API аутентификации приложений SASL в Astra Linux

Версия: 2022-04-27

Аннотация

Настоящий документ содержит описание универсального прикладного программного интерфейса SASL (Simple Authentication Security Layer, Простой уровень аутентификации и безопасности) (далее - SASL), предоставляющего унифицированные средства для аутентификации и защиты передаваемых данных в протоколах передачи данных на основе соединений.

Документ предназначен для разработчиков программного обеспечения.

Документ основан на информационных материалах разработчиков библиотеки SASL, свободно доступных в сети Интернет.

В документе приведено краткое описание библиотеки SASL, поддерживаемых библиотекой механизмов аутентификации и протоколов передачи данных, примеры программного интерфейса клиентского и серверного приложений.

Документ применим к Astra Linux Common Edition и к Astra Linux Special Edition.

Версия: 2022-04-27

СОДЕРЖАНИЕ

Версия: 2022-04-27

Описание библиотеки SASL

Библиотека SASL создана с целью предоставить разработчикам протоколов обмена данными унифицированный доступ к механизмам аутентификации. При использовании SASL разработчики протокола обмена данными должны просто использовать интерфейсы SASL, которые, в свою очередь, обеспечивают выбор и использование доступных механизмов аутентификации. При этом появление новых механизмов аутентификации не требует внесения изменений в протоколы передачи данных.

Разработчики приложений, вместо необходимости поддерживать каждый механизм аутентификации для каждого используемого протокола, просто используют SASL для каждого протокола. При этом разработчикам нет необходимости вникать в детали процедур аутентификации, скрытые за унифицированным интерфейсом SASL, а использование в SASL технологии динамически подключаемых библиотек позволяет добавлять новые механизмы аутентификации без остановки и перезапуска приложений.

При этом в задачи библиотека SASL не входит выполнение действий по авторизации подключений.

Полное описание SASL приведено в документе RFC 4422[1].

Механизмы аутентификации SASL

Общие стандарты организации встраиваемых в SASL механизмов аутентификации определены в стандарте RFC 4422[1], и в дополнительном стандарте RFC 7677[2] (механизмы аутентификации SCRAM SASL). При этом порядок работы поддерживаемых механизмов аутентификации определяется отдельными стандартами на сами эти механизмы.

Наиболее часто используемыми механизмами аутентификации являются:

Механизм аутентификации	**Описание**
EXTERNAL	Используется, когда аутентификация отделена от передачи данных (например, когда протоколы уже используют IPsec или TLS);
ANONYMOUS	Механизм для аутентификации гостевого (анонимного) доступа (RFC 4505[3]);
PLAIN	Простой механизм передачи паролей открытым текстом;
OTP	Механизм одноразовых паролей. OTP;
CRAM-MD5	Механизм аутентификации вида запрос-ответ (англ. challenge-response authentication mechanism, CRAM), определенному в RFC

	2195[4], основанному на алгоритме HMAC-MD5 MAC.
NTLM	Протокол сетевой аутентификации, разработанный фирмой Microsoft для Windows NT (RFC 2433[5] и RFC 2759[6]).
GSSAPI	англ. Generic Security Services API (Общий Программный Интерфейс Сервисов Безопасности) — API для доступа к сервисам безопасности. Это основной механизм, применяющийся для аутентификации через Керберос. (RFC 2743[7], RFC 2744[8], RFC 1964[9], RFC 4121[10], RFC 4178[11], RFC 2025[12], RFC 2847[13], JSR-72[14])
GateKeeper (GateKeeperPassport)	Механизм аутентификации, разработанный Microsoft для MSN Chat

Полный перечень механизмов аутентификации, доступных через SASL, ведётся службой IANA (Internet Assigned Numbers Authority — «Администрация адресного пространства Интернет»). Актуальный список поддерживаемых SASL механизмов аутентификации доступен в сети Интернет на WEB-сайте IANA[15].

В Приложении 1 к настоящему документу приведён список поддерживаемых механизмов аутентификации по состоянию на время создания настоящего документа.

Протоколы, для которых используется SASL

SASL может быть использован в любых протоколах передачи данных. При этом разработчики прикладных программ могут использовать ранее разработанные стандартные протоколы, в которые интегрированы механизмы аутентификации SASL. Несколько наиболее употребительных общеизвестных протоколов:

Протокол	**Описание**
ACAP	Протокол доступа к конфигурационным данным приложений (англ. Application Configuration Access Protocol, ACAP) — сетевой протокол, позволяющий пользователю иметь доступ к конфигурационным данным приложений, поддерживающих ACAP, с любого компьютера, подключенного к сети.
AMQP	Advanced Message Queuing Protocol — открытый протокол для передачи сообщений между компонентами системы. Обеспечивает отдельным подсистемам (или независимым приложением) возможность обмениваться произвольным образом сообщениями через AMQP-брокер, осуществляющий маршрутизацию, возможно гарантирует доставку, распределение потоков данных, подписку на нужные типы сообщений.

Протокол	Описание
IMAP	Internet Message Access Protocol — протокол прикладного уровня для доступа к электронной почте.
IRC (с IRCX, TS6 или IRCv3 с SASL расширением)	Internet Relay Chat — протокол прикладного уровня для обмена сообщениями в режиме реального времени.
LDAP	Lightweight Directory Access Protocol («упрощенный протокол доступа к каталогам») — протокол прикладного уровня для доступа к службе каталогов X.500, позволяющий производить операции аутентификации (bind), поиска (search) и сравнения (compare), а также операции добавления, изменения или удаления записей.
libvirt	API, демон и набор инструментов для управления виртуализацией. Позволяет управлять гипервизорами Xen, KVM, VirtualBox, OpenVZ, LXC, VMware ESX/GSX/Workstation/Player, QEMU и другими средствами виртуализации, предоставляет возможность контролировать по сети виртуальные машины, расположенные на других компьютерах.
memcached	Сервис кеширования данных в оперативной памяти на основе хеш-таблиц.
POP	Post Office Protocol Version 3 (протокол почтового отделения, версия 3) стандартный интернет-протокол прикладного уровня, используемый клиентами электронной почты для получения почты с удалённых почтовых серверов по TCP-соединению.
RFB	(англ. remote framebuffer) — простой клиент-серверный сетевой протокол прикладного уровня для удалённого доступа к графическому рабочему столу компьютера, используемый в VNC. Работает на уровне кадрового буфера, и применим для графических оконных систем, например X Window System, Windows, Quartz Compositor.
SMTP	Simple Mail Transfer Protocol — простой протокол передачи исходящей почты— это широко используемый сетевой протокол, предназначенный для передачи электронной почты в сетях TCP/IP (RFC5321[16]);
Subversion's «svn»	Протокол системы SVN управления версиями документов

Протокол	Описание
XMPP	eXtensible Messaging and Presence Protocol «расширяемый протокол обмена сообщениями о присутствии», ранее известный как Jabber — открытый протокол для мгновенного обмена сообщениями и информацией о присутствии в режиме, близком к режиму реального времени.

Применение SASL в домене

Клиент-серверные приложения, использующие библиотеку SASL, рекомендуется использовать в доменах (в первую очередь — доменах под управлением FreeIPA), так как это позволяет воспользоваться готовыми доменными механизмами аутентификации (стандартный механизм доменной аутентификации Kerberos).

При этом библиотеки SASL клиентских приложений самостоятельно определят настройки доменной аутентификации.

Для использования в домене серверных приложений:

1. Серверные приложения (службы) должены быть зарегистрированы в домене. Примеры см. в WEB-интерфейсе FreeIPA «Идентификация» - «Службы». Там же можно добавить свой сервис (службу). При этом рекомендуется использовать стандартные для домена FreeIPA сервисы, в первую очередь сервис HTTP.
2. Для службы должна быть получена таблица ключей (см. команду ipa-getkeytab) и сохранена на сервере этой службы. Таблица ключей должна предъявляться службой для её идентификации.
3. Имя службы может быть произвольным.
4. Сервер службы должен быть введён в домен, причём имя севера должно:
    1. разрешаться через DNS;
    2. реверсивно разрешаться через DNS;
5. При использовании резервирования серверов должны использоваться записи SRV доменного DNS, при этом клиенты должны уметь их обрабатывать, т.е.
    1. Обращаться за разрешением имён по SRV-записям, а не по именам серверов;
    2. Уметь выбирать работающий сервер из нескольких предложенных;

Средства разработчика

Библиотека разработчика

Пакет со средствами разработки для работы с SASL называется libsasl2-dev. Пакет доступен в репозитории Astra Linux Common Edition и на диске со средствами разработки Astra Linux Special Edition. Пакет может быть установлен командой:

```
sudo apt install libsasl2-dev
```

При установке пакета устанавливаются и становятся доступны полные примеры исходных кодов клиентского и серверного приложений. Эти исходные тексты располагаются в каталоге /usr/share/doc/libsasl2-dev/examples.

Пакет sasl2-bin (тестовые программы sasl-sample-*)

Для разработки приложений следует установить пакет sasl2-bin, содержащий собранные двоичные файлы тестового клиентского и тестового серверного приложений:

```
sudo apt install sasl2-bin
```

После установки пакета sasl2-bin тестовое клиентское приложение будет размещено в файле

/usr/bin/sasl-sample-client

а тестовое серверное приложение — в файле

/usr/sbin/sasl-sample-server

Примеры исходных текстов тестовых приложения приведены в Приложении 2 и Приложении 3, однако актуальные варианты этих исходных текстов следует искать в исходных текстах актуальной версии пакета.

Справка по тестовым приложениям доступна в справочной системе MAN. Исходный текст тестовых приложений из пакета sasl2-bin может быть использован как прототип для проектирования диалогов авторизации собственных приложений, использующих SASL. Приложения-прототипы для полноценных сетевых программ см. ниже.

Использование sasl-sample-server и sasl-sample-client

Программы-примеры sasl-sample-server и sasl-sample-client предназначены для «ручной» пошаговой отработки последовательности действий при выполнении авторизации. «Ручной» режим их применения подразумевает следующий порядок действий:

1. Для организации клиент-серверного диалога используются две терминальные сессии.

2. Диалог информационного обмена между клиентов и сервером выполняется через стандартный ввод/вывод приложений (пояснения см. ниже) в формате base64. Формат можно декодировать с помощью команды base64, например:

```
base64 -d <<< R1NTQVBJ
GSSAPI
```

3. По умолчанию в тестовых программах используется имя службы: rcmd.
4. В первой терминальной сессии запускается программа /usr/sbin/sasl-sample-server, которая выдаёт отладочную информацию, и строчку данных для передачи клиенту. Например, запуск сервера с указанием принудительного выбора механизма GSSAIP:

```
/usr/sbin/sasl-sample-server -m GSSAPI
Forcing use of mechanism GSSAPI
Sending list of 1 mechanism(s)
S: R1NTQVBJ
Waiting for client mechanism...
```

   Строчки данных, предназначенные для передачи клиенту, помечаются тегом «S: » и выводятся на терминал, как указано выше, в формате base64. В примере выше клиенту должен быть передан список механизмов «S: R1NTQVBJ».

5. Во второй терминальной сессии запускается программа sasl-sample-client, которая так же выдаёт диагностическую информацию и, по мере их появления, данные для передачи серверу. При этом данные, предназначенные для передачи серверу помечаются тегом «C: » и так же передаются в кодировке base64. После запуска клиент ожидает данные от сервера, :

```
sasl-sample-client
Waiting for mechanism list from server…
```

6. Далее строчка для клиента «вручную» копируется из первого «серверного» терминала и вставляется во второй «клиентский» терминал.
   1. Пример продолжения диалога на втором терминале после ввода строчки:

```
sasl-sample-client
Waiting for mechanism list from server...
S: R1NTQVBJ
recieved 6 byte message
Choosing best mechanism from: GSSAPI
sasl-sample-client: SASL Other: GSSAPI client step 1
sasl-sample-client: SASL Other: GSSAPI Failure: no serverFQDN
error was SASL(-1): generic failure: GSSAPI Failure: no serverFQDN
```

```
sasl-sample-client: Starting SASL negotiation: generic failure
```

В примере выше клиент сообщает, что для полученного от сервера механизма GSSAPI клиенту не задан необходимый параметр — имя сервера авторизации.

2. В качестве следующего примера повторим вызов клиента с указанием сервера авторизации:

```
sasl-sample-client -n ipa0.ipadomain.ru
Waiting for mechanism list from server...
S: R1NTQVBJ
recieved 6 byte message
Choosing best mechanism from: GSSAPI
sasl-sample-client: SASL Other: GSSAPI client step 1
sasl-sample-client: SASL Other: GSSAPI Error: Unspecified GSS failure.
Minor code may provide more information (No Kerberos credentials
available (default cache: FILE:/tmp/krb5cc_1000))
error was SASL(-1): generic failure: GSSAPI Error: Unspecified GSS
failure.  Minor code may provide more information (No Kerberos
credentials available (default cache: FILE:/tmp/krb5cc_1000))
sasl-sample-client: Starting SASL negotiation: generic failure
```

В данном примере клиент сообщает, что не найдены данные авторизации Kerberos.

3. Повторим вызов клиента на машине, введённой в домен FreeIPA, выполнив перед вызовом клиента команду kinit для получения ключей Kerberos:

```
kinit admin
Password for admin@IPADOMAIN0.RU:

sasl-sample-client -n ipa0.ipadomain0.ru
Waiting for mechanism list from server...
S: R1NTQVBJ
recieved 6 byte message
Choosing best mechanism from: GSSAPI
sasl-sample-client: SASL Other: GSSAPI client step 1
sasl-sample-client: SASL Other: GSSAPI Error: Unspecified GSS failure.
Minor code may provide more information (Server
rcmd/ipa0.ipadomain0.ru@IPADOMAIN0.RU not found in Kerberos database)
error was SASL(-1): generic failure: GSSAPI Error: Unspecified GSS
failure.  Minor code may provide more information (Server
rcmd/ipa0.ipadomain0.ru@IPADOMAIN0.RU not found in Kerberos database)
sasl-sample-client: Starting SASL negotiation: generic failure
```

В данном примере клиент сообщает, что в домене не зарегистрирована служба rcmd (как указано выше, для тестовых программ это имя службы по умолчанию): `Server rcmd/ipa0.ipadomain0.ru@IPADOMAIN0.RU not found in Kerberos database`

4. Далее можно либо настроить в домене нужную службу, либо повторить вызов клиента с указанием существующей службы, например, службы HTTP, всегда присутствующей в домене FreeIPA:

```
sasl-sample-client -n ipa0.ipadomain0.ru -s http
service=http
Waiting for mechanism list from server...
S: R1NTQVBJ
recieved 6 byte message
Choosing best mechanism from: GSSAPI
sasl-sample-client: SASL Other: GSSAPI client step 1
Using mechanism GSSAPI
Preparing initial.
Sending initial response...
C: R1NTQVBJ [часть текста удалена для краткости] dJ6X2xg=
Waiting for server reply...
```

   В данном случае клиент самостоятельно определил настройки домена, получил и обработал данные, вывел на терминал текст для передачи серверу и сообщил, что ожидает ответ сервера. Текст для передачи серверу отмечен тегом «C: »:
   `C: R1NTQVBJ [часть текста удалена для краткости] dJ6X2xg=`
   Отметим, что в приведённых примерах один ответ сервера был использован для четырёх итераций настройки клиента.

7. Далее ответ клиента копируется и передаётся серверу, после чего шаги по отладке повторяются до получения требуемого результата.

Пакет cyrus-sasl (сетевые приложения server и client)

В качестве дополнительного инструмента разработчика приложений можно использовать пакет, предоставляемый разработчиками SASL. Этот пакет не входит в состав дистрибутивов и может быть получен непосредственно с WEB-сайта разработчиков: https://github.com/cyrusimap/cyrus-sasl/releases/
Например:

```
wget –no-check-certificate \
https://github.com/cyrusimap/cyrus-sasl/releases/download/cyrus-sasl-2.1.27/
cyrus-sasl-2.1.27.tar.gz
```

После загрузки архив должен быть распакован и собран:

```
tar xzf cyrus-sasl-2.1.27.tar.gz
cd cyrus-sasl-2.1.27
./configure
make
```

После сборки в подкаталоге sample будут доступны приложения server и client, функционально аналогичные описанным выше тестовым приложениям sasl-sample-server и sasl-sample-cleint, однако использующие не «ручной» режим обмена данными, а полноценный сетевой обмен. В этом же подкаталоге доступны исходные тексты этих приложений, которые можно использовать для написания собственных полноценных прикладных программ.

Клиентские приложения

Типичная последовательность действий клиентского приложения выглядит следующим образом:

1. Клиентское приложение выполняет действия по инициализации SASL;
2. Каждый раз при новом подключении клиентского приложения к серверу это приложение создаёт новый контекст подключения, который должен сохраняться всё время подключения;
3. Клиентское приложение запрашивает у сервера список поддерживаемых механизмов аутентификации (возможно, предлагая свои механизмы);
4. Выбирается механизм аутентификации;
5. Запрашивается аутентификация с помощью выбранного механизма;
6. Клиентское приложение принимает ответ сервера и передаёт его библиотеке;
7. Ответ обрабатывается библиотекой;
8. Клиентское приложение принимает ответ библиотеки и передаёт его серверу;
9. Шаги 6 — 8 повторяются до завершения аутентификации успехом или отказом.

Далее приведены примеры программного кода, реализующего указанные действия.

Инициализация клиента

Инициализация библиотеки выполняется однократно с помощью функции sasl_client_init():

```
int result;
result=sasl_client_init(callbacks); /* Описание обработчиков см.ниже */
if ( result != SASL_OK) {
      /* обработка ошибки */
}
```

Создание нового клиентского подключения

Для создания нового клиентского подключения используется функция sasl_client_new():

```
sasl_conn_t *conn;      /* Контекст соединения.
                           Должен быть сохранён на всё время подключения */
result=sasl_client_new(
      "imap",     /* Имя используемого сервиса*/
      serverFQDN, /* Полное имя сервера, к которому выполняется подключение */
      NULL, NULL, /* Необязательные локальный и удалённый IP-адреса (строки) */
      NULL,       /* Обратный вызов (зависит от контекста) */
      0,          /* Флаги безопасности */
      &conn);     /* Контекст, создаваемый при успешном подключении */
if ( result != SASL_OK) {
      /* обработка ошибки */
}
```

Согласование механизма аутентификации

Следующим шагом клиентское приложение формирует список механизмов, которые оно предлагает использовать. Список предоставляется в виде строки названий, разделённых пробелами. Для начала процесса аутентификации список передаётся SASL:

```
char * mechlist = "GSSAPI" /* список механизмов */
sasl_interact_t *client_interact=NULL;
const char *out, *mechusing;
unsigned outlen;
do {
      result=sasl_client_start(conn, /* используется ранее созданный контекст */
            mechlist,         /* список запрашиваемых механизмов */
            &client_interact, /* обработчик взаимодействия с клиентом */
            &out,             /* ответ для сервера, заполняется при успехе */
            &outlen,          /* длина ответа, , заполняется при успехе */
            &mechusing);      /* имя выбранного механизма аутентификации */
      if (result==SASL_INTERACT) {
            /* [обработка диалогов (см. ниже) ] */
      }
} while (result==SASL_INTERACT); /* проверка резульата шага */
if (result!=SASL_CONTINUE){
       /* [обработка ошибки] */
}
/* в случае успешной аутентификации код завершения SASL_CONTINUE */
```

При вызове функции sasl_client_start() создаются данные для сервера out длины outlen (если протокол не требует первоначальной передачи со стороны клиента, то out присваивается значение NULL). При этом разработчику приложения не требуется заботиться о выделении и освобождении памяти. Однако важно помнить, что полученные результаты будут стёрты при следующем вызове sasl_client_start() или sasl_client_step().

Полученные от SASL данные (out и outlen) должны быть переданы приложением серверу (см. соответствующую документацию по используемому протоколу). Например, для протокола IMAP передаваемые данные могут выглядеть как:

```
      {tag} "AUTHENTICATE" {mechusing}\r\n
      A01 AUTHENTICATE GSSAPI\r\n
```

Проверка результатов

После передачи данных от клиентского приложения серверу от сервера может быть получен один из трёх вариантов ответа:

1. Отказ в аутентификации. Процесс аутентификации прекращен. Например, "A01 NO Authentication failure in IMAP or 501 Failed in SMTP. Either retry the authentication or abort."
2. Успех аутентификации. Аутентификация прошла успешно. Например, "A01 OK Authenticated successful in IMAP or 235 Authentication successful in SMTP. Go here."
3. Требуется следующий шаг аутентификации. Например, "+ HGHDS1HAFJ= in IMAP or 334 PENCeUxFREJoU0NnbmhNWitOMjNGNndAZWx3b29kLmlubm9zb2Z0LmNvbT4= in SMTP."
   Отметим, что в протоколе IMAP это может быть пустая строка: "+ \r\n".

Приложение передаёт ответы сервера библиотеке:

```
do {
      result=sasl_client_step(conn,  /* контекст */
            in,    /* данные, переданные сервером */
            inlen, /* длина данных */
            &client_interact,  /* this should be unallocated and NULL */
            &out,     /* буфер ответа серверу при успешном завершении */
            &outlen); /* длина ответа */
      if (result==SASL_INTERACT) {
             /* [Обработка очередного шага. См. ниже] */
      }
} while (result==SASL_INTERACT || result == SASL_CONTINUE);
if (result!=SASL_OK) {
      /* [обработка ошибки] */
}
```

При этом в out и outlen библиотека SASL опять возвращается ответ, который приложение должно передать серверу аутентификации, и эти шаги повторяются до завершения аутентификации успехом или отказом.

Успешная аутентификация

Перед завершением аутентификации рекомендуется вызывать функцию sasl_client_step() ещё раз, чтобы убедиться, что никакие данные не потеряны, так как некоторые протоколы требуют продолжения обмена данными после завершения аутентификации

```
result=sasl_client_step(conn,  /* контекст соединения */
      in,    /* данные */
      inlen, /* длина данных */
```

```
      &client_interact,  /*  */
      &out,     /* заполняется при успехе */
      &outlen); /* заполняется при успехе */
if (result!=SASL_OK) {
      /* [обработка ошибки] */
}
/* Аутентификация на сервере выполнена успешно */
```

Контекст соединения должен быть сохранён после успешной аутентификации, так как далее он будет использоваться для защитного преобразования передаваемых данных.

Завершение работы

При завершении сессии соединение должно быть закрыто вызовом функции sasl_dispose():

```
sasl_dispose(&conn);
```

При полном завершении работы (например, выход из приложения), используется функция sasl_client_done():

```
sasl_client_done();
```

Или, если приложение является сервером (и, возможно, одновременно клиентом) используется sasl_done():

```
sasl_done();
```

В любом случае приложение должно использовать либо sasl_client_done() либо sasl_server_done().

Серверные приложения

С точки зрения сервера взаимодействие происходит следующим образом:

1. Сервер выполняет инициализацию SASL.
2. При получении нового запроса на подключение сервер немедленно создаёт новый контекст подключения.
3. Клиент может запросить список механизмов аутентификации, поддерживаемых сервером. Одновременно клиент может запросить аутентификацию. Клиент указывает механизм аутентификации, который он хочет использовать.
4. Сервер выполняет шаги по аутентификации и далее сохраняет контекст для выполнения защитного преобразования передаваемых данных.

Инициализация сервера

Выполняется однократно с помощью функции sasl_server_init(). Для чтения конфигурационной информации используется имя приложения.

```
int result;
result=sasl_server_init(
      callbacks,      /* Поддерживаемые обратные вызовы */
      "TestServer");  /* Имя приложения */
```

Функция sasl_server_new() должна вызываться для каждого нового подключения когда принимается подключение по сокету.

```
sasl_conn_t *conn;
int result;
/* Создание нового контекста для подключения */
result=sasl_server_new("smtp", /* Зарегистрированное название сервиса */
      NULL,       /* Полное доменное имя сервера (FQDN);
                     если NULL то используется gethostname() */
      NULL,       /* Область (realm) поиска паролей; если NULL to serverFQDN
                     Note: Не относится к Kerberos */
      NULL, NULL, /* Строки с информацией об IP-адресах */
      NULL,       /* Callbacks supported only for this connection */
      0,          /* флаги безопасности  */
      &conn);     /* создаваемый контекст соединения */
```

Когда клиент запрашивает список поддерживаемых сервером механизмов аутентификации используется функция sasl_listmech(). Результатом её работы может быть, например, строка вида:

```
"{PLAIN, GSSAPI, CRAM-MD5, DIGEST-MD5}"
result=sasl_listmech(conn,  /* Контекст соединения */
      NULL,  /* Не поддерживается */
      "{",   /* Префикс строки */
      ", ",  /* Разделитель названий механизмов */
      "}",   /* Суффикс строки */
      &result_string, /* Строка с результатом */
      &string_length, /* Длина строки с результатом */
      &number_of_mechanisms); /* Количество механизмов в строке */
```

При получении от клиента запроса на аутентификацию, сервер использует функцию sasl_server_start():

```
int result;
const char *out;
unsigned outlen;
result = sasl_server_start(conn,    /* Контекст подключения */
      mechanism_client_chose, /* Механизмы, выбранные клиентом */
      clientin,               /* Необязательные данные от клиента */
      clientinlen,                  /* Длина клиентских данных */
      &out,                         /* Ответ библиотеки.
                                    Может быть не NULL-терминированным */
      &outlen);               /* Длина ответа */
if ((result!=SASL_OK) && (result!=SASL_CONTINUE)) {
      /* Ошибка. Предаётся определённое протоколом сообщение об отказе в
         аутентификации */
}
```

```
else if (result==SASL_OK)
      /* Успешная аутентификация. Передаётся определённое протоколом сообщение
         об успешном завершении аутентификации */
else
      /* Клиенту передаются данные,
         содержащиеся в аргументе 'out' длины 'outlen'.
         Формат передачи данных определяется протоколом */
```

При обработке ответа от клиента аргумент clientin содержит ответ от клиента, декодированный из формата протокола в строку байт длины clientinlen. Этот шаг может повторять ноль или более раз. Серверное приложение должно уметь обрабатывать произвольное количество ответов, используя функцию sasl_server_step():

```
int result;
result=sasl_server_step(conn, /* Контекст соединения */
      clientin,               /* Данные клиента */
      clientinlen,                  /* Длина данных клиента */
      &out,                   /* Назначается библиотекой при успехе.
                                 Может быть не NULL-терминировано */
      &outlen);               /* Длина ответа библиотеки */
if ((result!=SASL_OK) && (result!=SASL_CONTINUE))
      /* Отказ. Клиенту передаётся определённое протоколом сообщение
         об отказе в аутентификации */
else if (result==SASL_OK)
      /* Успех. Клиенту передаётся определённое протоколом сообщение
         об успешном завершении аутентификации */
else
      /* Клиенту передаются данные, возвращенные в параметре  'out' длины
'outlen'.
         Формат передачи определяется протоколом */
```

Цикл повторяется до завершения аутентификации. Если по завершении аутентификации требуется закрыть соединение, то должна быть использована функция sasl_dispose(), так же, как и для клиентского соединения.

Общие для сервера и клиента функции

Обработчики и диалоги

При запуске приложения и вызове функции sasl_client_init() приложение должно указать, необходимые функции-обработчики и необходимые диалоги, используемые SASL для получения информации от приложения, например имя для входа и пароль.

Обработчики применяются если необходимая информация доступна до запуска приложения. SASL вызывает указанный обработчик, который должен предоставить запрашиваемые данные. Например, такими данными может быть пользовательский идентификатор, если он уже определён до начала авторизации.

Диалоги обычно используются для получения необходимых данных во взаимодействии с пользователем. В ситуациях, когда требуется запросить информацию от пользователя

(например, пароль) функции sasl_client_start() и sasl_client_step() возвращают код завершения SASL_INTERACT, при обработке которого клиентское приложение должно провести диалог с пользователем.

Любая область памяти, выделенная библиотеке SASL для обработчиков или диалогов должна сохраняться до завершения аутентификации успехом или отказом. Другими словами, память должна сохраняться до тех пор, пока функция sasl_client_start() или функция sasl_client_step() не вернёт код завершения, отличный от кодов SASL_INTERACT и SASL_CONTINUE.

Управление памятью в обработчиках и диалогах

В общем случае в библиотеке SASL за осовобождение памяти отвечает тот, кто её выделяет. При работе с диалогами библиотека управляет диалоговой структурой sasl_interact_t, однако выделение и освобождение памяти для ответов диалога управляются приложением. В любом случае, как указано выше, память должна сохраняться до завершения аутентификации успехом или отказом.

Детальное описание типов обработчиков имеется в заголовочном файле файле sasl.h. Краткая таблица:

Обработчик	Описание
SASL_CB_AUTHNAME	Имя, для которого должна быть выполнена аутентификация.
SASL_CB_USER	Имя пользователя, для которого запрашивается аутентификация. (например, почтовая служба postman, запрашивающая доставку почты пользователю tmartin может использовать AUTHNAME postman и USER tmartin)
SASL_CB_PASS	Пароль для AUTHNAME
SASL_CB_GETREALM	Область (realm) сервера

Пример использования обработчиков:

```
/* Поддерживаемые обработчики. Глобальная переменная в начале программы */
static sasl_callback_t callbacks[] = {
      { SASL_CB_GETREALM, NULL, NULL  /* использовать диалог */ },
      { SASL_CB_USER, NULL, NULL      /* использовать диалог */ },
      { SASL_CB_AUTHNAME, &getauthname_func, NULL    /* использовать обработчик
                                                        getauthname_func */ },
      { SASL_CB_PASS, &getsecret_func, NULL          /* использовать обработчик
                                                        getsecret_func */ },
      { SASL_CB_LIST_END, NULL, NULL  }
};

static int getsecret_func(sasl_conn_t *conn,
      void *context __attribute__((unused)),
      int id,
      sasl_secret_t **psecret)
```

```
{
/* [запрос пароля от пользователя] */
/* [выделение памяти под sasl_secret_t **psecret и запись пароля] */
      return SASL_OK;
}

static int getauthname_func(void *context,
      int id,
      const char **result,
      unsigned *len)
{
      if (id!=SASL_CB_AUTHNAME) return SASL_FAIL;
      /* [сохранение result и len] */
      return SASL_OK;
}
```

Где-то в основной программе:

```
      sasl_client_init(callbacks);
```

Уровни защиты данных

После успешной аутентификации требуется защитить сеанс связи для предотвращения перехвата и/или искажения передаваемых данных. Если приложение сообщает, что может поддерживать защиту данных, происходит выбор вида защиты.

Для включения поддержки защиты данных используется вызов функции sasl_setprop(), которой передаётся структура sasl_security_properties_t с присвоенным параметру max_ssf ненулевым значением:

```
sasl_security_properties_t secprops;
secprops.min_ssf = 0;
secprops.max_ssf = 256;
secprops.maxbufsize = /* см. ниже */;
secprops.property_names = NULL;
secprops.property_values = NULL;
secprops.security_flags = SASL_SEC_NOANONYMOUS; /* если применимо */
sasl_setprop(conn, SASL_SEC_PROPS, &secprops);
```

Переменная secprops будет скопирована при вызове функции sasl_setprop(), так что она может быть освобождена немедленно после вызова. Аббревиатура SSF (“security strength factor”) обозначает некоторую обобщенную характеристику защищенности соединения. Например, соединение, обеспечивающее только целостность передаваемых данных без защитного преобразования этих данных обозначается уровнем SSF равным 1. Соединение, защищенное слабым 56-битным преобразованием DES, будет иметь уровень SSF равный 56. SSF 112 разрешает использовать защитное преобразование DES3, уровень SSF 128 разрешает использовать современн

При этом параметр min_ssf определяет минимально допустимый уровень защиты.

После успешной аутентификации приложение может поверить установленный уровень защиты соединения с помощью функции sasl_getprop():

```
const int *ssfp;
result = sasl_getprop(conn, SASL_SSF, (const **) &ssfp);
if (result != SASL_OK) {
    /* ??? */
}
if (*ssfp > 0) {
    /* Установлен ненулевой уровень защиты! */
}
```

При установленном ненулевом уровне защиты приложение должно использовать для передачи данных функции sasl_encode() и sasl_decode(). Весь вывод должен до передачи во внешние лини связи кодироваться через функцию sasl_encode(); весь ввод должен декодироваться через функцию sasl_decode() до обработки приложеним. Кроме того, приложение должно корректно обрабатывать ситуации, когда функция sasl_decode() не возвращает никаких данных (случаи, когда полученные данные недопустимы).

Важным ограничением при работе с уровнями безопасности является ограничение максимального размера данных, передаваемых функциям sasl_encode()/sasl_decode(). Следует придерживаться следующих правил:

- До начала аутентификации установить параметр свойств безопасности maxbufsize равным размеру буфера, передаваемому системному вызову read() (максимальный размер данных, которые могут быть прочитаны приложением за один раз).
- После аутентификации используйте функцию sasl_getprop() чтобы получить размер SASL_MAXOUTBUF и вызывайте функцию sasl_encode() с блоками данных не больше этого размера. При большем размере блока функция sasl_encode() будет завершаться ошибкой.

Подробнее:

- Приложение устанавливает параметр SASL_SEC_PROPS с размером буфера X равным объёму данных, который оно готово читать за один раз;
- Библиотека SASL передаёт этот параметр механизму аутентификации;
- Механизм аутентификации передаёт параметр другой стороне, а другая сторона передаёт соответствующий свой размер буфера чтения.
- Механизм аутентификации вычитает размеры служебных данных, необходимых для обеспечения согласованного уровня безопасности, и возвращает результат библиотеке;
- Библиотека возвращает (как SASL_MAXOUTBUF) получившееся число как максимальный размер буфера передачи, Y.
- Функция sasl_encode() обеспечивает ограничение длины Y.

Управление памятью при передаче данных

Как обычно, за освобождение памяти отвечает тот, кто её выделяет. Библиотека SASL сохраняет данные, выданные функцией sasl_encode()/sasl_decode() до следующего вызова этой же функции на этом же соединении. Приложения не должны пытаться самостоятельно управлять этой памятью.

Перечень терминов и сокращений

API (англ. Application Programming Interface) - Программный интерфейс взаимодействия с приложениями.

SASL (англ. Simple Authentication and Security Layer — простой уровень аутентификации и безопасности) — библиотека для предоставления аутентификации и защиты данных в протоколах на основе соединений.

RFC Request for Comments — документ из серии пронумерованных информационных документов сети Интернет, определяющих технические спецификации и стандарты, используемые во Всемирной сети.

Аутентификация - подтверждение подлинности (процесса, пользователя).

Авторизация - определение прав доступа (процесса, пользователя) к ресурсам и управление этими правами.

Список литературы

1. RFC4422: Simple Authentication and Security Layer (SASL)
2. RFC7677: SCRAM-SHA-256 and SCRAM-SHA-256-PLUS Simple Authentication and Security Layer (SASL) Mechanisms
3. RFC4505: Anonymous Simple Authentication and Security Layer (SASL) Mechanism
4. RFC2195: IMAP/POP AUTHorize Extension for Simple Challenge/Response
5. RFC2433: Microsoft PPP CHAP Extensions
6. RFC2759: Microsoft PPP CHAP Extensions, Version 2
7. RFC 2743 The Generic Security Service API Version 2 update 1
8. RFC 2744 The Generic Security Service API Version 2: C-Bindings
9. RFC 1964 The Kerberos 5 GSS-API mechanism
10. RFC 4121 The Kerberos 5 GSS-API mechanism: Version 2

Версия: 2022-04-27

11. RFC 4178 The Simple and Protected GSS-API Negotiation Mechanism (SPNEGO)
12. RFC 2025 The Simple Public-Key GSS-API Mechanism (SPKM)
13. RFC 2847 LIPKEY — A Low Infrastructure Public Key Mechanism Using SPKM
14. JSR-000072 Generic Security Services API Specification 0.1
15. https://www.iana.org/assignments/sasl-mechanisms/sasl-mechanisms.xhtml
16. RFC5321 SIMPLE MAIL TRANSFER PROTOCOL
17. GNU Simple Authentication and Security Layer 1.8.1
    https://www.gnu.org/software/gsasl/manual/html_node/index.html#SEC_Contents
18. SASL Plugin Programmer's Guide
    https://www.sendmail.org/~ca/email/cyrus2/plugprog.html
19. WEB-сайт разработчиков SASL:
    https://www.cyrusimap.org/

Приложение 1. Механизмы аутентификации SASL

Механизмы SASL

Механизм аутентификации	Применимость	Стандарт
9798-M-DSA-SHA1	Общеприменим	[RFC3163]
9798-M-ECDSA-SHA1	Общеприменим	[RFC3163]
9798-M-RSA-SHA1-ENC	Общеприменим	[RFC3163]
9798-U-DSA-SHA1	Общеприменим	[RFC3163]
9798-U-ECDSA-SHA1	Общеприменим	[RFC3163]
9798-U-RSA-SHA1-ENC	Общеприменим	[RFC3163]
ANONYMOUS	Общеприменим	[RFC4505]
CRAM-MD5	Ограниченно применим	[RFC2195]
DIGEST-MD5	Устаревший	[RFC6331]
EAP-AES128	Общеприменим	[RFC7055]
EAP-AES128-PLUS	Общеприменим	[RFC7055]
EXTERNAL	Общеприменим	[RFC4422]
GS2-*	Общеприменим	[RFC5801]
GS2-KRB5	Общеприменим	[RFC5801]
GS2-KRB5-PLUS	Общеприменим	[RFC5801]
GSS-SPNEGO	Ограниченно применим	[Paul_Leach]
GSSAPI	Общеприменим	[RFC4752]
KERBEROS_V4	Устаревший	[RFC2222]
KERBEROS_V5	Общеприменим	[Simon_Josefsson]
LOGIN	Устаревший	[draft-murchison-sasl-login]
NMAS_AUTHEN	Ограниченно применим	[Mark_G._Gayman]
NMAS_LOGIN	Ограниченно применим	[Mark_G._Gayman]
NMAS-SAMBA-AUTH	Ограниченно применим	[Vince_Brimhall]
NTLM	Ограниченно применим	[Paul_Leach]
OAUTH10A	Общеприменим	[RFC7628]
OAUTHBEARER	Общеприменим	[RFC7628]
OPENID20	Общеприменим	[RFC6616]
OTP	Общеприменим	[RFC2444]
PLAIN	Общеприменим	[RFC4616]
SAML20	Общеприменим	[RFC6595]
SCRAM-*	Общеприменим	[RFC7677]

Механизм аутентификации	Применимость	Стандарт
SECURID	Общеприменим	[RFC2808]
SKEY	Устаревший	[RFC2444]
SPNEGO	Не рекомендуется к применению	[RFC5801]
SPNEGO-PLUS	Не рекомендуется к применению	[RFC5801]
XOAUTH	Устаревший	нет
XOAUTH2	Устаревший	нет

Механизмы SASL SCRAM

Механизм аутентификации	Применимость	Стандарт	Регистрационный номер
SCRAM-SHA-1	Общеприменим	[RFC5802][RFC7677]	1.3.6.1.5.5.14
SCRAM-SHA-1-PLUS	Общеприменим	[RFC5802][RFC7677]	1.3.6.1.5.5.14
SCRAM-SHA-256	Общеприменим	[RFC7677]	1.3.6.1.5.5.18
SCRAM-SHA-256-PLUS	Общеприменим	[RFC7677]	1.3.6.1.5.5.18

Приложение 2. Текст тестового клиентского приложения

```
/* sample-client.c -- sample SASL client
 * Rob Earhart
 * $Id: sample-client.c,v 1.33 2011/09/01 14:12:18 mel Exp $
 */
/*
 * Copyright (c) 1998-2003 Carnegie Mellon University.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. The name "Carnegie Mellon University" must not be used to
 *    endorse or promote products derived from this software without
 *    prior written permission. For permission or any other legal
 *    details, please contact
 *      Office of Technology Transfer
 *      Carnegie Mellon University
 *      5000 Forbes Avenue
 *      Pittsburgh, PA  15213-3890
 *      (412) 268-4387, fax: (412) 268-7395
 *      tech-transfer@andrew.cmu.edu
 *
 * 4. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by Computing Services
 *     at Carnegie Mellon University (http://www.cmu.edu/computing/)."
 *
 * CARNEGIE MELLON UNIVERSITY DISCLAIMS ALL WARRANTIES WITH REGARD TO
 * THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS, IN NO EVENT SHALL CARNEGIE MELLON UNIVERSITY BE LIABLE
 * FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN
 * AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING
 * OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 */
#include <config.h>
#include <limits.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#ifdef WIN32
# include <winsock2.h>
__declspec(dllimport) char *optarg;
__declspec(dllimport) int optind;
__declspec(dllimport) int getsubopt(char **optionp, const char * const *tokens,
char **valuep);
#else  /* WIN32 */
# include <netinet/in.h>
#endif /* WIN32 */
```

```
#include <sasl.h>
#include <saslplug.h>
#include <saslutil.h>

#ifdef macintosh
#include <sioux.h>
#include <parse_cmd_line.h>
#define MAX_ARGC (100)
int xxx_main(int argc, char *argv[]);
int main(void)
{
      char *argv[MAX_ARGC];
      int argc;
      char line[400];
      SIOUXSettings.asktosaveonclose = 0;
      SIOUXSettings.showstatusline = 1;
      argc=parse_cmd_line(MAX_ARGC,argv,sizeof(line),line);
      return xxx_main(argc,argv);
}
#define main xxx_main
#endif

#ifdef HAVE_GETOPT_H
#include <getopt.h>
#endif
#ifdef HAVE_UNISTD_H
#include <unistd.h>
#endif

#ifndef HAVE_GETSUBOPT
int getsubopt(char **optionp, const char * const *tokens, char **valuep);
#endif

static const char
build_ident[] = "$Build: sample-client " PACKAGE "-" VERSION " $";

static const char *progname = NULL;
static int verbose;

#define SAMPLE_SEC_BUF_SIZE (2048)

#define N_CALLBACKS (16)

static const char
message[] = "Come here Watson, I want you.";

char buf[SAMPLE_SEC_BUF_SIZE];

static const char *bit_subopts[] = {
#define OPT_MIN (0)
  "min",
#define OPT_MAX (1)
  "max",
  NULL
};

static const char *ext_subopts[] = {
#define OPT_EXT_SSF (0)
  "ssf",
#define OPT_EXT_ID (1)
  "id",
```

```
  NULL
};

static const char *flag_subopts[] = {
#define OPT_NOPLAIN (0)
  "noplain",
#define OPT_NOACTIVE (1)
  "noactive",
#define OPT_NODICT (2)
  "nodict",
#define OPT_FORWARDSEC (3)
  "forwardsec",
#define OPT_NOANONYMOUS (4)
  "noanonymous",
#define OPT_PASSCRED (5)
  "passcred",
  NULL
};

static const char *ip_subopts[] = {
#define OPT_IP_LOCAL (0)
  "local",
#define OPT_IP_REMOTE (1)
  "remote",
  NULL
};

static sasl_conn_t *conn = NULL;

static void
free_conn(void)
{
  if (conn)
    sasl_dispose(&conn);
}

static int
sasl_my_log(void *context __attribute__((unused)),
          int priority,
          const char *message)
{
  const char *label;

  if (! message)
    return SASL_BADPARAM;

  switch (priority) {
  case SASL_LOG_ERR:
    label = "Error";
    break;
  case SASL_LOG_NOTE:
    label = "Info";
    break;
  default:
    label = "Other";
    break;
  }

  fprintf(stderr, "%s: SASL %s: %s\n",
        progname, label, message);
```

```
  return SASL_OK;
}

static int getrealm(void *context,
                int id,
                const char **availrealms __attribute__((unused)),
                const char **result)
{
  if (id!=SASL_CB_GETREALM) return SASL_FAIL;

  *result=(char *) context;

  return SASL_OK;
}

static int
getpath(void *context,
      const char ** path)
{
  const char *searchpath = (const char *) context;

  if (! path)
    return SASL_BADPARAM;

  if (searchpath) {
      *path = searchpath;
  } else {
      *path = PLUGINDIR;
  }

  return SASL_OK;
}

static int
simple(void *context,
       int id,
       const char **result,
       unsigned *len)
{
  const char *value = (const char *)context;

  if (! result)
    return SASL_BADPARAM;

  switch (id) {
  case SASL_CB_USER:
    *result = value;
    if (len)
      *len = value ? (unsigned) strlen(value) : 0;
    break;
  case SASL_CB_AUTHNAME:
    *result = value;
    if (len)
      *len = value ? (unsigned) strlen(value) : 0;
    break;
  case SASL_CB_LANGUAGE:
    *result = NULL;
    if (len)
      *len = 0;
    break;
  default:
```

```
    return SASL_BADPARAM;
  }

  printf("returning OK: %s\n", *result);

  return SASL_OK;
}

#ifndef HAVE_GETPASSPHRASE
static char *
getpassphrase(const char *prompt)
{
  return getpass(prompt);
}
#endif /* ! HAVE_GETPASSPHRASE */

static int
getsecret(sasl_conn_t *conn,
        void *context __attribute__((unused)),
        int id,
        sasl_secret_t **psecret)
{
  char *password;
  unsigned len;

  if (! conn || ! psecret || id != SASL_CB_PASS)
    return SASL_BADPARAM;

  password = getpassphrase("Password: ");
  if (! password)
    return SASL_FAIL;

  len = (unsigned) strlen(password);

  *psecret = (sasl_secret_t *) malloc(sizeof(sasl_secret_t) + len);

  if (! *psecret) {
    memset(password, 0, len);
    return SASL_NOMEM;
  }

  (*psecret)->len = len;
  strcpy((char *)(*psecret)->data, password);
  memset(password, 0, len);

  return SASL_OK;
}

static int
prompt(void *context __attribute__((unused)),
       int id,
       const char *challenge,
       const char *prompt,
       const char *defresult,
       const char **result,
       unsigned *len)
{
  if ((id != SASL_CB_ECHOPROMPT && id != SASL_CB_NOECHOPROMPT)
      || !prompt || !result || !len)
    return SASL_BADPARAM;
```

```
  if (! defresult)
    defresult = "";

  fputs(prompt, stdout);
  if (challenge)
    printf(" [challenge: %s]", challenge);
  printf(" [%s]: ", defresult);
  fflush(stdout);

  if (id == SASL_CB_ECHOPROMPT) {
    char *original = getpassphrase("");
    if (! original)
      return SASL_FAIL;
    if (*original)
      *result = strdup(original);
    else
      *result = strdup(defresult);
    memset(original, 0L, strlen(original));
  } else {
    char buf[1024];
    fgets(buf, 1024, stdin);
    if (buf[0]) {
      *result = strdup(buf);
    } else {
      *result = strdup(defresult);
    }
    memset(buf, 0L, sizeof(buf));
  }
  if (! *result)
    return SASL_NOMEM;

  *len = (unsigned) strlen(*result);

  return SASL_OK;
}

static void
sasldebug(int why, const char *what, const char *errstr)
{
  fprintf(stderr, "%s: %s: %s",
        progname,
        what,
        sasl_errstring(why, NULL, NULL));
  if (errstr)
    fprintf(stderr, " (%s)\n", errstr);
  else
    putc('\n', stderr);
}

static void
saslfail(int why, const char *what, const char *errstr)
{
  sasldebug(why, what, errstr);
  free_conn();
  sasl_done();
  exit(EXIT_FAILURE);
}

static void
fail(const char *what)
{
```

```
  fprintf(stderr, "%s: %s\n",
        progname, what);
  exit(EXIT_FAILURE);
}

static void
osfail()
{
  perror(progname);
  exit(EXIT_FAILURE);
}

static void
samp_send(const char *buffer,
        unsigned length)
{
  char *buf;
  unsigned len, alloclen;
  int result;

  alloclen = ((length / 3) + 1) * 4 + 1;
  buf = malloc(alloclen);
  if (! buf)
    osfail();
  result = sasl_encode64(buffer, length, buf, alloclen, &len);
  if (result != SASL_OK)
    saslfail(result, "Encoding data in base64", NULL);
  printf("C: %s\n", buf);
  free(buf);
}

static unsigned
samp_recv()
{
  unsigned len;
  int result;

  if (! fgets(buf, SAMPLE_SEC_BUF_SIZE, stdin)) {
    fail("Unable to parse input");
  }

  if (strncmp(buf, "S: ", 3) != 0) {
    fail("Line must start with 'S: '");
  }

  len = strlen(buf);
  if (len > 0 && buf[len-1] == '\n') {
      buf[len-1] = '\0';
  }

  result = sasl_decode64(buf + 3, (unsigned) strlen(buf + 3), buf,
                  SAMPLE_SEC_BUF_SIZE, &len);
  if (result != SASL_OK)
    saslfail(result, "Decoding data from base64", NULL);
  buf[len] = '\0';
  printf("recieved %d byte message\n",len);
  if (verbose) { printf("got '%s'\n", buf); }
  return len;
}

int
```

```
main(int argc, char *argv[])
{
  int c = 0;
  int errflag = 0;
  int result;
  sasl_security_properties_t secprops;
  sasl_ssf_t extssf = 0;
  const char *ext_authid = NULL;
  char *options, *value;
  const char *data;
  const char *chosenmech;
  int serverlast = 0;
  unsigned len;
  int clientfirst = 1;
  sasl_callback_t callbacks[N_CALLBACKS], *callback;
  char *realm = NULL;
  char *mech = NULL,
    *iplocal = NULL,
    *ipremote = NULL,
    *searchpath = NULL,
    *service = "rcmd",
    *fqdn = "",
    *userid = NULL,
    *authid = NULL;
  sasl_ssf_t *ssf;

#ifdef WIN32
  /* initialize winsock */
    WSADATA wsaData;

    result = WSAStartup( MAKEWORD(2, 0), &wsaData );
    if ( result != 0) {
      saslfail(SASL_FAIL, "Initializing WinSockets", NULL);
    }
#endif

  progname = strrchr(argv[0], HIER_DELIMITER);
  if (progname)
    progname++;
  else
    progname = argv[0];

  /* Init defaults... */
  memset(&secprops, 0L, sizeof(secprops));
  secprops.maxbufsize = SAMPLE_SEC_BUF_SIZE;
  secprops.max_ssf = UINT_MAX;

  verbose = 0;
  while ((c = getopt(argc, argv, "vhldb:e:m:f:i:p:r:s:n:u:a:?")) != EOF)
    switch (c) {
    case 'v':
      verbose = 1;
      break;
    case 'b':
      options = optarg;
      while (*options != '\0')
      switch(getsubopt(&options, (const char * const *)bit_subopts, &value)) {
      case OPT_MIN:
        if (! value)
          errflag = 1;
        else
```

```
      secprops.min_ssf = atoi(value);
    break;
  case OPT_MAX:
    if (! value)
      errflag = 1;
    else
      secprops.max_ssf = atoi(value);
    break;
  default:
    errflag = 1;
    break;
    }
  break;

case 'l':
  serverlast = SASL_SUCCESS_DATA;
  break;

case 'd':
  clientfirst = 0;
  break;

case 'e':
  options = optarg;
  while (*options != '\0')
  switch(getsubopt(&options, (const char * const *)ext_subopts, &value)) {
  case OPT_EXT_SSF:
    if (! value)
      errflag = 1;
    else
      extssf = atoi(value);
    break;
  case OPT_MAX:
    if (! value)
      errflag = 1;
    else
      ext_authid = value;
    break;
  default:
    errflag = 1;
    break;
    }
  break;

case 'm':
  mech = optarg;
  break;

case 'f':
  options = optarg;
  while (*options != '\0') {
  switch(getsubopt(&options, (const char * const *)flag_subopts, &value)) {
  case OPT_NOPLAIN:
    secprops.security_flags |= SASL_SEC_NOPLAINTEXT;
    break;
  case OPT_NOACTIVE:
    secprops.security_flags |= SASL_SEC_NOACTIVE;
    break;
  case OPT_NODICT:
    secprops.security_flags |= SASL_SEC_NODICTIONARY;
    break;
```

```
  case OPT_FORWARDSEC:
    secprops.security_flags |= SASL_SEC_FORWARD_SECRECY;
    break;
  case OPT_NOANONYMOUS:
    secprops.security_flags |= SASL_SEC_NOANONYMOUS;
    break;
  case OPT_PASSCRED:
    secprops.security_flags |= SASL_SEC_PASS_CREDENTIALS;
    break;
  default:
    errflag = 1;
    break;
    }
  if (value) errflag = 1;
  }
  break;

case 'i':
  options = optarg;
  while (*options != '\0')
  switch(getsubopt(&options, (const char * const *)ip_subopts, &value)) {
  case OPT_IP_LOCAL:
    if (! value)
      errflag = 1;
    else
      iplocal = value;
    break;
  case OPT_IP_REMOTE:
    if (! value)
      errflag = 1;
    else
      ipremote = value;
    break;
  default:
    errflag = 1;
    break;
    }
  break;

case 'p':
  searchpath = optarg;
  break;

case 'r':
  realm = optarg;
  break;

case 's':
  service=malloc(1000);
  strcpy(service,optarg);
  /*      service = optarg;*/
  printf("service=%s\n",service);
  break;

case 'n':
  fqdn = optarg;
  break;

case 'u':
  userid = optarg;
  break;
```

```
    case 'a':
      authid = optarg;
      break;

    default:                  /* unknown flag */
      errflag = 1;
      break;
    }

  if (optind != argc) {
    /* We don't *have* extra arguments */
    errflag = 1;
  }

  if (errflag) {
    fprintf(stderr, "%s: Usage: %s [-b min=N,max=N] [-e ssf=N,id=ID] [-m MECH]
[-f FLAGS] [-i local=IP,remote=IP] [-p PATH] [-s NAME] [-n FQDN] [-u ID] [-a
ID]\n"
          "\t-b ...\t#bits to use for encryption\n"
          "\t\tmin=N\tminumum #bits to use (1 => integrity)\n"
          "\t\tmax=N\tmaximum #bits to use\n"
          "\t-e ...\tassume external encryption\n"
          "\t\tssf=N\texternal mech provides N bits of encryption\n"
          "\t\tid=ID\texternal mech provides authentication id ID\n"
          "\t-m MECH\tforce use of MECH for security\n"
          "\t-f ...\tset security flags\n"
          "\t\tnoplain\t\trequire security vs. passive attacks\n"
          "\t\tnoactive\trequire security vs. active attacks\n"
          "\t\tnodict\t\trequire security vs. passive dictionary attacks\n"
          "\t\tforwardsec\trequire forward secrecy\n"
          "\t\tmaximum\t\trequire all security flags\n"
          "\t\tpasscred\tattempt to pass client credentials\n"
          "\t-i ...\tset IP addresses (required by some mechs)\n"
          "\t\tlocal=IP;PORT\tset local address to IP, port PORT\n"
          "\t\tremote=IP;PORT\tset remote address to IP, port PORT\n"
          "\t-p PATH\tcolon-seperated search path for mechanisms\n"
          "\t-r REALM\trealm to use"
          "\t-s NAME\tservice name pass to mechanisms\n"
          "\t-n FQDN\tserver fully-qualified domain name\n"
          "\t-u ID\tuser (authorization) id to request\n"
          "\t-a ID\tid to authenticate as\n"
          "\t-d\tDisable client-send-first\n"
          "\t-l\tEnable server-send-last\n",
          progname, progname);
    exit(EXIT_FAILURE);
  }

  /* Fill in the callbacks that we're providing... */
  callback = callbacks;

  /* log */
  callback->id = SASL_CB_LOG;
  callback->proc = (sasl_callback_ft)&sasl_my_log;
  callback->context = NULL;
  ++callback;

  /* getpath */
  if (searchpath) {
    callback->id = SASL_CB_GETPATH;
    callback->proc = (sasl_callback_ft)&getpath;
```

```
  callback->context = searchpath;
  ++callback;
}

/* user */
if (userid) {
  callback->id = SASL_CB_USER;
  callback->proc = (sasl_callback_ft)&simple;
  callback->context = userid;
  ++callback;
}

/* authname */
if (authid) {
  callback->id = SASL_CB_AUTHNAME;
  callback->proc = (sasl_callback_ft)&simple;
  callback->context = authid;
  ++callback;
}

if (realm!=NULL)
{
  callback->id = SASL_CB_GETREALM;
  callback->proc = (sasl_callback_ft)&getrealm;
  callback->context = realm;
  callback++;
}

/* password */
callback->id = SASL_CB_PASS;
callback->proc = (sasl_callback_ft)&getsecret;
callback->context = NULL;
++callback;

/* echoprompt */
callback->id = SASL_CB_ECHOPROMPT;
callback->proc = (sasl_callback_ft)&prompt;
callback->context = NULL;
++callback;

/* noechoprompt */
callback->id = SASL_CB_NOECHOPROMPT;
callback->proc = (sasl_callback_ft)&prompt;
callback->context = NULL;
++callback;

/* termination */
callback->id = SASL_CB_LIST_END;
callback->proc = NULL;
callback->context = NULL;
++callback;

if (N_CALLBACKS < callback - callbacks)
  fail("Out of callback space; recompile with larger N_CALLBACKS");

result = sasl_client_init(callbacks);
if (result != SASL_OK)
  saslfail(result, "Initializing libsasl", NULL);

result = sasl_client_new(service,
                   fqdn,
```

```
                  iplocal,ipremote,
                  NULL,serverlast,
                  &conn);
if (result != SASL_OK)
  saslfail(result, "Allocating sasl connection state", NULL);

if(extssf) {
    result = sasl_setprop(conn,
                  SASL_SSF_EXTERNAL,
                  &extssf);

    if (result != SASL_OK)
      saslfail(result, "Setting external SSF", NULL);
}

if(ext_authid) {
    result = sasl_setprop(conn,
                  SASL_AUTH_EXTERNAL,
                  &ext_authid);

    if (result != SASL_OK)
      saslfail(result, "Setting external authid", NULL);
}

result = sasl_setprop(conn,
               SASL_SEC_PROPS,
               &secprops);

if (result != SASL_OK)
  saslfail(result, "Setting security properties", NULL);

puts("Waiting for mechanism list from server...");
len = samp_recv();

if (mech) {
  printf("Forcing use of mechanism %s\n", mech);
  strncpy(buf, mech, SAMPLE_SEC_BUF_SIZE);
  buf[SAMPLE_SEC_BUF_SIZE - 1] = '\0';
}

printf("Choosing best mechanism from: %s\n", buf);

if(clientfirst) {
    result = sasl_client_start(conn,
                       buf,
                       NULL,
                       &data,
                       &len,
                       &chosenmech);
} else {
    data = "";
    len = 0;
    result = sasl_client_start(conn,
                       buf,
                       NULL,
                       NULL,
                       0,
                       &chosenmech);
}
```

```
  if (result != SASL_OK && result != SASL_CONTINUE) {
      printf("error was %s\n", sasl_errdetail(conn));
      saslfail(result, "Starting SASL negotiation", NULL);
  }

  printf("Using mechanism %s\n", chosenmech);
  strcpy(buf, chosenmech);
  if (data) {
    if (SAMPLE_SEC_BUF_SIZE - strlen(buf) - 1 < len)
      fail("Not enough buffer space");
    puts("Preparing initial.");
    memcpy(buf + strlen(buf) + 1, data, len);
    len += (unsigned) strlen(buf) + 1;
    data = NULL;
  } else {
    len = (unsigned) strlen(buf);
  }

  puts("Sending initial response...");
  samp_send(buf, len);

  while (result == SASL_CONTINUE) {
    puts("Waiting for server reply...");
    len = samp_recv();
    result = sasl_client_step(conn, buf, len, NULL,
                        &data, &len);
    if (result != SASL_OK && result != SASL_CONTINUE)
      saslfail(result, "Performing SASL negotiation", NULL);
    if (data && len) {
      puts("Sending response...");
      samp_send(data, len);
    } else if (result != SASL_OK || !serverlast) {
      samp_send("",0);
    }

  }
  puts("Negotiation complete");

  result = sasl_getprop(conn, SASL_USERNAME, (const void **)&data);
  if (result != SASL_OK)
    sasldebug(result, "username", NULL);
  else
    printf("Username: %s\n", data);

#define CLIENT_MSG1 "client message 1"
#define SERVER_MSG1 "srv message 1"

  result = sasl_getprop(conn, SASL_SSF, (const void **)&ssf);
  if (result != SASL_OK)
    sasldebug(result, "ssf", NULL);
  else
    printf("SSF: %d\n", *ssf);

 printf("Waiting for encoded message...\n");
 len=samp_recv();
 {
      unsigned int recv_len;
      const char *recv_data;
      result=sasl_decode(conn,buf,len,&recv_data,&recv_len);
      if (result != SASL_OK)
          saslfail(result, "sasl_decode", NULL);
```

```
      printf("recieved decoded message '%s'\n",recv_data);
      if(strcmp(recv_data,SERVER_MSG1)!=0)
          saslfail(1,"recive decoded server message",NULL);
 }
  result=sasl_encode(conn,CLIENT_MSG1,sizeof(CLIENT_MSG1),
      &data,&len);
  if (result != SASL_OK)
      saslfail(result, "sasl_encode", NULL);
  printf("sending encrypted message '%s'\n",CLIENT_MSG1);
  samp_send(data,len);

  free_conn();
  sasl_done();

#ifdef WIN32
  WSACleanup();
#endif
  return (EXIT_SUCCESS);
}
```

Приложение 3. Текст тестового серверного приложения

```
/* sample-server.c -- sample SASL server
 * Rob Earhart
 * $Id: sample-server.c,v 1.34 2011/09/01 14:12:18 mel Exp $
 */
/*
 * Copyright (c) 1998-2003 Carnegie Mellon University.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. The name "Carnegie Mellon University" must not be used to
 *    endorse or promote products derived from this software without
 *    prior written permission. For permission or any other legal
 *    details, please contact
 *      Office of Technology Transfer
 *      Carnegie Mellon University
 *      5000 Forbes Avenue
 *      Pittsburgh, PA  15213-3890
 *      (412) 268-4387, fax: (412) 268-7395
 *      tech-transfer@andrew.cmu.edu
 *
 * 4. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by Computing Services
 *     at Carnegie Mellon University (http://www.cmu.edu/computing/)."
 *
 * CARNEGIE MELLON UNIVERSITY DISCLAIMS ALL WARRANTIES WITH REGARD TO
 * THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS, IN NO EVENT SHALL CARNEGIE MELLON UNIVERSITY BE LIABLE
```

```
 * FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN
 * AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING
 * OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 */

#include <config.h>
#include <limits.h>
#include <stdio.h>

#ifdef HAVE_GETOPT_H
#include <getopt.h>
#endif
#ifdef HAVE_UNISTD_H
#include <unistd.h>
#endif

#ifdef WIN32
# include <winsock2.h>
__declspec(dllimport) char *optarg;
__declspec(dllimport) int optind;
__declspec(dllimport) int getsubopt(char **optionp, const char * const *tokens,
char **valuep);
#define HAVE_GETSUBOPT
#else /* WIN32 */
# include <netinet/in.h>
#endif /* WIN32 */
#include <sasl.h>
#include <saslplug.h>
#include <saslutil.h>

#ifndef HAVE_GETSUBOPT
int getsubopt(char **optionp, const char * const *tokens, char **valuep);
#endif

static const char
build_ident[] = "$Build: sample-server " PACKAGE "-" VERSION " $";

static const char *progname = NULL;
static int verbose;

/* Note: if this is changed, change it in samp_read(), too. */
#define SAMPLE_SEC_BUF_SIZE (2048)

static const char
message[] = "Come here Watson, I want you.";

char buf[SAMPLE_SEC_BUF_SIZE];

static const char *bit_subopts[] = {
#define OPT_MIN (0)
  "min",
#define OPT_MAX (1)
  "max",
  NULL
};

static const char *ext_subopts[] = {
#define OPT_EXT_SSF (0)
  "ssf",
#define OPT_EXT_ID (1)
```

```
  "id",
  NULL
};

static const char *flag_subopts[] = {
#define OPT_NOPLAIN (0)
  "noplain",
#define OPT_NOACTIVE (1)
  "noactive",
#define OPT_NODICT (2)
  "nodict",
#define OPT_FORWARDSEC (3)
  "forwardsec",
#define OPT_NOANONYMOUS (4)
  "noanonymous",
#define OPT_PASSCRED (5)
  "passcred",
  NULL
};

static const char *ip_subopts[] = {
#define OPT_IP_LOCAL (0)
  "local",
#define OPT_IP_REMOTE (1)
  "remote",
  NULL
};

char *mech = NULL,
  *iplocal = NULL,
  *ipremote = NULL,
  *searchpath = NULL,
  *service = "rcmd",
  *localdomain = NULL,
  *userdomain = NULL;
sasl_conn_t *conn = NULL;

static void
free_conn(void)
{
  if (conn)
    sasl_dispose(&conn);
}

static int
sasl_my_log(void *context __attribute__((unused)),
          int priority,
          const char *message)
{
  const char *label;

  if (! message)
    return SASL_BADPARAM;

  switch (priority) {
  case SASL_LOG_ERR:
    label = "Error";
    break;
  case SASL_LOG_NOTE:
    label = "Info";
    break;
```

```
  default:
    label = "Other";
    break;
  }

  fprintf(stderr, "%s: SASL %s: %s\n",
        progname, label, message);

  return SASL_OK;
}

static int
getpath(void *context __attribute__((unused)),
      char ** path)
{
  if (! path)
    return SASL_BADPARAM;

  if (searchpath) {
    *path = searchpath;
  } else {
    *path = PLUGINDIR;
  }

  return SASL_OK;
}

static sasl_callback_t callbacks[] = {
  {
    SASL_CB_LOG, (sasl_callback_ft)&sasl_my_log, NULL
  }, {
    SASL_CB_GETPATH, (sasl_callback_ft)&getpath, NULL
  }, {
    SASL_CB_LIST_END, NULL, NULL
  }
};

static void
sasldebug(int why, const char *what, const char *errstr)
{
  fprintf(stderr, "%s: %s: %s",
        progname,
        what,
        sasl_errstring(why, NULL, NULL));
  if (errstr)
    fprintf(stderr, " (%s)\n", errstr);
  else
    putc('\n', stderr);
}

static void
saslfail(int why, const char *what, const char *errstr)
{
  sasldebug(why, what, errstr);
  exit(EXIT_FAILURE);
}

static void
fail(const char *what)
{
  fprintf(stderr, "%s: %s\n",
```

```
        progname, what);
  exit(EXIT_FAILURE);
}

static void
osfail()
{
  perror(progname);
  exit(EXIT_FAILURE);
}

static void
samp_send(const char *buffer,
        unsigned length)
{
  char *buf;
  unsigned len, alloclen;
  int result;

  alloclen = ((length / 3) + 1) * 4 + 1;
  buf = malloc(alloclen);
  if (! buf)
    osfail();

  result = sasl_encode64(buffer, length, buf, alloclen, &len);
  if (result != SASL_OK)
    saslfail(result, "Encoding data in base64", NULL);
  printf("S: %s\n", buf);
  free(buf);
}

static unsigned
samp_recv()
{
  unsigned len;
  int result;

  if (! fgets(buf, SAMPLE_SEC_BUF_SIZE, stdin)) {
    fail("Unable to parse input");
  }

  if (strncmp(buf, "C: ", 3) != 0) {
    fail("Line must start with 'C: '");
  }

  len = strlen(buf);
  if (len > 0 && buf[len-1] == '\n') {
      buf[len-1] = '\0';
  }

  result = sasl_decode64(buf + 3, (unsigned) strlen(buf + 3), buf,
                  SAMPLE_SEC_BUF_SIZE, &len);
  if (result != SASL_OK)
    saslfail(result, "Decoding data from base64", NULL);
  buf[len] = '\0';
  printf("got '%s'\n", buf);
  return len;
}


int
```

```
main(int argc, char *argv[])
{
  int c = 0;
  int errflag = 0;
  int result;
  sasl_security_properties_t secprops;
  sasl_ssf_t extssf = 0;
  const char *ext_authid = NULL;
  char *options, *value;
  unsigned len, count;
  const char *data;
  int serverlast = 0;
  sasl_ssf_t *ssf;

#ifdef WIN32
  /* initialize winsock */
    WSADATA wsaData;

    result = WSAStartup( MAKEWORD(2, 0), &wsaData );
    if ( result != 0) {
      saslfail(SASL_FAIL, "Initializing WinSockets", NULL);
    }
#endif

  progname = strrchr(argv[0], HIER_DELIMITER);
  if (progname)
    progname++;
  else
    progname = argv[0];

  /* Init defaults... */
  memset(&secprops, 0L, sizeof(secprops));
  secprops.maxbufsize = SAMPLE_SEC_BUF_SIZE;
  secprops.max_ssf = UINT_MAX;

  verbose = 0;
  while ((c = getopt(argc, argv, "vlhb:e:m:f:i:p:s:d:u:?")) != EOF)
    switch (c) {
    case 'v':
      verbose = 1;
      break;
    case 'b':
      options = optarg;
      while (*options != '\0')
      switch(getsubopt(&options, (const char * const *)bit_subopts, &value)) {
      case OPT_MIN:
        if (! value)
          errflag = 1;
        else
          secprops.min_ssf = atoi(value);
        break;
      case OPT_MAX:
        if (! value)
          errflag = 1;
        else
          secprops.max_ssf = atoi(value);
        break;
      default:
        errflag = 1;
        break;
        }
```

```
  break;

case 'e':
  options = optarg;
  while (*options != '\0')
  switch(getsubopt(&options, (const char * const *)ext_subopts, &value)) {
  case OPT_EXT_SSF:
    if (! value)
      errflag = 1;
    else
      extssf = atoi(value);
    break;
  case OPT_MAX:
    if (! value)
      errflag = 1;
    else
      ext_authid = value;
    break;
  default:
    errflag = 1;
    break;
    }
  break;

case 'm':
  mech = optarg;
  break;

case 'f':
  options = optarg;
  while (*options != '\0') {
  switch(getsubopt(&options, (const char * const *)flag_subopts, &value)) {
  case OPT_NOPLAIN:
    secprops.security_flags |= SASL_SEC_NOPLAINTEXT;
    break;
  case OPT_NOACTIVE:
    secprops.security_flags |= SASL_SEC_NOACTIVE;
    break;
  case OPT_NODICT:
    secprops.security_flags |= SASL_SEC_NODICTIONARY;
    break;
  case OPT_FORWARDSEC:
    secprops.security_flags |= SASL_SEC_FORWARD_SECRECY;
    break;
  case OPT_NOANONYMOUS:
    secprops.security_flags |= SASL_SEC_NOANONYMOUS;
    break;
  case OPT_PASSCRED:
    secprops.security_flags |= SASL_SEC_PASS_CREDENTIALS;
    break;
  default:
    errflag = 1;
    break;
    }
  if (value) errflag = 1;
  }
  break;

case 'l':
  serverlast = SASL_SUCCESS_DATA;
  break;
```

```
    case 'i':
      options = optarg;
      while (*options != '\0')
      switch(getsubopt(&options, (const char * const *)ip_subopts, &value)) {
      case OPT_IP_LOCAL:
        if (! value)
          errflag = 1;
        else
          iplocal = value;
        break;
      case OPT_IP_REMOTE:
        if (! value)
          errflag = 1;
        else
          ipremote = value;
        break;
      default:
        errflag = 1;
        break;
        }
      break;

    case 'p':
      searchpath = optarg;
      break;

    case 's':
      service = optarg;
      break;

    case 'd':
      localdomain = optarg;
      break;

    case 'u':
      userdomain = optarg;
      break;

    default:
      errflag = 1;
      break;
    }

  if (optind != argc) {

    errflag = 1;
  }

  if (errflag) {
    fprintf(stderr, "%s: Usage: %s [-b min=N,max=N] [-e ssf=N,id=ID] [-m MECH]
[-f FLAGS] [-i local=IP,remote=IP] [-p PATH] [-d DOM] [-u DOM] [-s NAME]\n"
          "\t-b ...\t#bits to use for encryption\n"
          "\t\tmin=N\tminumum #bits to use (1 => integrity)\n"
          "\t\tmax=N\tmaximum #bits to use\n"
          "\t-e ...\tassume external encryption\n"
          "\t\tssf=N\texternal mech provides N bits of encryption\n"
          "\t\tid=ID\texternal mech provides authentication id ID\n"
          "\t-m MECH\tforce use of MECH for security\n"
          "\t-f ...\tset security flags\n"
          "\t\tnoplain\t\trequire security vs. passive attacks\n"
```

```
        "\t\tnoactive\trequire security vs. active attacks\n"
        "\t\tnodict\t\trequire security vs. passive dictionary attacks\n"
        "\t\tforwardsec\trequire forward secrecy\n"
        "\t\tmaximum\t\trequire all security flags\n"
        "\t\tpasscred\tattempt to receive client credentials\n"
        "\t-i ...\tset IP addresses (required by some mechs)\n"
        "\t\tlocal=IP;PORT\tset local address to IP, port PORT\n"
        "\t\tremote=IP;PORT\tset remote address to IP, port PORT\n"
        "\t-p PATH\tcolon-seperated search path for mechanisms\n"
        "\t-s NAME\tservice name to pass to mechanisms\n"
        "\t-d DOM\tlocal server domain\n"
        "\t-u DOM\tuser domain\n"
        "\t-l\tenable server-send-last\n",
        progname, progname);
  exit(EXIT_FAILURE);
}

result = sasl_server_init(callbacks, "sample");
if (result != SASL_OK)
  saslfail(result, "Initializing libsasl", NULL);

atexit(&sasl_done);

result = sasl_server_new(service,
                  localdomain,
                  userdomain,
                  iplocal,
                  ipremote,
                  NULL,
                  serverlast,
                  &conn);
if (result != SASL_OK)
  saslfail(result, "Allocating sasl connection state", NULL);

atexit(&free_conn);

if(extssf) {
    result = sasl_setprop(conn,
                   SASL_SSF_EXTERNAL,
                   &extssf);

    if (result != SASL_OK)
      saslfail(result, "Setting external SSF", NULL);
}

if(ext_authid) {
    result = sasl_setprop(conn,
                   SASL_AUTH_EXTERNAL,
                   &ext_authid);

    if (result != SASL_OK)
      saslfail(result, "Setting external authid", NULL);
}

result = sasl_setprop(conn,
                SASL_SEC_PROPS,
                &secprops);

if (result != SASL_OK)
  saslfail(result, "Setting security properties", NULL);
```

```
  if (mech) {
    printf("Forcing use of mechanism %s\n", mech);
    data = strdup(mech);
    if (! data)
      osfail();
    len = (unsigned) strlen(data);
    count = 1;
  } else {
    puts("Generating client mechanism list...");
    result = sasl_listmech(conn,
                     ext_authid,
                     NULL,
                     " ",
                     NULL,
                     &data,
                     &len,
                     &count);
    if (result != SASL_OK)
      saslfail(result, "Generating client mechanism list", NULL);
  }

  printf("Sending list of %d mechanism(s)\n", count);
  samp_send(data, len);

  if(mech) {
      free((void *)data);
  }

  puts("Waiting for client mechanism...");
  len = samp_recv();
  if (mech && strcasecmp(mech, buf))
    fail("Client chose something other than the mandatory mechanism");
  if (strlen(buf) < len) {
    /* Hmm, there's an initial response here */
    data = buf + strlen(buf) + 1;
    len = len - (unsigned) strlen(buf) - 1;
  } else {
    data = NULL;
    len = 0;
  }
  result = sasl_server_start(conn,
                       buf,
                       data,
                       len,
                       &data,
                       &len);
  if (result != SASL_OK && result != SASL_CONTINUE)
    saslfail(result, "Starting SASL negotiation",
sasl_errstring(result,NULL,NULL));

  while (result == SASL_CONTINUE) {
    if (data) {
      puts("Sending response...");
      samp_send(data, len);
    } else
      fail("No data to send--something's wrong");
    puts("Waiting for client reply...");
    len = samp_recv();
    data = NULL;
    result = sasl_server_step(conn, buf, len,
                         &data, &len);
```

```
    if (result != SASL_OK && result != SASL_CONTINUE)
      saslfail(result, "Performing SASL negotiation",
sasl_errstring(result,NULL,NULL));
  }
  puts("Negotiation complete");

  if(serverlast&&data) {
      printf("might need additional send:\n");
      samp_send(data,len);
  }

  result = sasl_getprop(conn, SASL_USERNAME, (const void **)&data);
  if (result != SASL_OK)
    sasldebug(result, "username", NULL);
  else
    printf("Username: %s\n", data ? data : "(NULL)");

  result = sasl_getprop(conn, SASL_DEFUSERREALM, (const void **)&data);
  if (result != SASL_OK)
    sasldebug(result, "realm", NULL);
  else
    printf("Realm: %s\n", data ? data : "(NULL)");

  result = sasl_getprop(conn, SASL_SSF, (const void **)&ssf);
  if (result != SASL_OK)
    sasldebug(result, "ssf", NULL);
  else
    printf("SSF: %d\n", *ssf);
#define CLIENT_MSG1 "client message 1"
#define SERVER_MSG1 "srv message 1"
  result=sasl_encode(conn,SERVER_MSG1,sizeof(SERVER_MSG1),
      &data,&len);
  if (result != SASL_OK)
      saslfail(result, "sasl_encode", NULL);
  printf("sending encrypted message '%s'\n",SERVER_MSG1);
  samp_send(data,len);
  printf("Waiting for encrypted message...\n");
  len=samp_recv();
 {
      unsigned int recv_len;
      const char *recv_data;
      result=sasl_decode(conn,buf,len,&recv_data,&recv_len);
      if (result != SASL_OK)
      saslfail(result, "sasl_encode", NULL);
    printf("recieved decoded message '%s'\n",recv_data);
    if(strcmp(recv_data,CLIENT_MSG1)!=0)
      saslfail(1,"recive decoded server message",NULL);
 }

#ifdef WIN32
  WSACleanup();
#endif

  return (EXIT_SUCCESS);
}
```