP (расшифровывается как Protected/Защищённый EAP) – это разработка компании Майкрософт и ещё одна разновидность EAP, по своему устройству напоминает TTLS. На этот раз компания Майкрософт отступила от своих принципов "небольших несовместимых модификаций" и устройство PEAP опубликовано в серии IETF-drafts [17, 18]. Так что создание PEAP-туннеля не является проблемой и реализовано в freeRadius в качестве одного из подмодулей `rlm_eap` (см. листинг `eap.conf` выше, где этот туннель уже сконфигурирован). А основную сложность представляет алгоритм аутентификации MSCHAPv2, на котором настаивает операционная система Windows. Сложность эта двоякого рода – во-первых, где взять NT-Key, и во-вторых, как этот NT-Key передать в freeRadius?

Понятно, что если пароль пользователя хранится в открытом виде, то вычислить по нему NT-Hash не представляет особого труда. Именно на такой режим работы и рассчитывает модуль `rlm_mschap` из пакета freeRadius, т.е. предполагается, что открытый текстовый пароль пользователя можно будет извлечь из файла `/etc/raddb/users` или другого типа баз данных (SQL/LDAP). Но Kerberos не предусматривает хранение пользовательских паролей в открытом виде. Вместо этого из него можно будет извлечь NT-Hash непосредственно, но это зависит от настроек самого Kerberos.

Каким именно образом, в смысле типа хэшей, хранятся пароли пользователей в Heimdal-Kerberos конфигурируется либо для всего KDC целиком или для каждого пользователя по-отдельности, причём определённые типы алгоритмов могут быть запрещены как ненадёжные при сборке Heimdal-Kerberos дистрибутерами. NT-Hash, как вы можете судить по-справочнику, – не самый надёжный из хэшей. В Heimdal-Kerberos этот тип хэшей носит название arcfour-hmac-md5. Сразу признаюсь, что я не знаю какое отношение все эти три криптографических алгоритма, начиная с alleged RC4, имеют к NT-Hash, но факт остаётся фактом. Проверить же, что этот тип хэша активирован для какого-то из пользователей на вашем сервере, можно командой kadmin:

```
# kadmin -l list -l mkondrin
            Principal: mkondrin@HPPI.TROITSK.RU
    Principal expires: never
     Password expires: 1 year
 Last password change: 2011-12-19 09:39:13 UTC
      Max ticket life: 1 day
   Max renewable life: 1 week
                 Kvno: 14
                Mkvno: 0
Last successful login: never
    Last failed login: never
   Failed login count: 0
        Last modified: 2011-12-19 09:39:13 UTC
             Modifier: admin/admin@HPPI.TROITSK.RU
           Attributes:
             Keytypes: des-cbc-md5(pw-salt), des-cbc-md4(pw-salt), des-cbc-crc(pw-
salt), aes256-cts-hmac-sha1-96(pw-salt), des3-cbc-sha1(pw-salt), arcfour-hmac-md5(pw-
salt)
          PK-INIT ACL:
              Aliases:
```

Т.е. последняя запись в строке `Keytypes:` как раз и означает, что NT-Key у этого пользователя есть. В противном случае, придётся разбираться, в каком именно месте запрещён `arcfour-hmac-md5`. Возможно, если база данных KDC импортирована из сервера Kerberos иного типа, то для исправления ситуации достаточно будет просто обновить пользователю пароль.

Теперь осталось только забрать этот хэш с KDC и передать его freeRadius. Но нужно понимать, что протокол Kerberos вовсе не предполагает, что любой желающий может вот так просто прийти и забрать нужный ему ключ. А идея билетиков или абонементов/tickets, на которых построен Kerberos [1], для freeRadius и WPA не совсем подходит. К сожалению.

Поэтому нам придётся писать скрипты. В модуле `rlm_mschap` есть возможность с помощью вызова (по умолчанию не активированного) команды `ntlm_auth` получить требуемый NT-Key с контроллера домена Windows. Т.е. цепочка действий там такова: `ntlm_auth` с опцией командной строки `request-key` запрашивает сервис `winbindd` из локальной установки Samba, чтобы тот по протоколу NTLMv1 запросил ключ пользователя с контроллера домена Windows NT4, на работу с которым должен быть настроен локальный сервер Samba. В случае успеха NT-Key по той же цепочке (если не одно звено при этом не оборвано!) отправляется назад. Вот вместо этого вызова мы и подставим свой скрипт, который будет запрашивать ключи непосредственно у Heimdal-Kerberos.

Для этого нам понадобится не собственно сам протокол Kerberos, а его собрат – kadmin, вспомогательный протокол, используемый для административного контроля за сервером KDC. Именно по этому протоколу происходит извлечение ключей с сервера KDC командой ktutil в локальный keytab-файл.

Скрипт я написал на языке TCL. Назовём его `heim_ntlm_auth.tcl`:

```
#!/usr/bin/tclsh

set kadmin "/usr/sbin/kadmin"
set kt /etc/raddb/certs/xxx
set ktparser /etc/raddb/certs/ktparser.tcl

set radprinc radius/[exec hostname -f]
set radkeytab /etc/krb5.keytab

set principal [string tolower [lindex [split [lindex $argv 0] "@"] 0]]
set challenge [binary format H* [lindex $argv 1]]
set responce [binary format H* [lindex $argv 2]]

package require SASL
package require SASL::NTLM
package require md4
proc ntlm1 { hash nonce } {
```

```
    set res ""
    foreach key [::SASL::NTLM::CreateDesKeys $hash] {
        append res [DES::des -dir encrypt -weak -mode ecb -key $key $nonce]
    }
    return $res
}

set ll [exec ${kadmin} -p ${radprinc} -K ${radkeytab} ext_keytab -k $kt $principal]
catch {radius/post.hppi.troitsk.ru

    set ff [open ${kt} RDONLY]
    source ${ktparser}
    set key [parse_keytab $ff]
}
close $ff
file delete ${kt}
if { [string equal [ntlm1 $key $challenge] $responce] } {
    puts -nonewline "NT_KEY: [md4::md4 -hex $key]"
    exit 0
} else {
    puts "Error"
    exit 1
}
```

Т.е. идея скрипта состоит в извлечении с помощью программы kadmin ключей нужного нам пользователя во временный keytab файл (`/etc/raddb/certs/xxx`), поиска там подходящего типа ключа и возвращении найденного ключа модулю `rlm_mschap`, чтобы тот с его помощью закончил аутентификацию пользователя. Понятно, что все эти действия не вполне безопасы, но нас спасает, что все сервисы (freeRadius и KDC) находятся на одном сервере, так что по сети NT-Hash не передается. Но само по себе по себе право на считывание данных пользователей с сервера Kerberos является достаточно серьезной брешью в безопасности и должно быть специально разрешено для какого-то принципала на сервере Kerberos. Пусть это будет сервисный принципал `radius/server.example.ru`, созданный нами ранее для аутентификации по механизму EAP-TTLS/PAP. Чтобы разрешить ему считывать ключи пользователей, нужно подредактировать файл прав доступа `/var/heimdal/kadmind.acl`, добавив туда строку вида:

```
radius/server.example.ru              get
```

и постараться ограничить доступ к файлу `/etc/krb5.keytab`, где хранятся ключи самого этого сервисного принципала. Поскольку у этого сервисного принципала появились какие-то административные функции, то было бы лучше вынести его ключ в отдельный keytab-файл (отличного от системног) и максимально ограничить доступ к нему. Но при этом надо не забыть внести соответствующие изменения и в конфигурационный файл `/etc/raddb/modules/krb5` и в приведенный выше скрипт. Предполагается, что первые пять директив `set` в этом скрипте подгоняются по месту системным администратором – полный путь к программе kadmin, расположение скриптов и файла для временного хранения ключей пользователя (директория `/etc/raddb/certs` с её ограничениями на доступ как раз самое место для них!), а также имя сервисного принципала и его keytab-файл (в скрипте взяты значения по-умолчанию, принятые в модуле `rlm_krb5`).

Теперь осталось только подправить файл `/etc/raddb/modules/mschap`:

```
#ntlm_auth = "/path/to/ntlm_auth --request-nt-key ....
ntlm_auth = "/etc/raddb/certs/heim_ntlm_auth.tcl %{mschap:User-Name:-
00} %{mschap:Challenge:-00} %{mschap:NT-Response:-00}"
```

и автоматически freeRadius сможет использовать все EAP-алгоритмы, основанные на MS-CHAP! При этом возможность аутентификации по алгоритму EAP-TTLS/PAP по прежнему сохраняется.

Разбор временного keytab-файла с ключами пользователя вынесен в отдельный скрипт `/etc/raddb/certs/ktparser.tcl`. Хотя формат keytab-файлов нигде не стандартизован,

но так получилось, что все открытые реализации Kerberos используют один и тот же формат [19], так что этот скрипт применим как для Heimdal так и для MIT-Kerberos:

```
proc parse_keytab { ff } {
    proc read_octet { stream } {
        binary scan [read $stream 2] S1 length
        set octet [read $stream $length]
        return [list $length $octet]
    }

    #read file version magic. Should be 0x502.
    if { ![string equal [read $ff 2] \x05\x02] } {
        error "Unknown file format!"
    }

        while {![eof $ff]} {
            #read entry length
            set size [read $ff 4]
            # number of parts separated by slashes in principal name
            binary scan [read $ff 2] S1 nparts
            #discard realm name
            read_octet $ff
            #discard principal's name parts
            for {set i 0} {$i < $nparts } {incr i } {
                read_octet $ff
            }
            #discard name type, timestamp and one-byte key version number
            read $ff 9
            #read encoding type and check that it is a NT-hash
            binary scan [read $ff 2] S1 enc_type
            if { $enc_type == 23 } {
                #this is arcfour-hmac-md5! Read and process key block
                set keyblock [read_octet $ff]
                if { [lindex $keyblock 0] != 16 } {
                    #This shouldn't happen
                    error "Wrong length of arcfour-hmac-md5 key!"
                } else {
                    #return the key
                    return -code 0 [lindex $keyblock 1]
                }
            } else {
                #this is not arcfour-hmac-md5! discard keyblock
                read_octet $ff
                #discard 4-byte key version number and continue
                read $ff 4
            }
        }
}
```

Протестировать на практике работоспособность всех этих скриптов можно программой `eapol_test` (поскольку MSCHAP алгоритмы задействованы только внутри EAP, то radtest для этой цели непригоден). В качестве первого приближения воспользуемся алгоритмом EAP-TTLS и аутентификацией методом MSCHAPv1, что не запрещено RFC5281 [13]. Для этого в файле `eapol.conf` нужно изменить одно значение:

```
phase2="auth=MSCHAPV1"
```

Т.е. внутри TTLS-туннеля применяется алгоритм MS-CHAPv1. По крайней мере это позволит отладить работу скрипта. После выполнения программы `eapol_test`, отмотав немного назад консоль, где запущен `radiusd -X` (понятно, что изменения в конфигурационных файлах требуют его перезапуска), можно найти результат работы скрипта с возможными сообщениями об ошибках. Вроде этого:

```
Found Auth-Type = MSCHAP
# Executing group from file /etc/raddb/sites-enabled/default
```

```
+- entering group MS-CHAP {...}
[mschap] Told to do MS-CHAPv1 with NT-Password
[mschap]        expand: %{mschap:User-Name:-00} -> idiot
[mschap]  mschap1: 56
[mschap]        expand: %{mschap:Challenge:-00} -> 563e6f90e40826ba
[mschap]        expand: %{mschap:NT-Response:-00} -> 196c463a6823a2fdeaa79a103de044950e3d98158701
Exec-Program output: NT_KEY: 4A369C0813444E9C889C335B77D9228F
Exec-Program-Wait: plaintext: NT_KEY: 4A369C0813444E9C889C335B77D9228F
Exec-Program: returned: 0
[mschap] adding MS-CHAPv1 MPPE keys
++[mschap] returns ok
```

В данном случае получен положительный ответ, и по трём входным параметрам – имени пользователя, 8-байтного вызова и 24-байтного ответа клиента, найден соответствующий им 16-байтный NT-Hash. Операционные системы Windows используют несколько более сложный алгоритм, с вызовом MSCHAPv2 обернутого в EAP протокол внутри PEAP-туннеля. Его проверку тоже можно реализовать с помощью `eapol_test`, если только изменить несколько настроек в `eapol.conf` (обратите внимание на `autheap=`, что активирует рекурсивный вызов EAP алгоритма):

```
eap=PEAP
phase2="autheap=MSCHAPV2"
```

Но если предыдущие тесты закончились успешно, то и на этом этапе никаких неожиданностей быть не должно, хотя для выполнения аутентификации в этом случае потребуется уже 10 шагов.

Осталось только подключить клиентские компьютеры с ОС Windows. Поскольку наша беспроводная сеть защищена, то подключаться к ней на лету (как к открытой сети) уже не получится, и поэтому, аналогично случаю Linux, предварительно придётся создавать конфигурационный файл для такого рода сетей. Правда, в случае Windows задача несколько упрощается, поскольку можно вызвать мастер настройки из Панели управления и ввести там необходимые параметры, требуемые для входя в сеть. У меня, например, эта настройка особых проблем не вызвала, см. Рис. 4. Мне пришлось, правда, отключить проверку серверного сертификата, поскольку как объяснено в документации freeRadius, ОС Windows требует наличия специальных полей в этом сертификате, да и вообще эта проверка не особо необходима.

8 Итог

Последний вопрос касается совместимости этого решения с другими реализациями Kerberos. Поскольку формат keytab файла одинаков, то это решение будет работать и под Mit-Kerberos и Shishi, с очевидными модификациями как флагов вызова под консольную утилиту kadmin, так и подстройку под имеющиеся там средства ограничения доступа к сервису kadmind. Для MIT-Kerberos схожую функциональность можно реализовать также с помощью стороннего протокола KCRAP [20], в чём-то аналогичному kdigest из пакета Heimdal. Хотя оба эти протокола значительно мощнее и шире, чем то, что было реализовано в этой статье, но для подключения их к freeRadius требуются более серьёзные навыки программирования. В тоже время для взаимодействия с AD-Kerberos необходимо следовать рекомендуемому “длинному” способу через `ntlm_auth` и Samba. Но тут есть один неясный для меня момент. Если программа `ntlm_auth` вызвана с флагом `--request-nt-key`, то естественно ожидать, что возвращать она будет именно NT-Key, а не что-то ещё. Тем не менее, модуль `rlm_mschap` ожидает получить от неё MD4-хэш от NT-Key и именно такое значение возвращает мой скрипт. Возможно, это какая-то ошибка, но на вопрос в списке рассылки авторы freeRadius ответили уклончиво. Будем считать, что на то у них были свои причины. Вместе с тем, жалобы на сложности с аутентификацией через `ntlm_auth` регулярно возникают в их списке рассылки. Возможно, это как-то связано с настройками всех звеньев в цепи, по которой передаётся NT-Hash (контроллер домена, winbind, `ntlm_auth`),

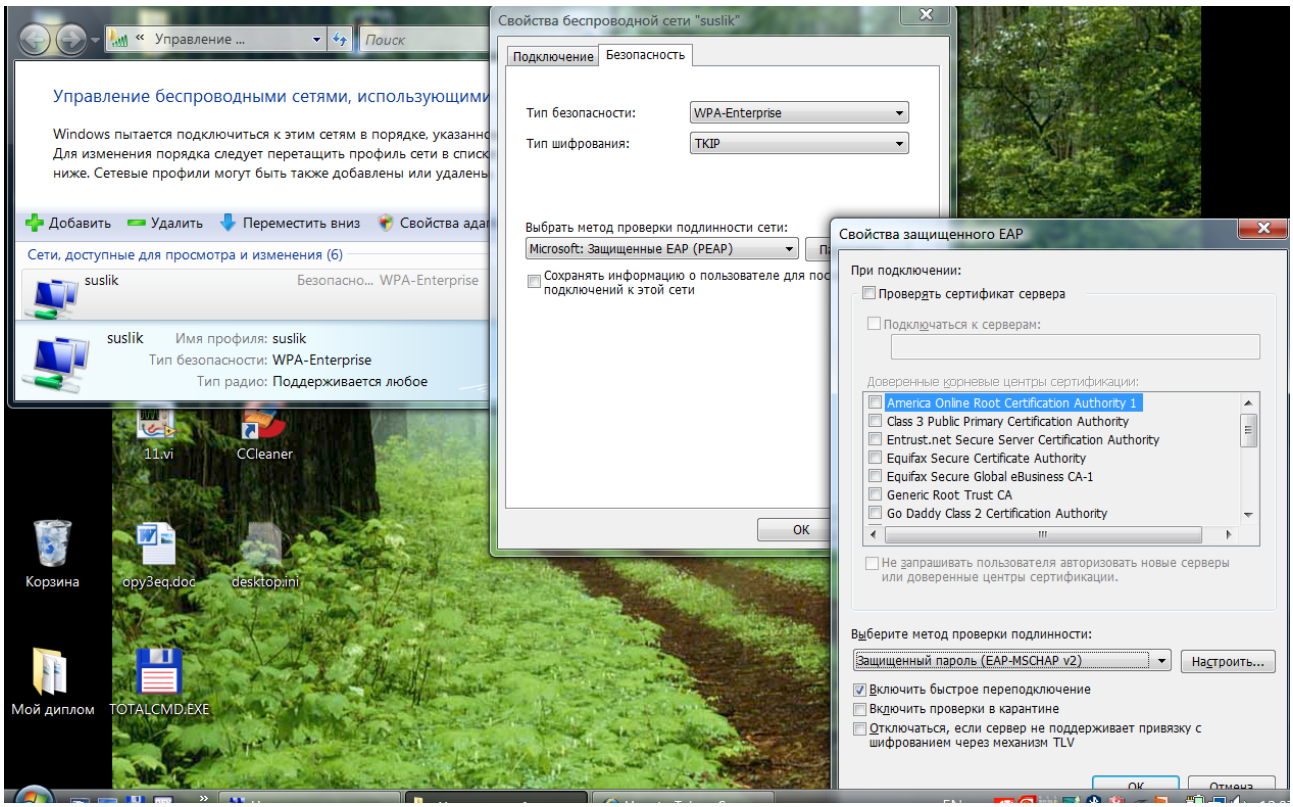

Рис. 4: Настройка беспроводного соединения WPA-Enterprise в Windows-Vista

так что возвращаемое значение в какой-то момент должно (или, напротив, не должно) хэшироваться ещё раз. Для алгоритма MSCHAPv1 это не так важно, поскольку в его случае процесс аутентификации заканчивается после положительного ответа от `ntlm_auth`, а вот в алгоритме MSCHAPv2 возвращаемое значение будет использовано на следующем шаге и тут-то как раз весьма критично, имеем ли мы на руках сам ключ пользователя или его MD4-хэш. Но до конца этот вопрос я не выяснил.

Тем не менее, несмотря на то, что нам удалось настроить пока только два способа аутентификации в freeRadius и Heimdal-Kerberos, но на самом деле это позволяет подключаться к беспроводной сети широкому набору устройств – от ноутбуков с ОС Linux и Windows, до отдельных моделей мобильных телефонов. Дальнейшее подключение других механизмов EAP осуществить не слишком сложно, а имея в наличии саму эту связку Heimdal-Kerberos и freeRadius можно использовать её и в других задачах, где требуется протокол RADIUS, например, в VPN или SIP.

Список литературы

[1] М.Кондрин. Изучаем принципы работы Heimdal Kerberos. *Системный Администратор*, 6, 2005.

[2] М.Кондрин. Развертываем Heimdal Kerberos. *Системный Администратор*, 7, 2005.

[3] FreeRADIUS: The world's most popular RADIUS Server [online].

[4] OSC Radiator RADIUS Server Software [online].

[5] Heimdal: KDIGEST and Heimdal 0.8 release process [online, cited April 2007].

[6] hostapd: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator [online, cited Sat Oct 29 13:50:22 EEST 2011].

[7] freeDiameter [online].

[8] Open1X.org [online].

[9] Linux WPA Supplicant (IEEE 802.1X, WPA, WPA2, RSN, IEEE 802.11i) [online, cited Sat Oct 29 13:50:18 EEST 2011].

[10] ndiswrapper [online, cited 21 November 2010].

[11] mac80211 - Linux Wireless [online].

[12] B. Aboba, L. Blunk, J. Vallbrecht, J. Carlson, and H. Levkowetz. RFC3748: Extensible Authentication Protocol (EAP). Technical report, Network Working Group, June 2004.

[13] P. Funk and S. Blake-Wilson. RFC5281: Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0). Technical report, Network Working Group, August 2008.

[14] D. Simon, B. Aboba, and R. Hurst. RFC51216: The EAP-TLS Authentication Protocol. Technical report, Network Working Group, March 2008.

[15] How to deploy eduroam on-site or on campus [online, cited 05 Oct 2011].

[16] Минимальный RADIUS сервер на freeradius [online].

[17] Vivek Kamath, Ashwin Palekar, and Mark Wodrich. Internet-Draft: Microsoft's PEAP version 0 (Implementation in Windows XP SP1). Technical report, PPPEXT Working Group, October 2002.

[18] H. Andresson, S. Josefsson, G. Zorn, and B. Aboba. Protected Extensible Authentication Protocol (PEAP). Technical report, Internet Engineering Task Force, October 2001.

[19] Heimdal Kerberos: File formats [online, cited Sep 5 22:14:31 2009].

[20] Kerberos Challange-Response Authentication Protocol [online].