Linux без /etc/passwd, /etc/group и /etc/shadow.

1 Зачем?

Начинающий администратор Линукса (вроде меня) по мере того как количество серверов находящихся под его контролем увеличивается, а сервисы начинают расползаться по ним, задумывается о том, как бы внедрить систему централизованного учета пользователей, сервисов и хостов. В самом деле - на одном из серверов крутится мэйлер, который выполняет свою работу для пользователей зарегистрированных на его компьютере, а на соседнем - mysqld, которым пользуются те же люди, что и почтой, но сведения о них хранятся в одной из собственных баз данных mysql. Кроме того, основной инструмент администратора - ssh - создает настолько запутанное распределение публичных и частных ключей на этих серверах (как прилежный администратор на каждом из серверов он помимо root-а держит и свой собственный логин, имя и пароль которого не всегда совпадают), что замена адресов на одном из серверов, способна надолго вывести админа из равновесия.

Вообщем, что-то нужно делать. Например, поставить еще один (;)) сервер, на котором хранились бы логины служебных и реальных пользователей, пароли, а все остальные сервера по мере необходимости 'зарегистрировать' пользователя обращались бы не к своим локальным /etc/passwd, /etc/group и /etc/shadow, а к этому серверу. Реализация такой системы могла бы принести выгоду не только администратору сети (вся информация о пользователях сети хранится на одном сервере, что значительно облегчает контроль над ней), но и ее пользователям, т.к. в качестве побочного эффекта можно реализовать 'sign-once' систему - пользователь вводит пароль один раз при входе в нее, и все сетевые службы внутри сети автоматически становятся ему доступны. О том как такая задача может быть решена будет показано в этой статье.

2 Немного теории.

На самом деле за термином 'регистрация' скрываются две абстракции слабо связанные друг с другом (computer science не зря все же считают наукой - хотя многие и сомневаются - разложение некоторого явления на абстрактные модели оказывается так же успешным, как и в естественных, например, науках). Во первых, система должна убедится, что регистрируемый пользователь именно тот, за кого он себя выдает (это делается с помощью пароля, ключа - не важно), а, во-вторых, система должна получить определенную информацию о пользователе, о том, на какие ресурсы системы он может рассчитывать (для Unix-like систем эта информация включает в себя user/group id, домашний каталог пользователя, его шелл или отсутствие такового). Вообщем-то, как московский милиционер, проверяя ваш паспорт, сначала смотрит на фотографию

убеждаясь что вы это вы, а потом смотрит на прописку, получая таким образом информацию, какие ресурсы он может с вас поиметь.

В дальнейшем два этих действия будем называть аутентификацией и авторизацией (калька с английского, authenticate - проверять подлиннсть, и authorize - предоставлять полномочия). Очевидны, кстати, еще пара различий между этими действиями - во-первых, разглашение информации необходимой для аутентификации (пароли, частные ключи) означает полный провал всей политики безопасности для системы, в то время как информация о том, что пользователь root имеет uid=0, gid=0, home=/root серьезной угрозы не представляет (с той только оговоркой, что злоумышленник должен быть лишен возможности получать всю эту информацию сразу и целиком - например, список ВСЕХ пользователей системы). Во-вторых, авторизация (в отличии от аутентификации) системно-зависима - значения uid/gid мало что говорят о пользователе M$Windows.

Таким образом, раз перед нами стоят две проблемы, то и решать их можно по-отдельности и вообще говоря разными способами.

Хронологически первым ответом была NIS (Network Information Service AKA YellowPages) разработанная Sun в середине 80-х годов, где обе задачи были решены в рамках одного проекта. Помимо обшеизвестных проблем с безопасностью, выявленных в сановской системе rpc, на которой базировалась как NIS так и NFS, к недостакам такого решения можно отнести и ограниченную область применения - оно работает только в Unix-like окружении. Собственно причина этого недостатка понятна - решение одновременно двух задач, из которых только одна (авторизация) системно-зависима, приводит к системной зависимости всего решения целиком.

Сегодня наиболее приемлемым решением является комбинация LDAP (Lightweight-DirectoryAccessProtocol - для авторизации) и PAM (PluggableAuthenticationModule - догадайтесь для чего?). Я не останавливаюсь на нем, так как имеется масса как английской, так и русской документации, описывающих использование этих компонентов для построения централизованной системы регистрации.

Вместе с тем имеется еще одна возможность (которая сейчас не сильно распространена, да и то используется в основном в академических сетях) - аутентификация через Kerberos и авторизация посредством Hesiod. Я не буду сейчас рассказывать почему мне показалось интересным именно такое решение (ну, помимо классического - just for fun (c) Линус Торвальдс, информации в сети не так много - было интересно поэкспериментировать), а сделаю это в конце статьи.

Что они из себя представляют? Если коротко, то Kerberos - сервер аутентификации, характеризующийся сильной криптографической защитой, а Hesiod - информационная система, передающая небольшие текстовые сообщения с помощью ResourceRecords DNS.

3 И немного истории.

Цербер и Гесиод разрабатывались в рамках одного проекта построения распределенной вычислительной системы Athena, запущенного Массачусетским Технологическим Институтом в конце 80-х годов. Проект жив до сих пор и информацию о нем и его компонентах (так же как и сами компоненты) вы можете найти на сайте: . Цербер (если кто не знает, то в древнегреческой мифологии это трехголовый пес, который стерег вход в подземное царство мертвых) стал доступен для свободного скачивания с середины 90-х годов, сейчас это один из наиболее знаменитых компонентов этой

системы, и давно уже живет свой собственной жизнью. В оригинальном Kerberos от MIT была реализована настолько сильная криптографическая защита, что его классифицировали как военную технологию, на которую распространяются законы США об экспорте вооружений. В результате, перед тем как он был выложен в общий доступ отдельные модули из Kerberos'а были изъяты. Впрочем, этот недостаток впоследствии исправили шведские программисты и их версия Kerberos (получившая название Heimdal - тоже какая-то загробная фауна из скандинавской мифологии) доступна по адресу:.

Гесиод (получивший имя от древнегреческого поэта) - менее на слуху, но тем не менее его составные части (в отличии от того-же Цербера) имеются в любом дистрибутиве Linux и в любом сервере Bind, начиная с версии 4.91.

Вооружившись этими сведениями из древнегреческой литературы и скандинавской мифологии приступим к делу. Хочу только предупредить, что в дальнейшем я не рассматриваю реальную рабочую систему, а описываю построение ее игрушечной модели, с которой вы можете поиграться, например, на своем домашнем компьютере и подумать как ее применить на практике.

4 Как отучить систему искать логины в /etc/passwd?

Откуда система знает, что информацию о пользователях нужно искать в файлах /etc/passwd, /etc/group? На самом деле это знает glibc и поскольку эта библиотека является одной из главных библиотек Linux, с которой динамически или статически слинкованы практически все Линуксовые приложения, то эта информация таким образом становится доступна и для всех этих приложений. Причем в glibc механизм получения информации о пользоваетлях, службах, хостах и др. можно достаточно легко настраивать с помощью NameServiceSwitch, технологии перенятой от Sun (информацию о NSS можно найти в `info glibc`). Метод, который glibc использует для пополнения той или иной системной базы данных (помимо passwd,group,shadow к ним относятся services,protocols,hosts и тд) прописан в файле `/etc/nsswitch.conf`. Формат файла прост, к примеру строка `hosts: files dns` означает, что для получения IP-адреса компьютера будет сначала просмотрен файл /etc/hosts, а в случае неудачи - будет послан запрос DNS. Когда glibc читает конфигурационный файл и наталкивается на название метода (к примеру dns), то динамический загрузчик пытается найти в системе библиотеку с именем `libnss_dns.so.X`. Ничего страшного не произойдет, если такой библиотеки найдено не будет, glibc прибегнет к методу записанному следующим или прибегнет к дефолтному методу (который, как правило, и состоит в чтении локальных файлов в каталоге /etc). В стандартной минимальной установке glibc (которая встречается в Slackware) таких методов (также как и библиотек `libnss_*.so.*`) всего 6 - compat, files, dns, hesiod, nis, nisplus (последние два позволяют использовать систему NIS, о которой упоминалось ранее). Для того чтобы добавить новый метод, нужно написать и скомпилировать свой собственный модуль libnss (для этого не требуется перекомпилировать всю glibc целиком), и положить его в такое место в системе, где динамический загрузчик в случае необходимости сможет его найти. Так обычно и поступают - например, для чтения баз данных BerkleyDB, в систему может быть добавлен модуль `libnss_db.so`, или модуль `libnss_ldap.so` - для получения информации по протоколу LDAP.

Таким образом, чтобы получать сведения о пользователях и группах из Гесиода, первое что нужно сделать - это отредактировать `/etc/nsswitch.conf`, заменив или

добавив туда строки:

```
passwd: hesiod files
group: hesiod files
```

(я считаю, что hesiod должен иметь приоритет перед локальными etc-файлами, но при этом сохраняю их как fallback механизм).

Теперь можно приступать к конфигурированию Гесиода.

5 Hesiod

Как я уже говорил ранее, Гесиод передает информацию с помощью DNS. Поэтому для того, чтобы использовать Гесиод у вас должен иметься работающий сервер bind. Для нашего игрушечного примера подойдет простейший кэширующий сервер bind (конфигурационные файлы которого идут вместе с дистрибутивом bind). Единственное, что вам нужно будет сделать - это добавить туда еще одну зону, в которой будем хранить информацию Гесиода, и написать для нее файл зоны, плюс пара зон (прямая и обратная) для dummy интерфейса. Для Гесиода эти две последние зоны не столь важны, но Цербер отказывается запускаться на nonroutable интерфейсе (типа localhost). Поэтому вначале поднимайте этот интерфейс `ifconfig dummy0 192.168.192.1 up` (или воспользуйтесь уже имеющимся в наличии). Мой /etc/named.conf выглядит таким вот образом:

```
options {
        directory "/var/named";
};

zone "." {
        type hint;
        file "example/named.ca";
};

zone "localhost" {
        type master;
        file "example/localhost.zone";
};

zone "0.0.127.in-addr.arpa" {
        type master;
        file "example/named.local";
};

zone "ns.hesiod.info" {
        type master;
        file "example/hesiod";
};


zone "myhome.domain" {
        type master;
```

```
        file "example/myhome.domain";
};

zone "192.168.192.in-addr.arpa" {
        type master;
        file "example/myhome.domain.rev";
};
```

Названия зон не важны - как видите я придумал свой домен myhome.domain и зону Гесиода ns.hesiod.info (совпадать они не обязаны). Файлы зон myhome.domain* в комментариях не нуждаются:

```
$cat /var/named/myhome.domain

\$TTL   86400
@                       IN      SOA     ns.myhome.domain. hostmaster.myhome.domain. (
                                        42              ; serial (d. adams)
                                        3H              ; refresh
                                        15M             ; retry
                                        1W              ; expiry
                                        1D )            ; minimum
                        NS      ns
ns                      A       192.168.192.1

$cat  /var/named/myhome.domain.rev

\$TTL   86400
@       IN      SOA     myhome.domain. hostmaster.myhome.domain.  (
                                      2004092303 ; Serial
                                      28800      ; Refresh
                                      14400      ; Retry
                                      3600000    ; Expire
                                      86400 )    ; Minimum
1       IN      PTR     ns.myhome.domain.
```

А зону Гесиода мы добавляем несколько TXT записей, в которых указываем по одной группе и пользователю (только в качестве примера; общее число пользователей и групп, разумеется, не ограничено). Формат этих записей соответствует формату файлов /etc/passwd, /etc/group (заметьте, что этот пользователь и группа должны иметь имена и идентификаторы не используемые локальными пользователями и группами)

```
@                       IN      SOA     serv.hesiod.info hostmaster.hesiod.info (
                                        42              ; serial (d. adams)
                                        3H              ; refresh
                                        15M             ; retry
                                        1W              ; expiry
                                        1D )            ; minimum
                        IN      NS      serv

;users info
```

```
newhesiodusr.passwd     TXT     "newhesiodusr::2000:1200:test:/:/bin/bash"
2000.uid                CNAME   newhesiodusr.passwd
;groups definitions
hesgr.group             TXT     "hesgr:*:1200:"
1200.gid                CNAME   hesgr.group
```

То что записано после точки в названии Resource Records (гесиодовский тип записи), подсказывает libnss_hesiod как интерпретировать значение этой записи (passwd означает, что в записи указана информация о пользователе, group - о группе). Т.е. в данном примере мы создали пользователя newhesiodusr и группу hesgr к которой принадлежит newhesiodusr. Записи типа CNAME служат в качестве своего рода реверсной зоны для домена Гесиод, с их помощью библиотека hesiod может быстро находить по заданным uid и gid имена пользователей и групп. Помимо использованных в данном примере passwd, group, gid и uid, гесиодовский тип записи может принимать еще несколько предопределенных значений, среди которых интересными на мой взгляд являются две. Запись service может помочь, если вы имеете много нестандартных служб и раздаете их названия через nsswitch (вместо правки локального файла `/etc/services`) - см. соответствующую запись в nsswitch.conf. Запись filsys дает описание файловой системы (соответствующая Resource Record должно принимать значения вида - <имя файловой системы>,<пользователь>,<точка монтирования>:<параметры монтирования>,<удаленный сервер NFS>:<экспортируемый каталог>). По-видимому, во втором случае удается каким-то образом настроить автомонтирование домашних каталогов пользователей, экспортируемых через NFS - см. запись automount в nsswitch.conf.

Осталось создать конфигурационный файл `/etc/hesiod.conf` (он практически стандартный, поменять вам нужно разве что rhs)

```
rhs=.hesiod.info
lhs=.ns
```

, после чего добавить запись о локальном nameserver'е (`nameserver 127.0.0.1`) в `/etc/resolv.conf` (чтобы система могла воспользоваться информацией Гесиода, она, по-крайней мере, должна знать где его искать) и запустить named.

Теперь проверим видны ли гесиодовские записи по DNS:

```
$dig TXT hesgr.group.ns.hesiod.info

; <<>> DiG 9.2.2-P3 <<>> TXT hesgr.group.ns.hesiod.info
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 51689
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 0

;; QUESTION SECTION:
;hesgr.group.ns.hesiod.info.    IN      TXT

;; ANSWER SECTION:
hesgr.group.ns.hesiod.info. 86400 IN    TXT     "hesgr:*:1200:"
```

```
;; AUTHORITY SECTION:
ns.hesiod.info.         86400   IN      NS      serv.ns.hesiod.info.

;; Query time: 2 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Sat Oct  2 00:03:40 2004
;; MSG SIZE  rcvd: 89
$
```

Для упрощения отладки вы можете скачать и установить пакет с Гесиодом с МТИ - там имеется утилитка hesinfo, с которой отслеживать гесиодовские записи несколько удобней - печатать меньше. Вот пример:

```
$hesinfo -bl newhesiodusr passwd
hes_to_bind(newhesiodusr, passwd) expands to
newhesiodusr.passwd.ns.hesiod.info
which resolves to
newhesiodusr:*:2000:1200:test:/:/bin/bash
$
```

Помимо этого, очень полезным может оказаться `man hesinfo`, где можно почитать о допустимых гесиодовских типах и их значениях. В сущности помимо этой man странички, никакой другой информации о Гесиоде мне найти не удалось.

Чтобы окончательно убедится, что Гесиод работает, залогинимся из-под root как newhesiodusr

```
$su
Password:
#login newhesiodusr
Linux 2.4.22.
Last login: Птн Окт  1 21:35:39 +0400 2004 on ttyp2.
No mail.
$id
uid=2000(newhesiodusr) gid=1200(hesgr) groups=1200(hesgr)
$whoami
newhesiodusr
$echo $HOME
/
$echo $SHELL
/bin/bash
$
```

Как видите, система не стала спрашивать у вас пароль, а сразу дала вам bash. Это и неудивительно, так как в записи Гесиода для newhesiodusr был указан пустой пароль. А что будет, если в эту запись добавить пароль (`newhesiodusr:*:2000:1200:test:/:/bin/bash` - после чего необходимо перезапустить named)? Если заглянуть в `/etc/nsswitch.conf`, тот ответ становится более или менее понятен - согласно конфигурации nsswitch теневые пароли glibc будет искать в `/etc/shadow` (строка `shadow: compat` - оно же дефолтное значение в случае отсутствия такой строки). Поскольку в shadow ничего про newhesiodusr не сказано, то система откажется его регистрировать.

В итоге - у нас остается два варианта: по-прежнему хранить теневые пароли в локальных `/etc/shadow` (что неудобно) или обучить системный login пользоваться услугами централизованной системы аутентификации. Однако во втором случае, несмотря на его очевидное удобство, есть одна серьезная проблема, связанная с безопасностью. Но прежде чем обсуждать ее - еще несколько слов о быстродействии. В нашей игрушечной модели, когда демон named обслуживает только локальные запросы особой потери в скорости наблюдаться не должно. Однако при переносе в реальные условия (один named на локальную сеть), падение скорости может оказаться весьма значительным. Например, файловый менеджер на одном из клиентов Гесиода хочет конвертировать uid/gid владельца файлов (хранимые на файловой системе) в их имена, которые ему придется получать по DNS. Снизить нагрузку на сеть и повысить быстродействие в этом случае можно с помощью локального кэширующего сервера DNS или его облегченного варианта - Name Service Caching Daemon (команда nscd - идет вместе с библиотекой glibc). Настройка nscd гораздо проще настройки сервера DNS, в сущности нужно только указать, запросы какой из nss служб nscd должен кэшировать и параметры кэша (его объем и время жизни).

6 Как Цербер проверяет пользователей

Наивный подход к построению централизованной системы аутентификации мог бы быть таким - пользователь печатает свой логин и пароль на своем терминале, а потом пароль сверяется с поролями, хранящимися на сервере. В принципе так работает telnet и именно поэтому от него постепенно отказывются. Прежде чем сравнивать пароль с 'оригиналом', хранящимся на сервере, его нужно туда доставить - а сетевые соединения по-умолчанию не безопасны и для серьезного злоумышленника не составит большого труда перехватить пароль в процессе передачи. Таким образом и для серьезной системы аутентификации отсутствие пересылки паролей по сети - непременное условие.

В Цербере эта задача решается с помощью системы билетиков (tickets). Если вспомнить о чем шла речь в начале статьи, то процедура аутентификации - это функция от двух переменных (имя, пароль) принимающая значения Истина (пароль соответствует имени и это имя известно Церберу)/ Ложь. Кому именно принадлежит это имя (пользователю,службе) при таком подходе безразлично. Поэтому с точки зрения Цербера любая сетевая служба, пользователь, хост требующие своей аутентификации выглядят совершенно одинаково и называются principals. Каждый принципал имеет имя вида `<principal name>/<principal instance>@<KERBEROS REALM>` и пароль, подтверждающий идентичность этого принципала. Realm, в терминологии Цербера, означает зону ответственности каждого из серверов Kerberos. Разделение имени принципала на name и instance позволяет достаточно гибко настраивать контроль доступа пользователей к базе данных Цербера (Цербер поддерживает acl). Характерный пример использования - иметь для отдельных системных пользователей два имени в Цербере - `joeuser@...` и `joeuser/admin@...`. Регистрируясь в Цербере под первым именем (без instance) пользователь получает мимнимальные права по модификации базы данных Цербера - менять свой пароль, просматривать список билетиков, выданных ему Цербером. Во втором случае, эти права расширяются - добавляется, например, возможность добавлять/удалять других пользователей, предусмотренная для принципалов с instance admin. Хосты и службы обычно имеют имена вида `host/<fqdn>@...`, `<service name>/<fqdn>@...`, где fqdn полное сетевое имя компьютера. Список принципалов и их паролей хранится на

сервере и (в параноидальных случаях) редактируется администратором не отходя от этого сервера. Понятно, что взлом этого сервера приводит к полному краху системы безопасности, поэтому рекомендуется не иметь на нем никаких сетевых служб кроме Kerberos и держать на этом сервере по возможности ограниченное число пользователей.

Аутентификация пользователя в Цербере происходит таким образом: предположим, пользователь хочет зарегистрироваться на компьютере, где установлена программа-логин с поддержкой Kerberos. После того как он вводит в логин свое имя, на Цербер отправляется запрос на билетик для принципала с этим именем. Цербер проверяет имя принципала по своей базе данных, и в случае если имя найдено, генерирует по определенным правилам билетик и отсылает его назад, предварительно закодировав его с помощью пароля этого принципала (вместе с билетиком генерируется еще и шифровальный ключ, который может быть в дальнейшем использован для создания шифрованных каналов для этого пользователя - об этом ниже). Программа логин получает этот закодированный билетик и пытается дешифровать его, используя в качестве ключа пароль пользователя, который он к этому времени уже напечатал. Если пароли не совпадают, вместо нормального билетика после дешифровки получается мусор, и в доступе пользователю отказывают. Если же пользователь успешно зарегистрировался в системе, то билетик вместе с шифровальным ключом записываются в директорию /tmp. Полученный таким образом билетик является не просто билетиком, а “единым проездным” (ticket-granting ticket или TGT в терминологии Цербера), поскольку при его предъявлении обладатель “проездного” может получить доступ к другим сетевым службам. Стоит отметить, что срок действия проездного ограничен (как правило - 24 часа), поэтому для его возобновления после этого срока пользователю придется снова вводить пароль и запрашивать Цербер о выдаче билета.

А теперь посмотрим, как Цербер разрешает доступ пользователей к сетевым службам. Предположим, пользователь хочет получить письмо с POP сервера и как мэйл-клиент, так и POP-server поддерживают аутентификацию через Цербер. Клиент разыскивает в /tmp проездной пользователя и отправляет проездной на Цербер вместе с запросом на предоставление сервиса (в данном случае запрос является именем принципала, под которым запрашиваемый сервис зарегистрирован на Цербере, типа - `pop/mail.myhome.domain@...`). Если же проездной не найден, то клиент должен предложить пользователю форму для ввода имени и пароля и действовать согласно ранее описанному сценарию. Цербер в ответ генерирует билетик (в случае если он распознает проездной как действительный и находит в своей базе запрашиваемый сервис), добавляет к нему шифровальный ключ, который уже знает обладатель проездного, и отсылает его клиенту в зашифрованном виде (в качестве шифровального ключа используется пароль сервиса). Мэйл-клиент просто передает полученный билетик POP-серверу. Для его расшифровки POP сервер использует свой пароль (пароли сетевых служб хранятся в файле `/etc/krb5.keytab` на локальном диске компьютера, где работает данная служба) и из расшифрованного билетика узнает какой шифровальный ключ он должен использовать при общении с клиентом (т.е. тот же самый, который был выдан вместе с проездным пользователю). Как видите, вместо пароля по сети путешествует только закодированные билетики, пароли же не покидают сервер Цербера и терминала, на котором регистрируется пользователь. Более того, использование билетиков позволяет прозрачно (т.е. незаметно от пользователя) разрешать доступ пользователю к сетевым службам на основании проездного, полученного им при входе в систему.А в качестве побочного эффекта, у вас есть

возможность средствами Цербера шифровать траффик между клиентом и сервером.

Теперь пора приступать к установке Цербера. Скачиваете исходники с сайта, распаковываете их и запускаете `./configure --prefix=/usr --enable-shared` (второй ключ позволит вам в дальнейшем с меньшей кровью компилировать приложения с поддержкой Kerberos). После чего выполняете стандартный набор команд `make; make install`. Итак Цербер установлен и можно начать запускать его. В этой части я буквально следую рекомендациям install-guide.ps, который упакован вместе с исходниками Цербера, поэтому более подробные разъяснения вы можете прочитать там. Во первых, создаем каталог для хранения баз Цербера `mkdir /usr/local/var/krb5kdc` и, во вторых, сочиняем конфигурационный файл Цербера (`/etc/krb5.conf`)

```
[libdefaults]
        default_realm=MYHOME.DOMAIN
[realms]
        MYHOME.DOMAIN={
                kdc=ns.myhome.domain
                admin_server=ns.myhome.domain
        }
[logging]
        kdc=FILE:/var/log/krb5kdc.log
        admin_server=FILE:/var/log/kadmin.log
        default=FILE:/var/log/krb5.log
```

Инициализируем базы данных Цербера.

```
#kdb5_util create -r MYHOME.DOMAIN -s
Loading random data
Initializing database '/usr/local/var/krb5kdc/principal' for realm 'MYHOME.DOMAIN',
master key name 'K/M@MYHOME.DOMAIN'
You will be prompted for the database Master Password.
It is important that you NOT FORGET this password.
Enter KDC database master key: <-masterpass
Re-enter KDC database master key to verify: <-masterpass
#
```

Еще вам нужен файл контроля доступа пользователей к базам данных Цербера, который вам придется написать в любимом текстовом редакторе и поместить в папку /usr/local/var/krb5kdc. В принципе для начала вот такого файла будет достаточно:

```
cat /usr/local/var/krb5kdc/kadm5.acl
*/admin@MYHOME.DOMAIN *
*@MYHOME.DOMIAN cil *1/admin@MYHOME.DOMAIN
*/*@MYHOME.DOMIAN i
```

Смысл такого файла состоит в том, что принципалам с instance admin даны все права по модификации базы данных Цербера, любому принципалу дается возможность менять пароль и получать данные о принципалах с тем же именем и instance admin (с помощью команд `kadmin`, `kadmin.local`), и любой принципал может просматривать данные о полученных им билетиках (команда `klist`).

После чего добавим в базу Цербера его администратора:

```
#kadmin.local
Authenticating as principal mike/admin@MYHOME.DOMAIN with password.
kadmin.local:  addprinc admin/admin@MYHOME.DOMAIN
WARNING: no policy specified for admin/admin@MYHOME.DOMAIN; defaulting to no policy
Enter password for principal "admin/admin@MYHOME.DOMAIN": adminpass
Re-enter password for principal "admin/admin@MYHOME.DOMAIN": adminpass
Principal "admin/admin@MYHOME.DOMAIN" created.
kadmin.local:#
```

Теперь все готово для запуска демонов. Демон kadmind необязателен, если мы собираемся администрировать Цербер с локального компьютера с помощью kadmin.local.

```
#krb5kdc
#tail /var/log/krb5kdc.log
Sep 25 07:27:20 ns krb5kdc[464](info): setting up network...
Sep 25 07:27:20 ns krb5kdc[464](info): listening on fd 6: udp 192.168.192.1.750
Sep 25 07:27:20 ns krb5kdc[464](info): listening on fd 7: udp 192.168.192.1.88
Sep 25 07:27:20 ns krb5kdc[464](info): set up 2 sockets
Sep 25 07:27:20 ns krb5kdc[465](info): commencing operation
#

#kadmind
#tail /var/log/kadmin.log
Sep 25 07:28:44 ns kadmind[467](info): Seeding random number generator
Sep 25 07:28:44 ns kadmind[467](info): No dictionary file specified, continuing witho
Sep 25 07:28:44 ns kadmind[468](info): starting
#
```

Теперь нам нужно добавить двух принципалов - сам хост на котором работает Цербер со случайно выбранным ключом и нашего newhesiodusr.

```
#kadmin -r MYHOME.DOMAIN -p admin/admin
Authenticating as principal admin/admin with password.
Password for admin/admin@MYHOME.DOMAIN: adminpass
kadmin:  addprinc -randkey host/ns.myhome.domain
WARNING: no policy specified for host/ns.myhome.domain@MYHOME.DOMAIN; defaulting to n
Principal "host/ns.myhome.domain@MYHOME.DOMAIN" created.
kadmin:  addprinc newhesiodusr
WARNING: no policy specified for newhesiodusr@MYHOME.DOMAIN; defaulting to no policy
Enter password for principal "newhesiodusr@MYHOME.DOMAIN": hesiodpass
Re-enter password for principal "newhesiodusr@MYHOME.DOMAIN": hesiodpass
Principal "newhesiodusr@MYHOME.DOMAIN" created.
kadmin: #
```

Вот теперь ввод пароля работает!

```
#login.krb5
login: newhesiodusr
Password for newhesiodusr:
newhesiodusr: Kerberos password incorrect
Login incorrect
```

```
login: newhesiodusr
Password for newhesiodusr:
Last login: Sat Sep 25 07:57:30 on ttyp2
Linux 2.4.22.
$whoami
newhesiodusr
$id
uid=2000(newhesiodusr) gid=1200(hesgr) groups=1200(hesgr)
```

7 Итоги

Еще раз напоминаю, что в этом тексте рассматривается только проверка идеи построения системы с централизованной регистрацией пользователей. По крайней мере на одном отдельном компьютере она работает. Перенос же модели в реальное сетевое окружение требует решения еще нескольких задач, на которые стоит обратить внимание. Во-первых, следует иметь ввиду возможность сбоя в сети - обрыв линии связи, выход из строя компьтеров, на которых работают Цербер и/или Гесиод. В таком случае ни на один из компьютеров локальной сети залогиниться не удасться. Тут можно предложить, во-первых, установить back-up сервера (slave серверы DNS для Гесиода и дополнительные Kerberos Domain Controllers), и, во-вторых, иметь альтернативный вход в компьютер под локальным пользователем с локальным паролем (по крайней мере для root). Ya мой взгляд второй вариант можно осуществить переключеним runlevels при старте системы. Скажем, для стандартного входа в систему (runlevel 3) мы имеем запись в /etc/inittab вида: `c1:3:respawn:/sbin/agetty -l login.krb5 38400 tty1 linux` с регистрацией в Цербере, а для экстренных случаев сохраням обычный логин `c1:125:respawn:/sbin/agetty 38400 tty1 linux`, причем в /etc/passwd - /etc/shadow для таких случаев держим минимальный набор системных пользователей и групп (достаточно ли будет одного root'a?).

Теперь о недостатках. Главный недостаток - сложная интеграция Kerberos с Linux. Как видно из примера с мэйл-клиентом и POP-сервером, чтобы приложение, требующее аутентификации пользователя, могло воспользоваться услугами Цербера, в исходном коде программы автором должна быть заложены вызовы соответствующих функций Цербера, причем это справедливо как для серверной, так и клиентской частей приложения. На практике (т.е. для системного администратора) это означает необходимость патчить, искать kerberized аналоги или (в лучшем случае) перекомпилировать программы с определенными опциями (конечно же администратор может отказаться от полной керберизации ситемы, и продолжать использовать программы имеющие свои собственные базы паролей/пользователей, например samba с ее smbpasswd или mysql с его таблицей mysql.user). Керберизованные стандартные сетевые утилиты и сервисы (r*-службы, telnet, ftp и соотвтествующие демоны) присутствуют в пакете Керберос от MIT. Для NFS специальной авторизации не требуется, т.к. она контролирует доступ к файлам по системным uid/gid, полученных пользователем при входе в систему. Использование Гесиода позволяет сделать uid/gid одинаковыми для всей сети, поэтому в случае с NFS у вас автоматически решаются проблемы с доступом к разделяемым ресурсам с разных компьютеров (в классическом варианте требуется ручная синхронизация /etc/passwd на всех компьютерах в сети). Что касается других сетевых приложений (почта, сетевая файловая система, сервер

баз данных...), то следует внимательно читать Release notes интересующей вас программы, чтобы понять возможна ли ее интеграция с Цербером. Samba и mysql курят в сторонке, так что, возможно, вам придется искать им замену (Coda и PostgresSQL?). Другая возможность керберизации приложений - это организовать обращение к Церберу через модули PAM. Но это применимо в случае, если инфраструктура PAM уже присутствует в вашем дистрибутиве (ее нет например в Slackware), иначе pam-ификация дистрибутива окажется занятием не менее сложным, чем его керберизация. Какие именно сетевые приложения можно использовать в рамках керберизованной системы я собираюсь расказать во второй части этой статьи.