

Jabberd2 – простой и нетребовательный к ресурсам XMPP-сервер

Михаил Кондрин

Рассмотрим конфигурирование, архитектуру и первоначальную настройку jabberd2.

История и предыстория

Проект Athena, запущенный в Массачусетском технологическом институте в середине 80-х годов, имевший своей целью создание распределенного вычислительного окружения, оказал сильное воздействие на развитие компьютерных технологий того времени. Фактически, в эпоху мэйнфреймов и терминалов, это был первый пример «сетевых 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 терминалами, подключенными к рабочей станции. По мере разработки этой проблемы возникла идея создания протокола Zephyr, который одновременно решал бы две задачи – обмен сообщениями и информирование 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 немодифицированном виде в ло-

кальных 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 удаленной аутентификации пользователей Kerberos. Поскольку Zephyr работает в закрытом окружении (университете или предприятии), с определенным кругом пользователей (сотрудники и/или студенты), особенно в том случае, когда их учетные записи и так уже хранятся в базе данных Kerberos, то такая система предоставляет, с одной стороны, большее удобство использования сервиса, за счёт развертывания 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, от которого в последнее время начали отказываться, что заставляет пользователей и администраторов искать альтернативные системы обмена сообщениями. (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.) 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 отличие от централизованного сервиса ICQ Jabber представляет собой распределенную систему, чем-то напоминающую электронную почту. То есть 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. 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 (XMPP Extension Protocols), 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 – 1.4.x и 2.x, оба написанных на C. Несмотря на сходство названий, jabberd 1.4.x и jabberd2 – это два независимых проекта, которые развивались параллельно 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 серьезному 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 учетных записей пользователей, хранящихся в базе данных Kerberos, и наличия как обычного текстового пароля входа, так и с помощью сертификатов 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.

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 ре-

Небольшое отступление

По умолчанию jabberd2 использует GNU версию библиотеки SASL – gssapi[5]. Это решение было принято основным руководителем проекта Томашем Стерной (Tomasz Sterna) примерно год назад. На мой взгляд, это было серьезной ошибкой. Помимо того, что оно спровоцировало уход из проекта Саймона Уилкинсона (Simon Wilkinson), который достоин отдельного упоминания за включение в jabberd2 поддержки GSSAPI, а также тестирование и усовершенствование jabber-клиентов для совмес-

тимости с этим механизмом, на самом деле, это решение было вызвано непониманием работы SASL с так называемым security layers. Грубо говоря, security layer позволяют шифровать трафик между клиентом и сервером, если они поддерживают определенные механизмы SASL. В настоящее время gssapi, в отличие от cyrus-sasl, не имеет такой функциональности, так что в данном случае мы видим просто один из примеров лечения головной боли радикальными средствами.

комендуется 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.

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

```
./configure --prefix=/usr --sysconfdir=/etc/jabberd2
--enable-db --with-sasl=cyrus --enable-debug
```

В этом 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 make install, необходимо совершить еще одно действие, чтобы активировать поддержку cyrus-sasl, а именно закомментировать одну строку с указаниями компилятору:

```
sed -i -r 's/^#define fnmatch/\#define fnmatch/' config.h
sed -i -r 's/^#error Cyrus SASL/\#error Cyrus SASL/'
sx/cyrus_sasl.c
```

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

Как уже говорилось, функциональность, связанная с поддержкой в jabberd2 многопользовательских конференций (Multi User/MU-conference), 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 mu-conference 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 приступить к конфигурированию jabberd2, но вначале стоит взглянуть на то, что же мы в итоге получили, и на внутреннее устройство jabberd2.

Конфигурирование и архитектура jabberd2

При создании jabberd2 авторы стремились сделать его максимально модульным и расширяемым, с тем чтобы заяв-

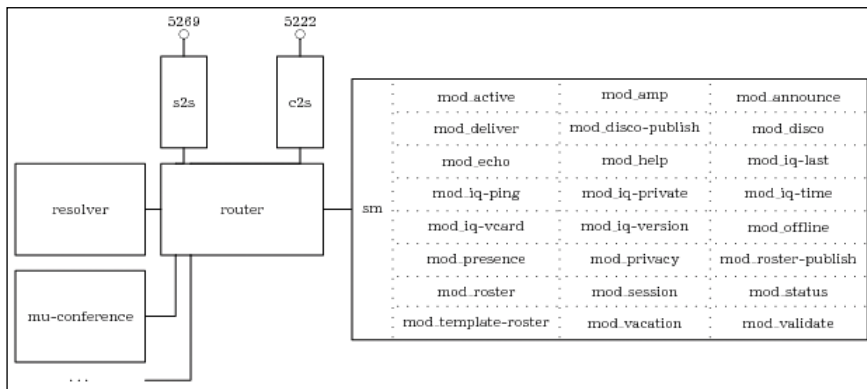


Рисунок 1. Структура jabberd2

ленная 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-0030.

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].

Перейдем к настройке сервера jabberd2, попутно разбираясь с функциями каждого из компонентов. Все компоненты настраиваются с помощью своего собственного 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 jabberd2, тем более что изучение комментариев к ним дает полезную (пусть и скудную) информацию о самой программе. Однако для простоты можно воспользоваться приводимыми ниже «минимальными» файлами, где большинство параметров не указано, но которые, тем не менее, позволяют получить вполне работоспособный сервер. По умолчанию используются конфигурационные файлы из директории /etc/jabberd с базовым именем, совпадающим с названием компонента, и расширением xml.

Начнем с центрального компонента, router, который работает в качестве диспетчера, координирующего работу всех компонентов. 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 (pidfile) 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 для обмена сообщениями между компонентами jabberd2.

Для подключения к router всем остальным компонентам необходимо пройти авторизацию (это не имеет ничего общего с авторизацией реальных пользователей или системного пользователя, под которым будет запущен сервер, о чём речь пойдет дальше). Причем для собственных компонентов 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.

Для подключения к router всем остальным компонентам необходимо пройти авторизацию (это не имеет ничего общего с авторизацией реальных пользователей или системного пользователя, под которым будет запущен сервер, о чём речь пойдет дальше). Причем для собственных компонентов 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 имеет смысл, по крайней мере, сменить все эти пароли. В принципе можно настроить сервер таким образом, чтобы каждый из компонентов имел доступ только к определенной части функциональности, просто дав каждому из компонентов свое собственное имя и отрегулировав права доступа с разделе acl блока aci. Однако мы поступим проще, дав все права единственному общему для всех компонентов пользователю jabberd.

Единственная задача компонента resolver состоит в разрешении в доменной системе имен 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 вернемся к настройке DNS, пока же приведу 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>
    <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 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 xml.dist из состава jabberd2. Более серьезная 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. Так же, 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, 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 дублирование имен пользователей из Kerberos, но внутреннее 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 (тег module). В блоке db указан путь, где будут располагаться эти базы данных. Список поддерживаемых механизмов аутентификации собран в блоке mechanisms: там указаны как простые (traditional, осуществляемые со 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, поступают на обработку модулю sm (Session Manager). Он имеет наиболее объемный конфигурационный файл, поскольку для каждого из событий необходимо указать последовательность обработки полученных пакетов, в виде цепочки (chain) подгружаемых модулей. В приведенном ниже листинге большая часть блока modules опущена, его можно будет восстановить из примеров, содержащихся в пакете jabberd2.

```
<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>
<modules>
  <path>/usr/lib/jabberd</path>
  ....
  <chain id='user-create'>
    <module>active</module>
  </chain>
  ....
</modules>
<user>
  <auto-create/>
  <template>
    <publish/>
  </template>
</user>
</sm>
```

Использование подгружаемых модулей является еще 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` (который, кстати, появился в jabberd2 сравнительно недавно, благодаря усилиям русского разработчика Никиты Смирнова) дает 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 в 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 убрал из цепочки `user-create` модуль `mod_roster-template` (во многом дублирующий функциональность `mod_published-roster`) или, например, `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 выбрана библиотека 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 учетных 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 `misc-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>
  <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, использованному 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 компонентов. 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, причем данные хранятся в виде `xml`-файлов, которые потом могут быть выложены в Интернете. Для их представления используется стилевой файл `style.css`, образец которого тоже можно найти в пакете `mu-conference`.

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.

```
$ORIGIN myrealm.ru.
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
```

Как уже упоминалось выше, смысл этих SRV-записей аналогичен 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 она позволяет удаленным 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 пользователем явно, 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-confertncse 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
```

причем 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 занесем 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>
```

то есть он состоит из парных записей: нечетные строки – это ключи, четные – значения. В качестве ключа берется имя пользователя, а значение представляет собой список с разделителем \00, состоящий из трех полей – имени пользователя, сектора SASL и случайной комбинации 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 остается зайти в директорию /var/jabberd/db, предназначенную 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 -l
-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 BerkeleyDB требует отдельного обсуждения, но в данном случае создается база данных в режиме обновления (ключ -n) типа hash (-t) и с 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

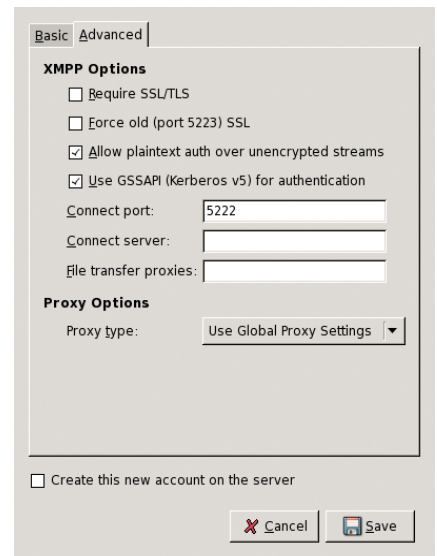


Рисунок 2. Настройка GSSAPI-аутентификации в Linux

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 или изменить. Проблемы с блокировкой пользователей проще решать удалением или блокированием соответствующих принcipалов в базе данных Kerberos либо же полностью переписать базу данных `authreg.db`, запустив команду `db_load` без ключа `-n`. Понятно, что в последнем случае вам нужно иметь полный список пользователей в текстовом файле `passwd.txt`.

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

```
#!/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}] \n"
            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} \n
            ${args} -D 2 &> debug${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) \n
            && 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: \n"
        $0 {start|stop|restart|condrestart}"
    esac
}
```

```
let retval=-1
```

```
esac
```

Данный скрипт выдает 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.

Сложно рекомендовать какой-то определенный `jabber`-клиент, поскольку этот вопрос уже для многих является достаточным поводом для начала «священной войны». Однако я всё же предлагаю использовать многопротокольный (который, кстати, поддерживает и упомянутый в начале статьи протокол `Zephyr`) клиент `Pidgin` [10]. Несмотря на грандиозный скандал, устроенный его пользователями, которым не понравилось 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, позволяет легко настроить GSSAPI-аутентификацию. Для этого нужно только поставить галочку напротив поля «Use GSSAPI...» на вкладке `Advanced` диалога управления учетными записями, 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`.

Кстати, использование GSSAPI решает ещё одну известную проблему, на которую часто жалуются пользователи `Pidgin` – хранение текстовых паролей в незашифрованном текстовом файле. В данном случае пароль «хранится» средствами библиотеки Kerberos, так что `Pidgin` не имеет к нему никакого доступа.

Многие пользователи считают функциональность `Pidgin` недостаточной для продвинутой работы с протоколом XMPP. Однако создание конференций в нем 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 вы не обнаружили кнопку «Use GSSAPI...» в сборке `Pidgin` присутствующей в вашем дистрибутиве, то, скорее всего, вам его (`Pidgin`) придется собирать 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 иницируют сборку двух динамических библиотек, обеспечивающих поддержку протокола XMPP.

Уже на этом этапе сервер вполне работоспособен, но для более тонкой настройки необходимо будет разбраться с содержимым баз данных `sm.db`, которые не всегда удается модифицировать лишь средствами протокола XMPP. Этому вопросу и будет посвящена следующая часть статьи. 