Jabberd2 – простой и нетребовательный к ресурсам XMPP-сервер

mkondrin
из
hppi.troitsk.ru

15 декабря 2008 г.

Аннотация

Рассматривается установка и администрирование защищенного сервера Jabberd2 (версии 2.1.24.1) с аутентификацией пользователей в Kerberos. Исправленная и расширенная версия цикла статей, опубликованных в 8-9 номере журнала “Системный администратор” за 2008 год. Опубликована в сети по согласованию с редакцией.

1 История и предыстория

Проект Athena, запущенный в Массачусетском технологическим институте в середине 80-х, имевший своей целью создание распределённого вычислительного окружение, оказал сильное воздействие на развитие компьютерных технологий того времени. Фактически, в эпоху мэйнфреймов и терминалов, это был первый пример “сетевого окружения” в том виде как его представляют сейчас. Среди прочего авторам проекта в процессе работы пришлось решить и такую задачу, как замены утилиты talk, которая позволяла передавать короткие сообщения между терминалов, подключённых в рабочей станции. По мере разработки этой проблемы возникла идея создания протокола Zephyr, который одновременно решал две задачи - обмена сообщениями и информировал бы о наличии нужного пользователя за компьютером. Таким образом этот протокол оказался не только одним из первых (Вместе с IRC / Internet Relay Chat) протоколов мгновенного обмена сообщениями (Instant Messaging/IM), но и во многом воспроизводил архитектуру, впоследствии использованную в протоколе Jabber (XMPP - eXtensible Message Passing and Presence Protocol).

Хотя Zephyr было далеко до популярности, сравнимой с популярностью появившегося гораздо позже протокола ICQ, но сам по себе он оказался довольно удачным, и его использование в не модифицированном виде в локальных университетских (в том же MIT, например) сетях продолжается вплоть до настоящего времени. Главным фактором, работающим в пользу Zephyr, была его тесная интеграция с другой разработкой, возникшей в рамках проекта Athena, – системой удалённой аутентификации пользователей Kerberos. Поскольку Zephyr работает в закрытом окружении (университете или предприятии), с определённым кругом пользователей (сотрудники и/или студенты), особенно в том случае, когда их учётные записи и так уже хранятся в базе данных Kerberos, то такая система предоставляет с одной стороны большее

удобство использования сервиса за счёт развёртывания single-sign-on, а с другой позволяет однозначно устанавливать идентичность пользователей.

Поскольку протокол Zephyr оставался практически замороженным довольно долгое время, то в нем по-прежнему использовался старый вариант протокола Kerberos4 [1], от которого в последнее время начали отказываться, что заставит пользователей и администраторов искать альтернативные системы обмена сообщениями. Наиболее подходящим, в смысле возможности его конфигурирования, таким образом, чтобы максимально напоминать Zephyr, является Jabber/XMPP.

Сам протокол Jabber начал разрабатываться в конце 90-х годов Джереми Миллером (Jeremie Miller), в качестве альтернативы закрытого протокола ICQ. В отличии от централизованного сервиса ICQ Jabber представляет из себя распределённую систему, чем -то напоминающую электронную почту. Т.е. на сервере предприятия можно установить свой собственный сервер JABBER, так что локальные пользователи, подключаясь к нему, могут обмениваться сообщениями как внутри предприятия, так и с внешним миром, в смысле – другими серверами JABBER. Отличительной особенностью протокола является его расширяемость и использование XML-формата для передачи данных (наверное, эти две вещи как-то связаны между собой). Поскольку протокол возник сравнительно поздно, то в нём, по крайней мере во второй редакции 2003 года, предусмотрена поддержка прослойки SASL (Simple Authentication Security Layer), средствами которой к нему можно подключать различные модули аутентификации, в том числе и Kerberos. Спецификации протокола состоят из двух RFC 3920-3921, которые описывают ядро протокола (Core), и довольно обширного числа так называемых XEP, которые определяют расширения базового протокола. В настоящее время существует несколько реализаций серверов Jabber, однако, пока что ни один из них не поддерживает весь набор XEP. Во-первых, это ejabber, написанный на языке erlang, и популярный в России из-за своей русской команды разработчиков, во-вторых, написанный на Java и рассчитанный на использование в корпоративных сетях OpenFire, а также два jabberd-а, 1.4.x и 2.x, оба написанных на C. Несмотря на сходство названий jabberd и jabberd2 это два независимых проекта, который развивались параллельно от первоначальной справочной реализации (reference implementation) сервера jabberd, выполненной самим Джереми Миллером.

Хотя каждый из этих серверов имеет свои достоинства, но в данной статье будет рассмотрен именно jabberd2, по-причине своей нетребовательности к ресурсам, относительной простоты настройки, полной поддержки SASL, а также возможности использования его без необходимости установки специализированных серверов баз данных, что также в значительной степени упрощает его администрирование. В истории самого проекта было несколько сложных моментов, когда например в начале 2005 года был полностью утрачен репозиторий исходных кодов, которые затем пришлось восстанавливать с нуля. В конце того же года он претерпел полное обновление команды разработчиков. Однако в настоящий момент проект динамично развивается и к самому серьёзному его недостатку можно отнести практически полное отсутствие документации, а отдельные руководства, встречающиеся в сети, уже устарели и иногда даже противоречат тому, что имеется в настоящее время. Частично данная статья и имеет своей целью восполнить этот пробел.

Собственно, после этого довольно долгого вступления можно сформулировать наше техническое задание к серверу Jabber, реализация которого и будет описана в данной статье. В первую очередь, мы потребуем использование учётных записей пользователей, хранящихся в базе данных Kerberos и наличие как обычного тексто-

[1] Хотя, по слухам, в Университете штата Айова используется вариант Zephyr с поддержкой Kerberos5, но исходный код этой системы в свободном доступе я найти так и не смог

вого парольного входа, так и с помощью билетиков Kerberos, а также хранение всех авторизационных и дополнительных данных. необходимых для работы сервера, в локальных дисковых базах данных (для управления которыми будет использована библиотека BerkeleyDB). Также дополнительным требованием будет поддержка сервером многопользовательских конференций, которая для Jabberd2 реализована в виде отдельного проекта.

2 Компиляция и сборка

Перед сборкой jabberd2, который можно скачать с их официальной страницы [1], следует убедиться, что у вас в системе установлены библиотеки cyrus-sasl [2] и настроен сектор Kerberos, например, с помощью Heimdal Kerberos [3]. Если нет, то вам лучше начать с их настройки, которая описана в статьях [4], поскольку дальнейшие действия будут сильно зависеть от работоспособности этих двух библиотек. Также новые версии jabberd2 требуют наличия системной библиотеки glibc версии не ниже 2.3.6, из-за неправильной реализации функции fnmatch в более ранних версиях. Также рекомендуется использовать последние версии Heimdal, например, 1.1, поскольку в более ранних версиях была допущена досадная ошибка взаимодействия с SASL, которая делала аутентификацию в jabber невозможной. В дальнейшем будет использована предпоследняя версия Jabberd2 – 2.1.24.1, но как правило (с небольшим количеством исключений) это работает и в серии 2.2.x, которая, по моему мнению, всё же ещё несколько сырovата.

После развёртывания пакета с исходными текстами jabberd2, его нужно сконфигурировать, запустив в его домашнем каталоге команду:

```
./configure --prefix=/usr --sysconfdir=/etc/jabberd \
            --enable-db --with-sasl=cyrus --enable-debug
```

В этому случае активируется использование хранилища баз данных BerkeleyDB, включается поддержка cyrus-sasl и предусматривается вывод отладочной информации. Однако, перед тем как выполнить уже привычные команды make и maker install необходимо совершить ещё одно действие,чтобы активировать поддержку cyrus-sasl, а именно, закомментировать одну строку с указаниями компилятору:

```
sed -ir 's/^#error Cyrus SASL/\/\/#error Cyrus SASL/' sx/sasl_cyrus.c
```

Небольшое отступление. По умолчанию jabberd2 использует GNU-версию библиотеки SASL — gsasl[5]. Это решение было принято основным руководителем проекта Томашем Стерной (Tomasz Sterna) примерно год назад. На мой взгляд, это было серьёзной ошибкой. Помимо того, что оно спровоцировало уход из проекта Саймона Уилкинсона (Simon Wilkinson), который достоин отдельного упоминание за включение в jabberd2 поддержки GSSAPI, а также тестирование и усовершенствование jabber-клиентов для совместимости с этим механизмом, на самом деле это решение было вызвано непониманием работы SASL с т.н. security layers, которые, грубо говоря, позволяют шифровать трафик между клиентом и сервером, если они поддерживают определённые механизмы SASL. В настоящее время gsasl, в отличии от cyrus-sasl, не имеет такой функциональности, так что в данном случае мы видим просто один из примеров лечения головной боли радикальными средствами[2].

После чего сборка и установка программного пакета должны пройти без помех.

[2]В версии 2.2.x интерфейс к cyrus-sasl поломан, из-за изменения функций отвечающих за base64-encoding. Есть патч — `http://jabberd2.xiaoka.com/ticket/246`, но в релиз он пока не попал.

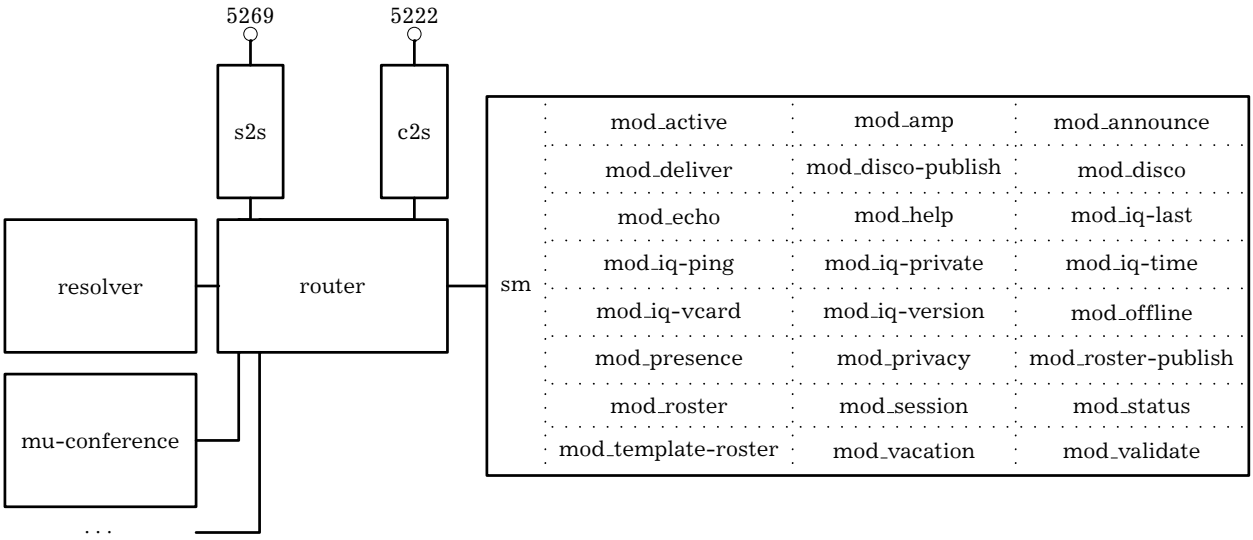

Рис. 1: Структура Jabberd2

Как уже говорилось, функциональности связанная с поддержкой в Jabberd2 многопользовательских конференций (Multi User/MU-confernce), вынесена в отдельный проект. Исходный код этого компонента можно получить со страницы [6], распаковать, скомпилировать командой make и для единообразия переместить полученный в итоге исполняемый файл в директорию, где лежат остальные компоненты jabberd2 (в нашем случае `/usr/bin`). Вопреки сведениям различных руководств, которые можно найти в интернете, специально компилировать так называемую JCR, среду запуска компонентов Jabberd2 не нужно, для последней версии mu-conference (0.7) она не требуется.

Собственно говоря, после этого можно приступать к конфигурированию Jabberd2, но вначале стоит взглянуть, на то, что же мы в итоге получили, и на внутренне устройство Jabberd2.

3 Конфигурирование и архитектура Jabberd2

При создании Jabberd2 авторы стремились сделать его максимально модульным и расширяемым, с тем, чтобы заявленная в названии протокола расширяемость (eXtensible), присутствовала и в его серверной реализации. В результате, jabberd2 состоит из нескольких процессов, взаимодействующих между собой через сетевой интерфейс. С одной стороны, такая архитектура позволяет легко добавлять новые компоненты, как, например, MU-Conference, реализующую спецификации XEP-0045. С другой, позволяет системному администратору, по разному распределяя компоненты на нескольких серверах, добиваться либо большей производительности системы, либо же поддерживать на одном компьютере несколько “виртуальных” хостов [7].

Перейдём к настройке сервера jabberd2, попутно разбираясь с функциями каждого из компонентов (см. рис. 1). Каждый компонент настраивается с помощью своего конфигурационного файла. Поскольку протокол XMPP широко использует формат XML, то нет ничего удивительного, что разработчики Jabberd2 выбрали для конфигурационных файлов тот-же самый формат. В качестве образца для создания собственных конфигурационных файлов можно использовать файлы с расширением xml.dist, входящие в состав пакета с Jabberd2, тем более что изучение комментариев

к ним даёт полезную (пусть и скудную) информацию о самой программе. Однако для простоты, можно воспользоваться приводимыми ниже “минимальными” файлами, где большинство параметров не указано, но тем не менее позволяющими получить работоспособный сервер. По умолчанию используются конфигурационные файлы из директории `/etc/jabberd` с именем совпадающим с названием компонента и расширением xml.

Начнём с центрального компонента, router, который работает в качестве диспетчера, координирующего работу всех компонент. Его конфигурационный файл router.xml выглядит следующим образом:

```
<router>
  <id>router</id>
  <pidfile>/var/jabberd/pid/router.pid</pidfile>
  <log type='syslog'>
    <ident>jabberd/router</ident>
    <facility>local3</facility>
  </log>
  <local>
    <ip>127.0.0.1</ip>
    <port>5347</port>
    <user>
      <name>jabberd</name>
      <secret>change-me</secret>
    </user>
    <secret>change-me-too</secret>
  </local>
  <aci>
    <acl type='all'>
      <user>jabberd</user>
    </acl>
  </aci>
</router>
```

Он состоит из блока с идентификатором процесса (`id`), ссылкой на файл с Process ID запущенного процесса (`pid`) и указанием использовать syslog для журналирования событий (`log`). Это практически стандартная часть, которая с очевидными изменениями будет повторяться во всех остальных файлах. Следующие две части более интересны. Теги `ip` и `port` блока `local` привязывают процесс к локальному интерфейсу (нам не нужно, чтобы router был доступен по сети) и принятому по умолчанию разработчиками Jabberd2 порту 5347, выбранному для обмена соощениями между компонентами jabberd2.

Для подключения к router всем остальным компонентам необходимо пройти авторизацию (это не имеет ничего общего с авторизацией реальных пользователей и системного пользователь под которым будет запущен сервер, о чём речь пойдёт ниже). Причём к собственным компоненты Jabberd2 применимы имя пользователя и пароль, указанные в блоке `user`, а сторонними компонентами (к числу которых относится MU-Conference) используется пароль, вынесенный в отдельный блок `secret`. При настройке своего сервера стоит по крайней мере сменить все эти пароли. В принципе, можно настроить сервер таким образом, чтобы каждый из компонент имел доступ только к определённой части функциональности, просто дав каждому из компонент своё собственное имя и отрегулировав права доступа в разделе `acl` блока `aci`. Однако, мы поступим проще, дав все права единственному общему для всех компонент пользователю jabberd.

Единственная задача компонента resolver[3] состоит в разрешении в доменной системе имён SRV записей, запрошенных компонентом s2s и соответствующих внешним jabber серверам. SRV записи вида `_xmpp-server._tcp` или их более старый вариант `_jabber._tcp` играют примерно ту же роль, что и MX записи для электронной почты, т.е. позволяют по доменной части имени пользователя (JID) получать адрес jabber-сервера (или серверов), отвечающих за обслуживание этого домена. Позже мы ещё вернёмся к настройке DNS, пока же приведём только конфигурационный файл resolver.xml:

```
<resolver>
  <id>resolver</id>
  <pidfile>/var/jabberd/pid/resolver.pid</pidfile>
  <log type='syslog'>
    <ident>jabberd/resolver</ident>
    <facility>local3</facility>
  </log>
  <router>
    <ip>127.0.0.1</ip>
    <port>5347</port>
     <user>jabberd</user>
    <pass>change-me</pass>
  </router>
  <lookup>
    <srv>_xmpp-server._tcp</srv>
    <srv>_jabber._tcp</srv>
  </lookup>
</resolver>
```

Блок `router` указывает адрес и порт, который прослушивает router, а также имя пользователя и пароль необходимые для подключения к нему (те же самые, что и в секции `local` файла router.xml).

Вообще-то говоря, протокол XMPP состоит из двух потоков данных привязанных к разным сетевым портам: один между клиентом и сервером, а второй между двумя серверами. Как раз за обработку второго потока данных отвечает компонент s2s. Настройка его достаточно проста:

```
<s2s>
  <id>s2s</id>
  <pidfile>/var/jabberd/pid/s2s.pid</pidfile>
  <log type='syslog'>
    <ident>jabberd/s2s</ident>
    <facility>local3</facility>
  </log>
  <router>
    <ip>127.0.0.1</ip>
    <port>5347</port>
    <user>jabberd</user>
    <pass>change-me</pass>
  </router>
  <local>
    <ip>0.0.0.0</ip>
    <port>5269</port>
```

[3] В серии 2.2.x этот компонент отсутствует, его функциональность перенесена в компонент s2s, а блок `<lookup>' перемещен в файл s2s.xml`

```
    <resolver>resolver</resolver>
  </local>
</s2s>
```

Блок `local`, также как и в случае router, определяет привязку к сетевому интерфейсу и порту. В данном случае от компонента требуется, чтобы доступ к нему был открыт извне, поэтому `ip` (0.0.0.0) указывает, что компонент использует все доступные интерфейсы и привязан к стандартному порту 5269 (jabber-server). Также в этой же секции указан идентификатор компонента resolver.

Пока что вся настройка не требует существенной правки конфигурационных файлов, практически в каждом случае можно было бы обойтись минимальной переделкой файлов xml.dist из состава Jabberd2. Более серьёзная правка требуется для двух других файлов, отвечающих стандартным компонентам Jabberd2.

За второй поток данных (между клиентом и сервером), а также аутентификацию реальных пользователей отвечает компонент c2s. Также как и для компонента s2s доступ к нему должен быть открыт по сети через порт 5222 (jabber-client):

```
<c2s>
  <id>c2s</id>
  <pidfile>/var/jabberd/pid/c2s.pid</pidfile>
  <log type='syslog'>
    <ident>jabberd/c2s</ident>
    <facility>local3</facility>
  </log>
  <router>
    <ip>127.0.0.1</ip>
    <port>5347</port>
    <user>jabberd</user>
    <pass>change-me</pass>
  </router>
  <local>
    <id realm='MYREALM.RU'>myrealm.ru</id>
    <ip>0.0.0.0</ip>
    <port>5222</port>
  </local>
  <authreg>
    <path>/usr/lib/jabberd</path>
    <module>db</module>
    <db>
      <path>/var/jabberd/db</path>
    </db>
    <mechanisms>
      <traditional/>
      <sasl>
        <plain/>
        <gssapi/>
      </sasl>
    </mechanisms>
  </authreg>
</c2s>
```

Обратите внимание на тег `id` в блоке `local`. В качестве его значения необходимо указывать не имя вашего jabber-сервера, а доменное имя, иначе у вас гарантированно возникнут проблемы с аутентификацией пользователей. Фактически то же самое

имя, набранное большими буквами, можно указать в атрибуте realm, оставленного для тех редких случаев, когда имя домена и сектора (SASL или Kerberos) не совпадают. Заметьте также, что атрибут register-enable, обычно рекомендуемый в руководствах, отсутствует. Собственно говоря, это и понятно, мы не собираемся делать общедоступный jabber сервер[4], а напротив, собираемся разрешить к нему доступ только тем, кто уже зарегистрирован в базе данных Kerberos. Однако это условие будет необходимым, но ещё не достаточным для подключения к серверу Jabberd2. Дополнительным условием будет присутствие соответствующей записи для пользователя в собственной базе данных сервера Jabberd2 — authreg. Собственно говоря, в данном случае происходит просто дублирование имён пользователей из Kerberos, но внутренне устройство сервера Jabberd2 не позволяет избежать этого. Как уже говорилось, чтобы не поднимать дополнительный сервер баз данных, хранение всех данных jabberd2 будет производиться в файловых базах BerkeleyDB(тэг module). В блоке db указан путь, где будут располагаться эти базы данных. Список поддерживаемые механизмов аутентификации собран в блоке mechanisms: там указаны как простые (traditional, осуществляемые согласно старыми спецификациями протокола jabber 1998 года, в нашем случае список пуст), так и более новые механизмы SASL (sasl). В последнем случае, в соответствии с техническим заданием, разрешено использовать как открытые текстовые пароли (plain), так и аутентификацию средствами инфраструктуры Kerberos (gssapi).

Данные, которые получены компонентом c2s, поступает на обработку модулю sm (Session Manager). Он имеет наиболее объёмный конфигурационный файл, поскольку для каждого из событий необходимо указать последовательность обработки полученных пакетов, в виде цепочки (chain) подгружаемых модулей.

```
<sm>
  <id>myrealm.ru</id>
  <pidfile>/var/jabberd/pid/sm.pid</pidfile>
  <router>
    <ip>127.0.0.1</ip>
    <port>5347</port>
    <user>jabberd</user>
    <pass>change-me</pass>
  </router>
  <log type='syslog'>
    <ident>jabberd/sm</ident>
    <facility>local3</facility>
  </log>
  <storage>
    <path>/usr/lib/jabberd</path>
    <driver>db</driver>
    <db>
      <path>/var/jabberd/db</path>
    </db>
  </storage>
  <aci>
    <acl type='all'>
      <jid>root@myrealm.ru</jid>
    </acl>
  </aci>
```

[4]Вернее, его можно сделать и общедоступным, если только настроить регистрацию пользователей в секторе Kerberos какими-нибудь дополнительными средствами, в виде web-интерфейса, например.

```
<modules>
  <chain id='sess-start'>
    <module>status</module>           <!-- record status information
       -->
  </chain>
  <chain id='sess-end'>
    <module>status</module>           <!-- update status information
       -->
    <module>iq-last</module>          <!-- update logout time -->
  </chain>
  <chain id='in-sess'>
    <module>validate</module>         <!-- validate packet type -->
    <module>status</module>           <!-- update status information
       -->
    <module>privacy</module>          <!-- manage privacy lists -->
    <module>roster</module>           <!-- handle roster get/sets and
       s10ns -->
    <module>vacation</module>         <!-- manage vacation settings
       -->
    <module>iq-vcard</module>         <!-- store and retrieve the user
       's vcard -->
    <module>iq-private</module>       <!-- manage the user's private
       data store -->
    <module>disco</module>            <!-- respond to agents requests
       from sessions -->
    <module>amp</module>              <!-- advanced message processing
        -->
    <module>offline</module>          <!-- if we're coming online for
       the first time, deliver queued messages -->
    <module>announce</module>         <!-- deliver motd -->
    <module>presence</module>         <!-- process and distribute
       presence updates -->
    <module>deliver</module>          <!-- deliver packets with full
       jids directly -->
  </chain>
  <chain id='out-sess'/>
  <chain id='in-router'>
    <module>session</module>          <!-- perform session actions as
       required by c2s -->
    <module>validate</module>         <!-- validate packet type -->
    <module>presence</module>         <!-- drop incoming presence if
       user not online -->
    <module>privacy</module>          <!-- filter incoming packets
       based on privacy rules -->
  </chain>
  <chain id='out-router'>
    <module>privacy</module>          <!-- filter outgoing packets
       based on privacy rules -->
  </chain>
  <chain id='pkt-sm'>
    <module>iq-last</module>          <!-- return the server uptime
       -->
    <module>iq-time</module>          <!-- return the current server
```

```
    time -->
  <module>iq-version</module>       <!-- return the server name and
    version -->
  <module>amp</module>              <!-- advanced message processing
     -->
  <module>disco</module>            <!-- build the disco list;
    respond to disco queries -->
  <module>announce</module>         <!-- send broadcast messages (
    announce, motd, etc) -->
  <module>help</module>             <!-- resend sm messages to
    administrators -->
  <module>echo</module>             <!-- echo messages sent to /echo
     -->
</chain>
<chain id='pkt-user'>
  <module>roster</module>           <!-- handle s10n responses -->
  <module>presence</module>         <!-- process and distribute
    incoming presence from external entities -->
  <module>iq-vcard</module>         <!-- grab user vcards -->
  <module>amp</module>              <!-- advanced message processing
     -->
  <module>deliver</module>          <!-- deliver the packet to an
    active session if we can -->
  <module>vacation</module>         <!-- send vacation messages -->
  <module>offline</module>          <!-- save messages and s10ns for
     later -->
  <module>iq-last</module>          <!-- return time since last
    logout -->
</chain>
<chain id='pkt-router'>
  <module>session</module>          <!-- take sessions offline their
     c2s disappears -->
  <module>disco</module>            <!-- query new components for
    service information -->
</chain>
<chain id='user-load'>
  <module>active</module>           <!-- get active status -->
  <module>roster</module>           <!-- load the roster and trust
    list -->
  <module>roster-publish</module>   <!-- load the published roster
    -->
  <module>privacy</module>          <!-- load privacy lists -->
  <module>vacation</module>         <!-- load vacation settings -->
</chain>
<chain id='user-create'>
  <module>active</module>           <!-- activate new users -->
</chain>
<chain id='user-delete'>
  <module>active</module>           <!-- deactivate users -->
  <module>announce</module>         <!-- delete motd data -->
  <module>offline</module>          <!-- bounce queued messages -->
  <module>privacy</module>          <!-- delete privacy lists -->
  <module>roster</module>           <!-- delete roster -->
```

```
      <module>vacation</module>          <!-- delete vacation settings
         -->
      <module>status</module>            <!-- delete status information
         -->
      <module>iq-last</module>           <!-- delete last logout time -->
      <module>iq-private</module>        <!-- delete private data -->
      <module>iq-vcard</module>          <!-- delete vcard -->
    </chain>
  <user>
    <auto-create/>
    <template>
      <publish/>
    </template>
  </user>
</sm>
```

Использование подгружаемых модулей является ещё одним дополнительным источником расширения функциональности сервера Jabber2. Например, показанный на диаграмме модуль `mod_iq-vcard` позволяет пользователям публиковать свои личные данные в виде электронных визитных карточек, а модуль `mod_published-roster`[5] — даёт возможность администратору создавать на сервере динамический список контактов, доступный всем пользователям этого сервиса (внутри предприятия имеет смысл просто занести в этот список всех сотрудников). Показанная на листинге цепочка `user-create` просто добавляет в базу данных необходимые записи (при помощи модуля `mod_active`) при первом подключении пользователя. Администратору ничего не мешает оптимизировать работу сервера удаляя из цепочек «лишние» по его мнению модули, как например в данном случае убран из цепочки `user-create` модуль `mod_roster-template` (во многом дублирующий функциональность `mod_published-roster`), а из цепочек `user-load`,`user-delete` и `pkt-user` удалён модуль `mod_disco-publish`, аналогичный по функциональности `mod_iq-vcard`.

При редактировании этого файла нужно обратить внимание на тэг `id` — там должен быть указан тот же домен, что и в настройках c2s. В качестве хранилища данных выбран движок BerkeleyDB(блок `storage`), в качестве администратора сервиса указан пользователь root с передачей ему всех прав по управлению сервисом (блок `aci`). Кроме того (блок `user`) разрешена активация учётных записей пользователей, которые уже содержатся в базе данных authreg, а в качестве шаблона списка контактов для вновь созданных пользователей будет использован `published-roster`.

Последний конфигурационный файл используется для настройки компонента mu-conference. В качестве примера лучше всего использовать входящий в состав пакета файл muc-example.xml. Например, возможна такая редакция файла mu-conference.xml:

```
<jcr>
<name>conference.myrealm.ru</name>
<host>conference.myrealm.ru</host>
<ip>127.0.0.1</ip>
```

[5]Который, кстати, появился в Jabberd2 сравнительно недавно, благодаря усилиям русского разработчика Никиты Смирнова. Кстати, при работе с BerkeleyDB у этого модуля есть проблемы (драйвера ldapvcard, о котором речь пойдёт в конце статьи, они не касаются), обсуждаемые здесь |`http://jabberd2.xiaoka.com/ticket/212`. Упомянутый там патч не вошёл в 2.1.24.1 релиз (он появился только в версии 2.2.0), так что если вам эта функциональность важна, то патчить версию 2.1.24 придётся вручную.

```
<port>5347</port>
<secret>change-me-too</secret>
<spool>/var/jabberd/spool/rooms</spool>
<logdir>/var/jabberd/log</logdir>
<pidfile>/var/jabberd/pid/mu-conference.pid</pidfile>
<loglevel>255</loglevel>
<conference xmlns="jabber:config:conference">
      <public/>
      <vCard>
        <FN>Public Chatrooms</FN>
        <DESC>This service is for public chatrooms.</DESC>
        <URL>http://www.myrealm.ru/</URL>
      </vCard>
      <history>40</history>
      <logdir>/var/jabberd/logs/</logdir>
      <stylesheet>/var/jabberd/logs/style.css</stylesheet>
      <notice>
        <join>has become available</join>
        <leave>has left</leave>
        <rename>is now known as</rename>
      </notice>
      <sadmin>
        <user>root@myrealm.ru</user>
      </sadmin>
      <persistent/>
      <roomlock/>
</conference>
</jcr>
```

Атрибуты name и host лучше выбрать по принципу, использованном в данном примере, “conference”+<имя домена> (с помощью настройки DNS это имя должно разрешаться в адрес jabber-сервера). Тэги `ip`, `port` и `secret` относятся к компоненту router, причём в качестве пароля должен быть указан дополнительный пароль, применимый для старых компонент. Все дискуссионные комнаты сделаны постоянными (`persistent`), но создавать их может только администратор (`roomlock`). В качестве владельца/модератора конференций указан всё тот же root. Конференции журналируются, причём данные хранятся в виде xml-файлов, которые потом могут быть опубликованы в сети. Для их представления используется стилевой файл style.css, который тоже можно найти в пакете mu-conference.

4 Создание списка пользователей и запуск

Собственно, после создания конфигурационных файлов требуется решить ещё несколько административных задач — создать системного пользователя, под которым будет работать сервер jabberd2, создать каталоги для хранения баз данных, настроить DNS и систему аутентификации. А также написать скрипт, который будет запускать сервер jabberd2.

Первые две задачи решаются простым скриптом:

```
useradd -d /var/jabber -s null  jabber
mkdir /var/jabber/db
chown -R jabber /var/jabber/db
```

Настройка DNS состоит в добавлении SRV записей и создании нескольких псевдонимов для jabber-сервера.

```
jabber          A       192.168.1.252
myrealm.ru.     A       192.168.1.252
conference        CNAME   jabber

_jabber._tcp              SRV   5 0 5269 jabber
_xmpp-server._tcp         SRV   5 0 5269  jabber
_xmpp-client._tcp         SRV   5 0 5222 jabber
```

SRV записи играют примерно туже роль, что и MX записи для электронной почты. В данном примере для сервера с адресом 192.168.1.252 выбрано имя `jabber.myrealm.ru`, и все три записи указывают на него. Первые две — для порта обслуживающего подключения сервер-сервер (первая из них — это просто более старый вариант), а последняя — для порта клиент-сервер. Тут стоит подчеркнуть, что если вы хотите, чтобы ваш сервер был доступен извне, т.е. чтобы пользователи других jabber-серверов могли обмениваться сообщениями с вашими пользователями, то по крайней мере первая пара SRV записей должна быть видна снаружи, поскольку именно они позволяют удалённым jabber-серверам правильно находить сервер, отвечающий пользователю с JID `username@myrealm.ru`.

Последняя запись (`_xmpp-client._tcp`) не столь важна в этом смысле, поскольку клиенты jabber обычно позволяют напрямую указывать адрес сервера, но лучше создать и её. Кроме того некоторые из jabber клиентов игнорируют SRV записи, и если адрес сервера не указан, то просто интерпретируют доменную часть JID, как имя сервера. Для таких “поломанных” клиентов обычно приходится создавать дополнительный псевдоним для jabber сервера, состоящий только из доменной части. Также для нормальной работы компонента mu-conference необходимо создать ещё один псевдоним conference.myrealm.ru.

И последнее — это настройка аутентификации пользователей, что включает в себя настройку Kerberos, SASL и заполнение базы данных authreg. Предполагается, что пользователи уже занесены в базы данных Kerberos. В случае jabberd2 дальнейшие действия напоминают настройку электронной почты, как она описана, например, в статьях [8, 9].

Иными словами, нам нужно создать принципала для службы Jabberd (`xmpp/jabber.myrealm.ru@MYREALM.RU`) и выгрузить его ключ в файл /etc/krb5.keytab:

```
kadmin
>add xmpp/jabber.myrealm.ru
>ext xmpp/jabber.myrealm.ru
```

Следуя рекомендациям статьи [9], чтобы этот ключ был доступен для системного пользователя jabber, под которым запускается jabberd2, то нужно добавить этого пользователя в системную группу kerberos и открыть файл `/etc/krb5.keytab` на чтение этой группе.

Настройка SASL состоит в создании файла `/usr/lib/sasl2/xmpp.conf` с практически неизменным содержимым:

```
pwcheck: saslauthd
mech_list: gssapi plain
```

Чтобы jabberd2 мог аутентифицировать пользователей, входящих с простыми текстовыми паролями, то на том же хосте, где работает jabberd2, должен быть запущен saslauthd:

```
/usr/sbin/saslauthd -a kerberos5
```

причём необходимо убедиться, что именованный сокет `/var/spool/saslauthd`, через который saslauthd обменивается данными с внешними программами, доступен на чтение и запись системному пользователю jabber.

После выполнения всех этих шагов и тестирования работоспособности SASL с помощью утилиты sasltest [8], можно приступить к заполнению базы данных authreg. Для простоты мы пока занесём двух пользователей root и mike, оба из которых должны иметь своих принципалов в секторе Kerberos. В реальных условиях вам, скорее всего, понадобится скрипт, который считывает данные из базы данных Kerberos (в этом вам поможет команда `kadmin -l dump`, например), фильтрует вывод и переносит имена пользователей в базу данных authreg.

Для заполнения базы данных authreg нужно будет написать текстовый файл passwd.txt такого вида:

```
root
root\00MYREALM.RU\00<random passwd>
mike
mike\00MYREALM.RU\00<random passwd>
```

Т.е. он состоит из парных записей: нечётные строки — это ключи, чётные — значения. В качестве ключа берётся имя пользователя, а значение представляет из себя список с разделителем `\00`, состоящим из трёх полей — имени пользователя, сектора SASL и случайной комбинацией символов, которая служит в качестве простого текстового пароля для этого пользователя. Поскольку при настройке сервера jabberd2 простые (`traditional` из файла c2s.xml) механизмы аутентификации отключены, то последнее поле можно оставить пустым, оно в такой конфигурации не будет использоваться. Также следует помнить, что ни одно из полей не должно превышать по длине 255 символов.

Теперь остаётся зайти в директорию `/var/jabberd/db`, специально созданную для хранения баз данных jabberd2, и с помощью утилиты `db_load`, входящей в состав пакета BerkeleyDB, создать эту базу данных:

```
cat passwd.txt | db_load -n -t hash -c database=myrealm.ru -h ./ -T
   authreg.db
```

Справку по команде `db_load` в html формате можно найти в пакете BerkeleyDB. Подробное объяснение принципов работы с базами данных вида BerkeleyDBтребует отдельного обсуждения, но в данном случае, создаётся база данных в режиме обновления (ключ `-n`) типа hash (`-t`) и именем myrealm.ru (которое вам нужно поменять на имя вашего домена) в файле `authreg.db` с окружением/environment (`-h`), в качестве которого используется текущая директория (т.е. `/var/jabberd/db`). Ключ `-T` указывает, что в качестве входных данных используется текстовый файл. Ту же самую команду нужно использовать для пополнения или редактирования уже существующей базы данных. При этом в файл passwd.txt нужно вносить только те записи, которые вы хотите добавить и/или изменить. При необходимости удаления пользователей, проще всего блокировать соответсвующих принципалов в Kerberos, любо полностью переписать всю базу данных authreg.db, запустив команду `db_load` без ключа `-n`. В этом случае, разумеется, необходимо иметь под рукой полный список пользователей Jabber в файле passwd.txt.

Теперь всё готово к запуску сервера. Разработчики jabberd2 и этот этап переложили на плечи администраторов, так что вам придётся создавать собственный скрипт запуска. Можно воспользоваться моей заготовкой (rc.jabberd), которая, будучи запущенной с аргументами start/stop, запускает или останавливает все шесть компонент.

```
#!/bin/bash
progs="router resolver sm c2s s2s"
user=jabber.jabber
progsPath="/usr/bin"
confPath="/etc/jabberd"
pidPath="/var/jabberd/pid"
Start ( ) {
        echo "Initializing jabberd2 processes ..."
        for prog in ${progs}; do
                if [ $( pidof -s ${prog} ) ]; then
                        echo -ne "\tprocess [${prog}] already running"
                        sleep 1
                        continue
                fi
                echo -ne "\tStarting ${prog}... "
                rm -f ${pidPath}/${prog}.pid
                args="-c ${confPath}/${prog}.xml"
                sudo -u jabber ${progsPath}/${prog} ${args} -D 2 &>
                   debug${prog}.log &
                echo
                sleep 1
        done
        sudo -u jabber ${progsPath}/mu-conference
        return ${retval}
}
#
Stop ( ) {
        echo "Terminating jabberd2 processes ..."
        for prog in ${progs} ; do
                echo -ne "\tStopping ${prog}... "
                kill $(cat ${pidPath}/${prog}.pid) && rm -f ${pidPath}/$
                   {prog}.pid
                echo
                sleep 1
        done
        killall mu-conference
        return ${retval}
}
#
case "$1" in
        start)
                Start
                ;;
        stop)
                Stop
                ;;
        restart)
                Stop
```

```
                Start
                ;;
        condrestart)
                if [ -f /var/lock/subsys/${prog} ]; then
                        Stop
                        sleep 3
                        Start
                fi
                ;;
        *)
                echo "Usage: $0 {start|stop|restart|condrestart}"
                let retval=-1
esac
```

Данный скрипт выдаёт большое количество отладочной информации в файлы вида debug“имя компонента”.log в текущей директории, что бывает полезно при настройке сервиса, но сильно сказывается на быстродействии в рабочем режиме. Исправить ситуацию можно редактированием скрипта запуска[6].

Сложно рекомендовать какой-то определённый jabber-клиент, поскольку этот вопрос уже приобрёл черты holy-war cause. Однако, я всё же предлагаю использовать много протокольный клиент Pidgin [10] (который, кстати, поддерживает и упомянутый в начале статьи протокол Zephyr). Несмотря на грандиозный скандал, устроенный его пользователями, которым не понравилась введение разработчиками автоматически масштабируемого поля ввода, что даже привело к расколу среди разработчиков и появления форка — Funpidgin, мне лично кажется, что клиент довольно удобен, и, по крайней мере в Linux-е, позволяет легко настроить GSSAPI аутентификацию. Для этого нужно только поставить галочку напротив поля “Use GSSAPI...” на вкладке Advanced диалога управления учётными записями[7], который показан на рис. 2. В данном случае сервер можно не указывать, поскольку при правильно настроенном DNS он вычисляется из доменного имени, которое вводится в том же окне, что и показанное на рис. 2, но на вкладке Basic.

Многие пользователи считают функциональность Pidgin недостаточной для продвинутой работы с протоколом XMPP. Однако создание конференции в нём делается просто — запуском мастера из меню Buddies→Add Chat в основном окне, разумеется, если вы зашли под административным логином, которому разрешено создание конференций в конфигурационном файле `mu-conference.xml`.

Если Pidgin присутствующей в вашем дистрибутиве не поддерживает GSSAPI-аутентификацию, то вам придётся собирать его (Pidgin) вручную. Это не слишком сложно, нужно только при запуске `configure` обратить внимание на ключи:

```
./configure
  --enable-cyrus-sasl \
  --with-dynamic-prpls=jabber,simple \
  --with-static-prpls= \
```

которые включают поддержку Cyrus-SASL и инициируют сборку двух динамических библиотек обеспечивающих поддержку протокола XMPP[8].

[6]Заодно не забываем убрать из списка компонент resolver, если используется jabberd2 версии выше, чем 2.2.0.

[7]В последних версиях разработчики Pidgin, в полном соответствии с идеями HIG-а, эту галочку убрали. Теперь клиент молча подхватывает GSSAPI, если обнаруживает подходящий сертификат в кэше.

[8]У Pidgin-а в режиме GSSAPI есть одна неприятная проблема, что соединение обрывается, если

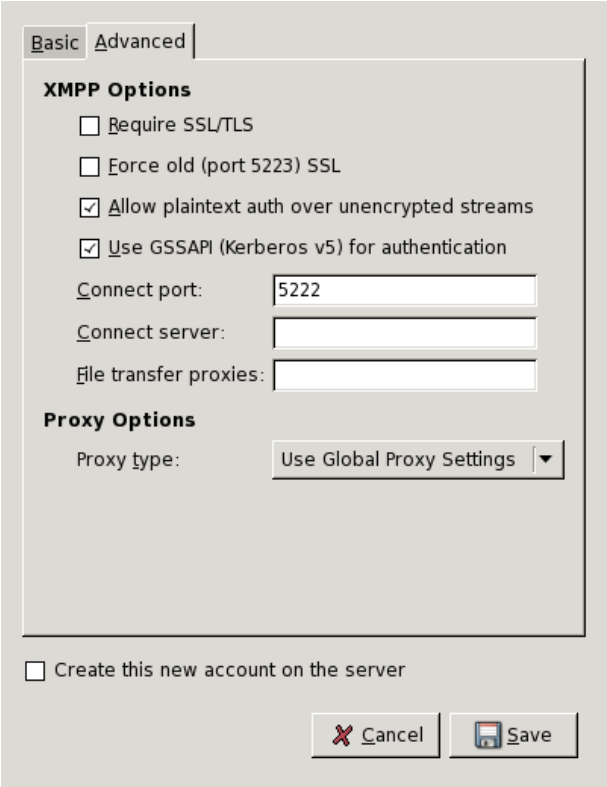

Рис. 2: Настройка GSSAPI в Pidgin 2.4.0

В принципе, уже на этом этапе сервер вполне работоспособен, но для более тонкой настройки необходимо будет разобраться с содержимым баз данных sm, которые не всегда удаётся модифицировать лишь средствами протокола XMPP. Этому вопросу и будет посвящена следующая часть статьи.

5 Настройка jabberd2 через редактирование баз данных

Итак, сервер запущен, список пользователей создан, с помощью jabber клиента Pidgin можно обмениваться сообщениями как с пользователями нашего сервера так и внешними серверами. Что дальше?

Как вы помните одним из условий техзадания для нашего jabber сервера был отказ от использования специализированных серверов баз данных для хранения учётных записей пользователей и другой необходимой информации. Поддержка сервера баз данных требует дополнительных усилий, так что гораздо проще хранить информацию в дисковых файлах, тем более что в большой степени эта информация представляет интерес только для самого сервера jabberd2 и у вас вряд ли появится необходимость делиться ею с каким нибудь другим сервисом. Также для доступа к этим дисковым файлам нами была выбрана библиотека BerkeleyDB.

Обычно базы данных BerkeleyDB характеризуют как не нуждающиеся в администрировании. Это справедливо, в том смысле, что приложения может само решать административные вопросы в более менее автоматическом режиме в процессе выпол-

послан большой объем текста. Это как раз связано с упомянутыми выше security-layers, т.е. при использовании GSSAPI клиент и сервер автоматически шифруют посылаемые данные. Если этот зашифрованный пакет по каким-то причинам оказывается оборванным, то расшифровать его не удаётся, и соединение рвётся. Эта тема обсуждается здесь - `http://developer.pidgin.im/ticket/5910`, но переубедить разработчиков так и не удалось. HIG-анутость неизлечима.

нения. В большинстве случаев это даже позволяет восстанавливать базы данных BerkeleyDB после аварийного завершения. Вопрос же создания резервных копий (вторая головная боль системных администраторов) при этом решается созданием специализированных утилит, которые решают задачу резервирования для всего приложения целиком.

Но это в том случае, если разработчики позаботились о создании таких инструментов. К сожалению, подход авторов jabberd2 в том что касается баз данных BerkeleyDB не слишком конструктивен, поскольку ничего подобного они предложить не могут. Поэтому если вы решите хранить данные jabberd2 средствами BerkeleyDB, то вам придётся волей-неволей вникать во внутренне устройство и методы работы с базами данных этого типа.

Есть, правда, одна хорошая новость — это не слишком сложно.

6 От BSD4.4 до Oracle.

История BerkeleyDB, библиотеки, которая в настоящее время насчитывает десятки миллионов инсталляций, началась в 1992 году, когда молодые сотрудники университета Беркли Марго Зельцер (Margo Seltzer) и Майк Олсон (Mike Olson) под руководством Кейта Бостича (Keith Bostic) взялись за модификацию утилиты ndbm (которая использовалась для работы с базами данных в оперативной памяти) к релизу операционной системы BSD4.4. Эта модификация оказалась удачной и уже в 1996 Кейт Бостич и Марго Зельцер основали семейный бизнес (к тому времени они поженились) - компанию Sleepycat, которая занималась продвижением BerkeleyDB. Благодаря сотрудничеству с университетом Мичигана, где был реализован один из первых серверов LDAP, и с компанией Netscape, первоначальный проект был существенно усовершенствован и приобрёл широкую популярность. В 2006 году компания SleepyCat (к которой к тому времени присоединился и Майк Олсон) была куплена лидером на рынке баз данных, компанией Oracle. До этого долгое время BerkeleyDB выходила под двойной лицензией, потому на программистских форумах слухи об этой покупке вызвали обеспокоенность, что Oracle закроет код BerkeleyDB. Однако, эти опасения не подтвердились и за последующие два года вышло несколько новых релизов этой библиотеки, причём Oracle продолжает выполнять своё обещание оставить код этой библиотеки открытым и свободным. Все трое сотрудников SleepyCat по-прежнему работают над своим продуктом, совмещая программистскую деятельность с преподавательской: Марго Зельцер одновременно является профессором Гарвардского Университета.

7 Администрирование баз данных BerkeleyDB.

Как вы уже знаете, BerkeleyDB — это библиотека, которая позволяет нескольким приложениям совместно работать как над дисковыми так и оперативными (расположенных в памяти) базами данных. Раз речь идёт о совместной работе с дисковым файлом, то возникает вопрос контроля доступа к этому файлу, чтобы информация для каждого из активных приложений имела консистентный вид, и чтобы работа нескольких приложений в параллельном режиме не приводила к порче данных. Это одна из задач, которую BerkeleyDB решает с помощью “окружений” (environment).

Если вы заглянете в директорию `/var/jabberd/db/` то обнаружите там два файла authreg.db и sm.db, один или несколько файлов типа `log.0000xx`, а также несколько файлов вида `__db.00*`. И что же с ними делать?

Собственно сама директория /var/jabberd/db/ это и есть домашняя директория environment. Файлы с расширением *.db — это собственно база данных (authreg — регистрационная информация, а sm — пользовательские данные). Файлы log.* — это, так сказать, оперативные поправки, внесённые в базу данных, и которые впоследствии переносятся в файлы *.db. А __db.* — это отображение областей памяти в дисковые файлы, которые собственно и составляют tnvironment, поскольку реализуют совместную работу нескольких приложений над базой данных. Тут следует заметить, что сам jabberd2 состоит из нескольких независимых процессов, так что для него разделение доступа к базе данных весьма актуально. Собственно информация хранится в файлах *.db и в одном из log файлов, в том что активен на текущий момент. Длинна log файла фиксирована, по умолчанию она задаётся во время компиляции и составляет примерно 10 MB, и может варьироваться с помощью файла настроек DB_CONFIG (о нем ещё пойдёт речь ниже). По мере заполнения log файла, создаётся новый, с порядковым номером на единицу больше. Поскольку информация из старых log-файлов уже сброшена в файлы *.db, то они вообще говоря не нужны, так что в этом случае необходимо будет позаботиться о ротации логов, хотя эта ротация может осуществляться и автоматически приложением, если только его автор об этом позаботился. Как вы понимаете jabberd2 — это не тот случай.

Все файлы — бинарные, так что просмотр их в текстовом редакторе или командой less вам мало что даст. Для этого случае предусмотрены специальные утилиты например db_printlog, которая позволит посмотреть последние правки внесённые в базу данных(или по крайней мере — время когда они сделаны, поскольку остальная информация не слишком вразумительна). Какие именно из log-файлов активны в данный момент можно узнать с помощью команды:

```
#db_archive -l
log.0000000001
```

и удалить лишние с помощью:

```
#db_archive -d
```

Также эта утилита позволяет выяснить, какие файлы данных содержатся в данном каталоге (если вы забыли об этом):

```
#db_archive -s
authreg.db
sm.db
```

Собственно вывод этих двух команд указывают, какие файлы нуждаются в архивировании при создании резервных копий — всего три штуки: log.0000000001, authreg.db, sm.db. Файлы вида __db.*, вообще говоря, системно зависимы и поэтому непереносимы, однако они могут быть восстановлены при наличии правильной резервной копии.

Проще всего при архивировании не копировать файлы вручную, а использовать готовую команду:

```
#db_hotbackup -v -c -h ./ -b /root/backup
db_hotbackup: hot backup started at Sun May  4 00:29:55 2008
db_hotbackup: ./: force checkpoint
db_hotbackup: ./: remove unnecessary log files
db_hotbackup: copying .//authreg.db to /root/backup/authreg.db
db_hotbackup: copying .//sm.db to /root/backup/sm.db
```

```
db_hotbackup: copying .//log.0000000001 to /root/backup/log.0000000001
db_hotbackup: lowest numbered log file copied: 1
db_hotbackup: /root/backup: run catastrophic recovery
db_hotbackup: /root/backup: remove unnecessary log files
db_hotbackup: hot backup completed at Sun May  4 00:29:57 2008
```

Ключ -h указывает домашнюю директорию environment, а -b — директорию, куда осуществляется копирование. С помощью ключа -c перед копированием происходит сбрасывание последних изменений из log файла в базы данных и удаление устаревших log-файлов. Кстати, принудительное сбрасывание данных можно осуществить с помощью команды:

```
#db_checkpoint -v -1
db_checkpoint: checkpoint begin: Sun May  4 00:32:31 2008

db_checkpoint: checkpoint complete: Sun May  4 00:32:31 2008
```

Только учтите, что на самом деле это не команда, а демон, который, будучи запущенным без ключа -1, периодически проверяет состояние log-ов в текущей директории и по-мере их заполнения или в определённые промежутки времени производит синхронизацию баз данных.

Содержимое директории с резервной копией сейчас выглядит таким образом:

```
#ls /root/backup
authreg.db  log.0000000001  sm.db
```

В таком виде база данных не работоспособна. Т.е. после копирования необходимо будет её восстановить, предварительно скопировав в ту-же директорию конфигурационный файл DB_CONFIG (если вас не устраивают параметры по умолчанию) и запустив там команду:

```
#db_recover -c -v -h ./ -e
Finding last valid log LSN: file: 1 offset 522761
Recovery starting from [1][28]
Recovery complete at Sun May  4 00:49:52 2008
Maximum transaction ID 800003eb Recovery checkpoint [1][522761]
```

Здесь ключи запуска помимо очевидного -v (verbose), означают, что команда делает восстановление после катастрофического завершения (-c) и восстанавливает окружение -e в текущей директории (-h ./). Последняя строка результата указывает, какой записи в лог-файле соответствует точка синхронизации. С помощью db_printlog можно убедиться, что это предпоследняя запись в log-файле.

Теперь резервный каталог выглядит таким образом:

```
#ls ./
DB_CONFIG  __db.002  __db.004  __db.006    log.0000000001
__db.001   __db.003  __db.005  authreg.db  sm.db
```

Проверим результат:

```
#db_verify  -h ./ authreg.db sm.db
```

Отсутствие сообщений об ошибках указывает, что копирование и восстановление базы данных прошло удачно.

По мере обновления программного обеспечения, может возникнуть ситуация, что формат базы данных не соответствует новым версиям библиотек BerkeleyDB, установленной на компьютере в процессе обновления. В этом случае нужно также обновить базы данных с помощью утилиты db_upgrade, соответствующей новой версии библиотек:

```
#db_upgrade -v -h ./ authreg.db sm.db
db_upgrade: authreg.db upgraded successfully
db_upgrade: sm.db upgraded successfully
```

Последний вопрос, который обычно возникает у администраторов баз данных — это оптимизация работы базы данных. Для BerkeleyDB решение этого вопроса сводится к написанию файла DB_CONFIG, с параметрами environment более соответствующими мощности и возможностям вашей системы.

С помощью перечисленных ниже параметров можно управлять величиной дискового кэша и кэша в памяти, метод ведения и размер логов, механизм блокировок и т.д.

```
set_cachesize
set_data_dir
set_flags DB_AUTO_COMMIT|DB_CDB_ALLDB|DB_DIRECT_DB|DB_DIRECT_LOG|
   DB_DSYNC_DB|DB_DSYNC_LOG|DB_LOG_AUTOREMOVE|DB_LOG_INMEMORY|
   DB_NOLOCKING|DB_MULTIVERSION|DB_NOMMAP|DB_NOPANIC|DB_OVERWRITE|
   DB_PANIC_ENVIRONMENT|DB_REGION_INIT|DB_TIME_NOTGRANTED|DB_TXN_NOSYNC|
   DB_TXN_SNAPSHOT|DB_TXN_WRITE_NOSYNC|DB_YIELDCPU
set_lg_bsize
set_lg_dir
set_lg_max
set_lg_mode
set_lg_regionmax
set_lk_detect DB_LOCK_DEFAULT|DB_LOCK_EXPIRE|DB_LOCK_MAXWRITE|
   DB_LOCK_MINLOCKS|DB_LOCK_MINWRITE|DB_LOCK_OLDEST|DB_LOCK_RANDOM|
   DB_LOCK_YOUNGEST
set_lk_max_lockers
set_lk_max_locks
set_lk_max_objects
set_mp_mmapsize
set_shm_key
set_thread_count
set_timeout
set_tmp_dir
set_tx_max
set_verbose
mutex_set_align
mutex_set_max
set_tas_spins
rep_set_config DB_REP_CONF_BULK|DB_REP_CONF_DELAYCLIENT|
   DB_REP_CONF_NOAUTOINIT|DB_REP_CONF_NOWAIT
```

Более подробную информацию по каждому из параметров можно получить в документации по соответствующей C функции из BerkeleyDB API в каталоге docs/api_c, где файлы руководства для этого класса функций имеет приставку env_, так что, скажем,

справка по set_cachesize находится в файле env_set_cachesize.html. Большинство параметров принимают в качестве значений одно или несколько текстовых или целочисленных значений, что можно узнать в документации. Для случаев, когда выбор ограничен набором элементов, варианты допустимых значений приведены в листинге вверху.

Например, можно использовать такой DB_CONFIG файл:

```
# one 0GB 2MB cache
set_cachesize 0 2097152 1

# Transaction Log settings
set_lg_max 2097152
set_lg_bsize 262144
set_flags DB_LOG_AUTOREMOVE
set_flags DB_DIRECT_LOG
set_flags DB_DIRECT_DB
```

который увеличивает размер кэша до 2MB (стандартных 256кБ для современных программ обычно всегда оказывается мало, что приводит к частым обращения к файлу базы данных на диске), ограничивает размер дискового log-файла 2МБ, а log-файла в оперативной памяти — 262кБ. Кроме того, с помощью флагов конфигурируется автоматическое удаление устаревших логов, а также отключается системная буферизация доступа к файлам баз данных и логов (поскольку они и так буфферизуется средствами BerkeleyDB). Проверить эффект от изменения параметров по умолчанию можно с помощью команды db_stats -e, которая позволяет получить суммарную информацию о размерах кэша, размерах лог-файлов, количества выполненных и незаконченных транзакций, установленных локах и репликациях (по отдельности эту информацию можно получить с помощью ключей -m, -l, -t, -c и -r соответственно). Также эта команда с помощью ключей -s и -d выдаёт минимальные сведения о состоянии баз данных, например, количества записей в них.

Для полноты стоит упомянуть об, уже встречавшейся в предыдущей части, команде db_loadи дополнительной к ней команде db_dump. Например с их помощью удобно производить ахивирование и восстановление баз данных сильно пострадавших в результате аварийной остановки. Для таких случаев советуют использовать команду db_dump с ключом -r, что позволяет пропускать испорченные участки в базе данных. Вывод этой команды затем можно будет подать на вход db_load, что при благоприятном стечении обстоятельств позволяет ликвидировать последствия аварии.

Хотя ранее с помощью команды db_load нам удалось создать и загрузить список пользователей authreg.db, однако для более сложных случаев создание текстового файла в формате пригодном для загрузки в BerkeleyDBоказывается ничем не легче, чем прямое редактирование баз данных с помощью программного интерфейса. В следующей главе мы как раз и перейдём к программированию BerkeleyDB.

8 В недрах sm.db

Иногда бывает нужно оперативно изменить данные в базах данных BerkeleyDB, или посмотреть уже имеющуюся там информацию. Если для решения первой части задачи можно прибегнуть к команде db_load, то со второй частью не всё так просто. К сожалению универсального инструмента для решения этой задачи нет и быть не может, что связано именно с присущими BerkeleyDB ограничениям. Но эта задача

может быть решена с помощью написания скрипта для каждого конкретного случая, в данном случае нашей целью будет создание “официально утверждённого списка контактов” (published roster), хранящегося на стороне сервера. Для достижения этой цели нам придётся отредактировать базу данных sm.db, содержащую служебную информацию, которая используется в основном компонентом sm. В отличии от authreg.db её не нужно создавать до запуска сервера, создаётся она при регистрации первого пользователя, а затем пополняется, по большей части, в автоматическом режиме.

Попутно для достижения этой цели нам придётся немного глубже разобраться с устройством BerkleyDB. В качестве рабочего инструмента будем использовать скриптовой язык TCL, который довольно распространён и отличается простым синтаксисом («все есть строка, а каждая инструкция начинается с имени процедуры, за которым следует список формальных параметров»). Более того BerkeleyDB имеет встроенный интерфейс под этот язык, наравне с C++ и Java, сборка которого активируется выбором параметра `--enable-tcl` при компиляции. Более полное представление об архитектуре и программировании BerkeleyDB можно получить из книги [11], однако там в качестве рабочего языка при разборе примеров взят C++.

В качестве напоминания в одном абзаце будут перечислены основные элементы языка Tcl. Этого хватит, чтобы разобраться с приведёнными ниже скриптами.

Tcl — это фактически shell, у переменных типов нет, всё является строкой. Каждая команда tcl представляет из себя вызов функции, за которым следует список параметров. Например, создание собственной функции:

```
proc myfunction { z } {
   puts [set z]
}
```

представляет из себя вызов функции `proc`, которой передано три параметра — имя определяемой функции `myfunction`, список формальных параметров, состоящий из одного элемента `z` и собственно тела процедуры, в котором просто выдаётся на печать значение параметра. Фигурные скобки являются группирующими элементами, квадратные - означают выполнение функции и подстановку результата. Т.е. в данном случае `[set z]` подставляет вместо имени переменной её значение. В качестве синтаксической глазури в tcl для той же самой цели используется более привычный `$`. Команда `set` используется также для присваивания `set z 1`, например.

BerkeleyDB устроена более примитивно, чем более популярные иерархические или реляционные базы данных, поскольку все данные в BerkeleyDB хранятся в виде списков пар — «ключ-значение». Если проводить аналогию с реляционными базами данных, то мы имеем как-бы таблицу, с одной колонкой — первичным индексом, и второй — значением. В то же время именно простота устройства BerkeleyDB позволяет использовать её в качестве строительного блока для создания как иерархических (OpenLDAP), так и реляционных (MySQL) баз данных.

Вторым отличительным моментом BerkeleyDB является отсутствие типизации данных, так что в качестве ключей и их значений могут быть использованы любые данных, в том числе массивы, списки, если только приложения, подключённые к базе данных, умеют их правильно интерпретировать. С одной стороны это даёт больший простор для разработчиков приложений, так как они не ограниченны в выборе структур данных. а с другой — усложняет обмен данными между приложениями, поскольку для работы с базой данных приложение должно изначально точно знать используемые типы структур. Собственно поэтому и не существует универсальных утилит для доступа к базам данных BerkeleyDB. Вообще-то, и использование `db_load` применимо только для баз данных содержащих только строковые значения,

так что нам в некотором смысле повезло, что authreg.db не содержит записей другого типа.

Тип доступа и механизм индексации в базах данных можно выбрать при создании базы данных. Всего BerkeleyDB поддерживает 4 типа — Queue, Recno, Hash, BTree. Условно говоря, эти типы означают, каким именно образом BerkeleyDB индексирует и хранит базы данных. Все базы используемые в Jabberd2 представляют из себя хэши.

Любой файл базы данных BerkeleyDB (*.db) на самом деле может содержать несколько «таблиц»/списков. Чтобы посмотреть, что представляет из себя база данных sm.db изнутри, откроем её с помощью скрипта list-sm.tcl:

```
load /usr/lib/libdb_tcl-4.4.so
set h [berkdb open -rdonly -env $e sm.db]
set c [$h cursor]
catch {
   for {set entry [$c get -first] } { $entry != {} } { set entry [$c get
       -next] } {
      eval entry $entry
      puts [lindex $entry 0]
   }
}
$c close
$h close
$e close
```

Идея проста. Вначале подгружается интерфейс доступа к библиотеке BerkeleyDB, а потом открывается environment, что (как вы помните) позволяет безопасным образом контролировать доступ к базам данных. Затем в уже созданном окружении открывается сама интересующая нас база данных, причём делается это в read-only режиме, поскольку для баз, содержащих несколько таблиц, иного выбора нет (позже мы увидим, как открыть доступ к отдельным таблицам на запись). Далее в базе данных создаётся курсор-итератор, и в цикле последовательно перебираются все ключи, которые представляют из себя имена списков. Вся логика работы с базой данных собрана в операторе перехвата ошибок `catch`, что гарантирует корректный переход к закрытию всех открытых дескрипторов точек доступа к файлам и базам данных при возникновении ошибок. Запускается скрипт в директории, где находится файл sm.db, таким образом:

```
# tclsh ../scripts/list-sm.tcl
active
disco-items
logout
motd-message
privacy-default
privacy-items
queue
roster-groups
roster-items
status
vacation-settings
vcard
```

Так выглядит список «таблиц» в только что созданной базе данных. Здесь и далее предполагается, что все скрипты собраны в отдельной директории

/var/jabberd/scripts. Заметьте также, что в скрипте нужно указывать ту версию библиотеки BerkeleyDB, с которой фактически собран Jabberd2. Как правило в системе (для совместимости) присутствуют несколько версий этой библиотеки, так что тут нужно быть внимательным.

Попробуем теперь продвинуться дальше и прочитать из таблицы vcard список виртуальных визитных карточек с информацией о пользователях, которую они сочли нужным опубликовать в Jabberd. Впоследствии, например, можно будет сделать скрипт, который переносил бы эту информацию или её часть на LDAP сервер.

Чтение таблицы осуществляется скриптом list-vcard.tcl:

```
source ../scripts/serialize.tcl
load /usr/lib/libdb_tcl-4.4.so
set e [berkdb env -home ./]
set h [berkdb open -env $e sm.db vcard]
set c [$h cursor]
catch {
   for {set entry [$c get -first] } { $entry != {} } { set entry [$c get
       -next] } {
      eval set vcards $entry
      puts "->key(jid) = [lindex $vcards 0]"
      array set card [deserialize [lindex $vcards 1]]
      foreach field [array names card] {
        puts "$field : $card($field)"
      }
   }
}
$c close
$h close
$e close
```

Как видите по сравнению с предыдущим скриптом изменений немного. Во-первых, на этот раз открывается только одна из «таблиц» (vcard) базы данных sm.db, и, во-вторых, добавлена специальная обработка значений с помощью дополнительного файла serialize.tcl, который выглядит таким образом:

```
proc deserialize { block } {
    set LL {}
    set tail $block
    while { $tail != {} } {
        set key {}
        set w " "
        while { $w != "\x0" } {
            binary scan $tail {a a*}  w tail
            set key ${key}$w
        }
        binary scan $tail {i1 a*} type tail
        switch $type {
                0 -
            1 {
                binary scan $tail {i1 a*}  val tail
            }
            2 {
                set val {}
                set w " "
                while { $w != "\x0" } {
```

```
                    binary scan $tail {a a*} w tail
                    set val ${val}$w
                }
            }
        }
        lappend LL $key $val
    }
    return $LL
}

proc pub_roster_format { line } {
    set k [split $line " "]
    set a [ binary format { a4 i1 i1 a5 i1 i1 a3 i1 i1 a5 i1 a* a a4 i1
       a* a a6 i1 a* a } ask 1 [lindex $k 1] from 0 [lindex $k 2] to 0 [
       lindex $k 3]  name 2 [lindex $k 5] "\x0" jid 2 [lindex $k 0] "\x0
       " group 2 [lindex $k 4] "\x0" ]
    return $a
}
```

Хотя как ключ так и данные в BerkeleyDB могут быть любого типа, но разработчики Jabberd2 как правило используют в качестве ключей строки, а для значений используют список-хэш. Формат списка выглядит так — вначале идёт имя элемента списка в виде строки заканчивающейся нулём, затем идёт целое число, которое обозначает тип элемента, затем собственно его значение, и так для каждого из элементов. Функция deserialize как раз и служит для разбора записей выполненных в таком формате и возвращает их в виде парного Tcl-списка, который потом группируется попарно через разделитель-двоеточие и выводится на экран. Вот, например, моя электронная визитка (та же самая, что и на рис. 3):

```
$tclsh ../scripts/list-vcard.tcl
->key(jid) = mike@hppi.troitsk.ru
nickname : mike
email : mkondrin@hppi.troitsk.ru
fn : Kondrin Mikhail
adr-region : MO
adr-country : RU
n-given : Mikhail
n-family : Kondrin
```

До настоящего ldif-формата, пригодного для загрузки на сервер LDAP пока ещё далеко, но направление движения, по крайней мере, понятно.

Не стоит думать, что BerkeleyDB поддерживает только простые таблицы. Поскольку BerkeleyDB позволяет осуществлять склейку таблиц (join) по первичному и вторичному индексу, в нем можно хранить данные достаточно сложной структуры. Также замечательной чертой BerkeleyDB является наличие транзакций, что позволяет вносить изменения в несколько таблиц атомарным способом, в стиле “всё или ничего”.

Как раз для решения поставленной в начале задачи — заполнение published-roster нам и придётся столкнуться с транзакциями. В данном случае это будет скорее вынужденная мера, поскольку в jabberd2 доступ ко всем таблицам из sm.db возможен только в режиме транзакций. Чтобы jabberd2 мог использовать созданную нами таблицу, в точно таком же режиме нам и следует открыть published-roster. Все элементы в таблице имеют ключ, состоящий из пустой строки, а в качестве значений

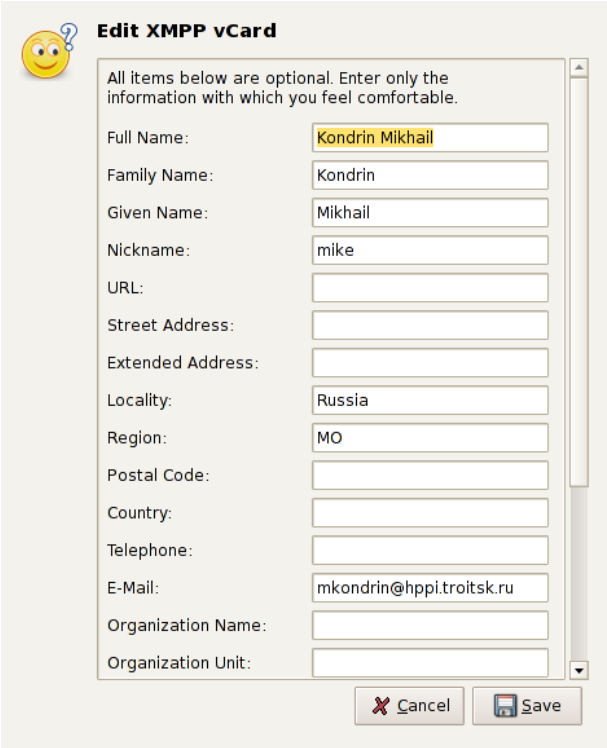

Рис. 3: Электронная визитка (vcard) в окне Pidgin

используется хэш-список с полями ask from to (целочисленными) jid name group (строкового типа). Эта информация построчно заносится в текстовый файл pub.txt откуда она и будет впоследствии считываться скриптом. Например, в нашем случае можно предложить такой вариант этого файла:

```
root@hppi.troitsk.ru 0 0 0 Admin root
postmaster@hppi.troitsk.ru 0 0 0 Admin postmaster
mike@hppi.troitsk.ru 0 0 0 Colleagues mike
```

где учётные записи разнесены по двум группам — реальных пользователей (группа Colleagues) и административных/технических учётных записей (Admin). Сам же скрипт выглядит таким образом:

```
source ../scripts/serialize.tcl
load /usr/lib/libdb_tcl-4.4.so
set e [berkdb env -create -txn -home ./ -recover]
set h [berkdb open -dup -hash -create -auto_commit -env $e sm.db
   published-roster]
set f [open published.txt]
set t [$e txn]
set err [ catch {
  foreach line [split [read $f] "\n"] {
    if { $line == {} } break
    set a [ pub_roster_format $line]
    $h put -txn $t ""  $a
  }
}]
if { $err } {
    $t abort
} else {
    $t commit
```

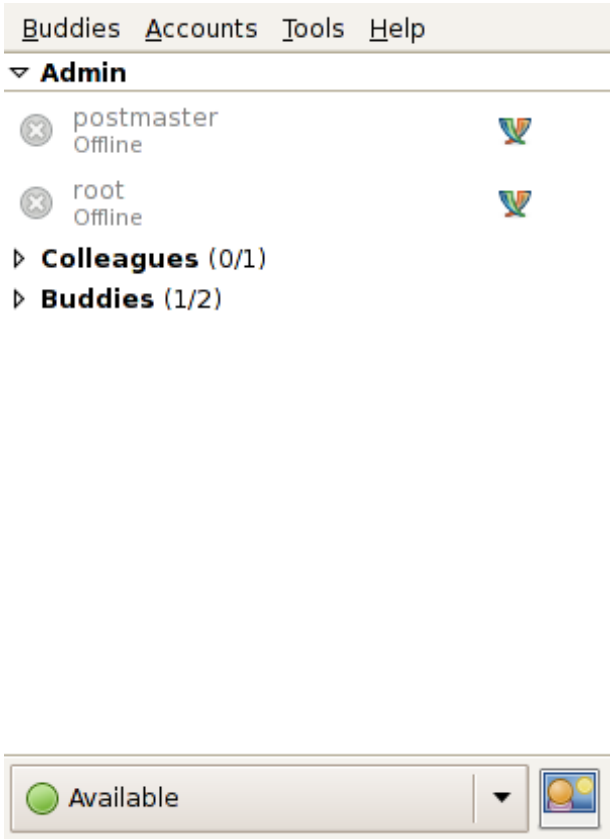

Рис. 4: Published_roster в окне Pidgin

```
}
close $f
$h sync
$h close
$e close
```

В данном случае окружение открывается несколько иным способом — в режиме восстановления (флаг `-recover`), что является рекомендуемым способом обработки ошибок, вызванных возможным предыдущим некорректным завершением работы с базой данных. Поскольку при исправлении ошибок может возникнуть необходимость вносить изменения в базу данных, которая к тому же ещё является транзакционной, то приходится добавлять ещё два флага `-create -txn`. Они как раз и означают, что база открыта для записи в транзакционном режиме.

Все базы создаваемые внутри sm.db являются транзакционными за счёт выставленного флага `auto_commit`, имеют тип Hash и не требуют уникальности индекса(флаг `-dup`). Все изменения, вносимые в файлы, обернуты в транзакцию, причём в отличии от предыдущих случаев вызов `catch` анализируется на предмет обнаружения ошибок, и (в зависимости от результата) транзакция либо откатывается, либо нормально завершается после прочтения файла и заполнения published-roster. Затем база данных синхронизируется, т.е. все изменения хранящиеся в кэше в памяти принудительно сбрасываются на диск. Сама процедура форматирования записей `pub_roster_format` для помещения в базу данных выглядит довольно уродливо, и поэтому вынесена также в подгружаемый файл serialize.tcl. В итоге результат выглядит примерно так, как показано на рис. 4.

Можно также проверить, что изменения внесены в `published_roster`, непосредственно на сервере с помощью такого скрипта:

```
source ../scripts/serialize.tcl
load /usr/lib/libdb_tcl-4.4.so
set e [berkdb env -create -txn -home ./ -recover]
set h [berkdb open -env $e sm.db published-roster]
set c [$h cursor]
for {set i [$c get -first] } { $i != {} } { set i [$c get -next] } {
    eval set i $i
    puts "Key: [lindex $i 0]"
    puts [deserialize [lindex $i 1]]
}
$c close
$h close
$e close
```

Этот скрипт я комментировать не буду, поскольку все использованные в нем трюки уже рассматривались ранее. Он демонстрирует ещё и тот факт, что базы BerkeleyDB могут не иметь ключа вообще, в данном примере он пуст.

На этом задачу можно считать решённой. При входе на сервер каждый пользователь теперь должен видеть отдельные общие группы со списком «опубликованных» пользователей, которые администратор может пополнять с помощью скрипта приведённого выше. Следует подчеркнуть, что использование именно локальной базы данных для этой цели связано именно с техническим заданием, которое мы перед собой поставили. Если же, скажем, у вас уже есть готовый готовый каталог LDAP с электронными визитными карточками сотрудников, то вам скорее имеет смысл использовать его в качестве хранилища v-cards. Jabberd2 достаточно гибок в этом отношении, и позволяет выбрать тип хранилища информации для каждого из модулей индивидуально. Например, в конфигурационном файле `sm.xml.dist` из дистрибутива Jabberd2 показано, каким образом можно настроить хранение визиток и published roster в каталоге LDAP, отдельно от остальных данных, для которых используется сервер MySQL.

9 Подключаем jabberd2 к LDAP

В качестве альтернативы этой возне с tcl и BerkeleyDB можно хранить vcard и published-roster на сервере LDAP, хотя поддержание его в рабочем состоянии тоже не слишком простая задача. Если вы не слишком вникали в устройство LDAP, то лучше начать с руководств [12, 13], где рассмотрено размещение пользовательских системных логинов в каталоге OpenLDAP, что имеет непосредственное отношение к нашей теме. Также можно посмотреть мой текст [14], где описано устройство ldap-схем и другие сложные вопросы организации справочных служб, что также бывает полезно иметь в виду. Здесь же установку и настройку каталога я описывать не буду.

Модули jabberd2, предназначенные для работы с ldap, также дело рук Никиты Смирнова. Такое впечатление, что первоначально предполагалось, что `published_roster` будет работать исключительно с ldap, но удачная архитектура системы позволила практически без лишних усилий и даже без ведома со стороны разработчиков перенести эту функциональность на любые другие движки (на тот же, рассмотренный в предыдущем параграфе, BerkeleyDB, например). Поэтому начать придётся с компиляции соответствующего storage-модуля (по умолчанию он не собирается), для чего нужно будет запустить скрипт configure с флагом `--enable-ldap`, что помимо модуля `storage-ldapvcard`, запускает ещё сборку authreg модулей

authreg_ldap и authreg_ldapfull, позволяющих пользователям авторизоваться в jabberd средствами LDAP. Последнее, впрочем, для нас интереса не представляет.

После этого можно приступать к настройке jabberd и OpenLDAP. План действий будет таков — нужно настроить jabberd2, чтобы он мог подключиться к каталогу, на котором содержатся учетные записи с контактами пользователей, по определённому правилу конвертировал полученную оттуда информацию и выдавал её пользователям jabberd уже в виде электронной визитки и/или в виде списка контактов.

Включение этой функциональности достигается следующим блоком, который нужно будет врезать в файл sm.xml в блок <storage> (не забыв поменять там значения!):

```
<driver type='vcard'>ldapvcard</driver>
<driver type='published-roster'>ldapvcard</driver>
 <ldapvcard>
   <uri>ldap://ds.myrealm.ru</uri>
   <uidattr>uid</uidattr>
   <objectclass>inetOrgPerson</objectclass>
   <basedn>ou=Persons,o=myrealm</basedn>
   <groupattr>jabberPublishedGroup</groupattr>
   <publishedattr>jabberPublishedItem</publishedattr>
   <binddn>ou=Persons,o=myrealm</binddn>
   <bindpw>ldap-simple-secret</bindpw>
   <pwattr>userPassword</pwattr>
   <publishedcachettl>60</publishedcachettl>
 </ldapvcard>
```

Директивы <driver> указывают, что "таблицы" published_roster и vcard будут храниться отдельно от остальных данных (для которых по-прежнему используется BerkeleyDB) с помощью драйвера ldapvcard. Следует иметь в виду, что этот модуль работает только в режиме чтения, поменять с помощью него информацию в каталоге нельзя, так что для модификации записей придется прибегнуть к другим средствам, отличным от клиентов Jabber. Блок ldapvcard отвечает за настройку этого хранилища. Атрибут <uri> указывает имя сервера, на котором расположен каталог, а basedn — DistinguishedName (DN, уникальное имя) ветки каталога, в которой находятся учётные данные пользователей. Предполагается, что для хранения информации о пользователях будет использован популярный структурный класс inetOrgPerson (тэг <objectclass>) . Соответствующая схема содержится в пакете OpenLDAP и ,более того, этот класс неформально стандартизован (RFC2798). Атрибуты этого класса конвертируются в поля vcard следующим образом (из storage_ldapvcard.c, слева — атрибуты LDAP, справа — поля vcard):

```
ldapvcard_entry_st ldapvcard_entry[] =
{
    {"displayName","fn",os_type_STRING},
    {"cn","nickname",os_type_STRING},
    {"labeledURI","url",os_type_STRING},
    {"telephoneNumber","tel",os_type_STRING},
    {"mail","email",os_type_STRING},
    {"title","title",os_type_STRING},
    {"description","desc",os_type_STRING},
    {"givenName","n-given",os_type_STRING},
    {"sn","n-family",os_type_STRING},
    {"st","adr-street",os_type_STRING},
    {"l","adr-locality",os_type_STRING},
```

```
    {"postalCode","adr-pcode",os_type_STRING},
    {"c","adr-country",os_type_STRING},
    {"o","org-orgname",os_type_STRING},
    {"ou","org-orgunit",os_type_STRING},
    {NULL,NULL,0}
};
```

Работа драйвера ldapvcard зависит от наличия и значения нескольких атрибутов, содержащихся в каталоге. С помощью атрибута uid класса inetOrgPerson (тэг <uidattr>) драйвер находит в каталоге нужные записи, причём в качестве ключа используется имя пользователя jabber (точнее его имя и доменная часть). Другими словами, чтобы драйвер ldapvcard заработал, поле uid учётных записей пользователей не должно быть пустым и должно совпадать с JID-ом. Если это поле занято под что-то другое (системный логин, например), то следует выбрать другой атрибут и указать его имя в файле sm.xml. В качестве варианта для этих целей можно добавить к структурному классу inetOrgPerson известный вспомогательный (auxiliary) класс mozillaAddressBookEntry [15] и использовать его поле nsAIMid. После того как нужная запись найдена в каталоге, то решение публиковать её в jabber или нет, принимается драйвером ldapvcard в зависимости от значения логического атрибута jabberPublishedItem (тэг <publishedattr>). Этот атрибут определён в классе jabberExtendedObject (OID 1.3.6.1.4.1.8381.1.6, зарегистрированный на Certainty Solutions, Inc.) из схемы jabberd2.schema, которая включена в пакет jabberd2 (искать в каталоге tools). Если значение атрибута равно 0, то соответствующая запись из LDAP каталога пользователям jabber видна не будет. Чтобы подключить схему в OpenLDAP, то следует прибегнуть к стандартной процедуре — переместить её в каталог /etc/openldap/schema, прописать её в конфигурационном файле /etc/openldap/slapd.conf и перезапустить сервер slapd.

Кроме этого класс jabberExtendedObject содержит второе полезное поле — jabberPublishedGroup (тэг <groupattr>). Это имя группы в published_roster, куда будет добавлен соответствующий пользователь. Значения полей ask,from,to жёстко вшиты в драйвер и равны 0,1,1 соответственно.

Три тэга <bindn>,<bindpw> и <pwattr> отвечают за авторизацию сервера jabberd2 в каталоге OpenLDAP. В принципе, если опустить соответствующие тэги, то сервер попытается прочесть каталог OpenLDAP анонимно. В общем-то, если вас устраивает анонимный доступ к каталогу на чтение, то можно на этом и остановиться. Если же доступ к контактной информации сотрудников как-то ограничен, то придётся немного повозиться с настройкой аутентификации.

Автор ldapvcard ограничился простой аутентификацией в каталоге LDAP (SASL-аутентификация, наподобие рассмотренной в [14], тут не работает). При этом тэг <bindn> отвечает DN объекта в каталоге, от имени которого Jabber будет регистрироваться в каталоге, <bindpw> — его паролю, а <pwattr> — имени атрибута в выбранном объекте, где этот пароль хранится. Традиционно для этой цели применяется userPassword, и нарушать традиции обойдётся себе дороже. Вместо того чтобы плодить сущности в каталоге, для авторизации будет использован корень ветки каталога, где содержатся учётные записи (т.е. binddn = basedn). Поскольку этот объект является объектом класса organizationalUnit (о чем можно догадаться по присутствию атрибута ou=... в его DN), который обычно служит для группирования записей в каталоге, то поле userPassword у него имеется, так что авторизоваться под ним нам удастся. Нужно только по возможности ограничить свободу передвижения по каталогу, а также доступ к самому файлу sm.xml, поскольку пароль от этого объекта хранится там в открытом виде.

Настройка сервера OpenLDAP осуществляется правкой конфигурационного файла /etc/openldap/slapd.conf. Ниже я собрал все изменения, которые мне потребовалось внести в этот файл:

```
...
include         /etc/openldap/schema/jabberd2.schema
...
access to attr=userPassword
       by dn="cn=admin,ou=People,o=myrealm" write
       by self write
       by * auth

access  to *
        by dn="cn=admin,ou=People,o=myrealm" write
        by dn="ou=Persons,o=myrealm" none break
        by users read

access to dn.subtree="ou=Persons,o=myrealm"
       by dn="ou=Persons,o=myrealm" read
...
index uid pres,eq
index jabberPublishedItem pres,eq
...
```

С помощью директивы include подключается файл схемы jabberd2.schema. Директивы access разрешают аутентификацию с помощью полей userPassword и ограничивают доступ к каталогу для пользователя с DN ou=Persons,o=myrealm. Т.е. каталог открыт на запись “администратору” (cn=admin,ou=People,o=myrealm) и на чтение всем зарегистрированным пользователям, кроме пользователя ou=Persons,o=myrealm, который может читать только свою (одноимённую) ветку. В конце я добавил индексы по полям uid и jabberPublishedItem. Поскольку драйвер ldapvcard ищет в каталоге, используя только соотношения типа точного равенства и наличия, то для ускорения поиска достаточно ограничиться индексами вида pres,eq.

В качестве примера можно воспользоваться файлом test-jabber.ldif такого вида:

```
dn: o=myrealm
objectClass: top
objectClass: organizationalUnit
o: myrealm

dn: ou=Persons,o=myrealm
objectClass: top
objectClass: organizationalUnit
ou: Persons
userPassword: ldap-simple-secret

dn: cn=mike,ou=Persons,o=myrealm
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: mozillaAddressBookEntry
objectClass: jabberExtendedObject
sn: Кондрин
```

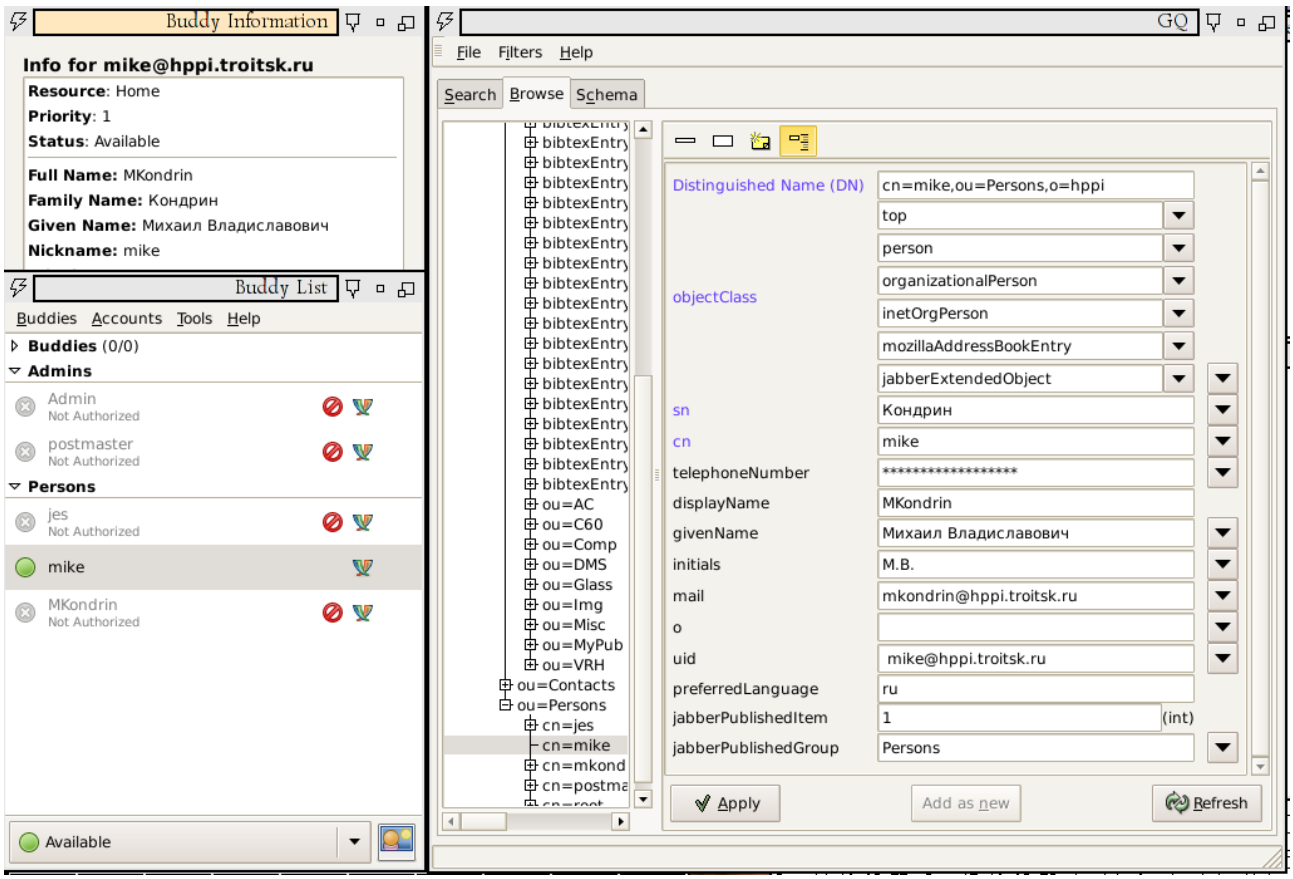

Рис. 5: GQ LDAP-клиент и окна Pidgin с изображением published_roster и vcard из каталога LDAP.

```
cn: mike
givenName: Михаил
displayName: MKondrin
mail: mike@myrealm.ru
o: Организация
preferredLanguage: ru
telephoneNumber: 1-111-111-11-11
jabberPublishedItem: 1
uid:  mike@myrealm.ru
jabberPublishedGroup: Persons
```

Загружать его на сервер следует командой:

```
cat test-jabber.ldif| iconv -t UTF8 | ldapadd -h server
```

В итоге должна получиться картина наподобие изображённой на рис. 5.

И в завершение небольшая ложка дёгтя. Как уже говорилось, основной недостаток Jabberd2 - это практически полное отсутствие документации и несовпадением уже имеющихся руководств с реальным положением дел. В эту же ловушку попал и я сам. Этот цикл статей был написан в рссчете на серию релизов Jabberd2 2.1.x, последний из которых (2.1.24.1) вышел в апреле этого года. Однако, уже в следующем релизе (2.2.0) произошли изменения в архитектуре Jabberd2 и схема приведённая в предыдущей статье уже не соответствует действительности. Компонент resolver был упразднён, а вся его функциональность перенесена в компонент s2s. При этом единственный информативный блок `<lookup>` из его конфигурационного файла также перемещён в файл `s2s.xml`. В общем-то, для динамично меняющегося проекта

такие резкие изменения не являются чем-то необычным, но хотелось бы, чтобы эти изменения хоть каким-то образом находили своё отражение в документации, хотя бы на уровне файлов README (где по-прежнему упоминается resolver как один из независимых компонентов) и ChangeLog. К сожалению, этот призыв можно адресовать не только к авторам Jabberd2, но и к другим разработчикам свободного программного обеспечения.

Список литературы

[1] Jabberd2 trac [online, cited 23 jun 2008].

[2] ftp://ftp.andrew.cmu.edu/pub/cyrus-mail/cyrus-sasl-2.1.22.tar.gz.

[3] Последняя версия heimdal-kerberos.

[4] М.Кондрин. Развертываем heimdal kerberos. *Системный Администратор*, 7, 2005.

[5] Gnu sasl homepage [online, cited 23 jun 2008].

[6] Mu-conference component for jabberd [online, cited 23 jun 2008].

[7] Installing and configuring jabberd2 [online, cited 23 jun 2008].

[8] М.Кондрин. Как настроить библиотеку SASL для совместной работы с Kerberos. *Системный Администратор*, 10, 2006.

[9] М.Кондрин. Kerberos и электронная почта. *Системный Администратор*, 11, 2006.

[10] Home | pidgin [online, cited 23 jun 2008].

[11] Himanshu Yadava. *The Berkeley DB Book*. Apress, 2007.

[12] А.Б. Барабанов. *LDAP и все-все-все. Черновик версии 2*.

[13] А.Б.Барабанов. Размещаем пользовательские бюджеты в LDAP. *Системный Администратор*, 1, 2007.

[14] mkondrin из hppi.troitsk.ru. *Проектируем справочную службу с помощью протокола LDAP*.

[15] MailNews:Mozilla LDAP Address Book Schema [online].