

Настройка аутентификации веб-сервисов с SSL-шифрованием

EddyEm

12 июля 2010 г.

1 Настройка HTTPS

1.1 Конфигурирование apache

Для запуска SSL-модуля нам необходимо будет отредактировать файл

`/etc/httpd/modules.d/40_mod_ssl.conf`

Вполне возможно, что в вашей реализации apache этот файл будет располагаться в другой директории, кроме того, настройки виртуального хоста для SSL-соединения можно записать и в основной конфигурационный файл. Во-первых, следует убедиться, что в файле `40_mod_ssl.conf` раскомментирована строка

```
LoadModule ssl_module modules/mod_ssl.so
```

Во-вторых, если вы хотите изменить стандартный порт для https-соединения, отредактируйте строку с директивой `Listen 443`, иначе— оставьте ее без изменений. Остальные директивы пока можно оставить без изменений.

Теперь перейдем собственно к созданию виртуального хоста. В самом конце редактируемого файла до строки `</IfModule>` мы создадим секцию `VirtualHost`, в которую занесем необходимые параметры.

```
<VirtualHost localhost:443>
    ScriptAlias /cgi-bin/ "/var/www/SSL/cgi-bin/"
    ServerName localhost
    ServerAdmin adm@localhost
    DocumentRoot /var/www/SSL/html
    ErrorLog /var/log/httpd/SSL.err
    TransferLog /var/log/httpd/SSL.transf
    CustomLog /var/log/httpd/SSL.access combined

    SSLEngine On
    SSLCertificateFile /etc/httpd/server.cert
    SSLCertificateKeyFile /etc/httpd/nopass.key

<Files ~ "\.(cgi|shtml|phtml|php?)$" >
    SSLOptions +StdEnvVars
</Files>
```

```
<Directory "/var/www/SSL/cgi-bin">
    SSLOptions +StdEnvVars
    AllowOverride None
    Options ExecCGI
    Order allow,deny
    Allow from all
</Directory>

<Directory "/var/www/SSL/html">
    Options -Indexes -FollowSymLinks +MultiViews
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>
</VirtualHost>
```

Все вносимые нами изменения находятся внутри «скобок» `<IfModule>`. Это означает, что если в вашей реализации `apache` отсутствует SSL-модуль, `https` сервер не запустится (но это и не приведет к ошибкам). В этом случае установите пакет `apache-mod_ssl` (или, если вы устанавливали веб-сервер из исходников, перекомпилируйте его с поддержкой `mod_ssl`).

С настройкой `apache` мы закончили, но перезапускать его пока еще рано: у нас нет SSL-сертификатов. Их можно купить, а можно (тем более, если ваш сервер предназначен лишь для внутреннего использования) сделать самому. Итак, переходим к следующему шагу.

1.2 Создание SSL-сертификатов

Для создания самоподписанного сертификата нам потребуется пакет `openssl`. Во-первых, нам нужен закрытый ключ. Здесь возможны два варианта: создать ключ, защищенный паролем, либо ключ без пароля. Если вы не хотите вводить пароль всякий раз при запуске/перезапуске `apache`, вам подойдет ключ без пароля, иначе, если вам нужна повышенная безопасность, лучше сделать ключ с паролем. Чтобы создать ключ с паролем, выполните команду

```
openssl genrsa -des3 -out pass.key
```

Вам будет предложено ввести пароль: не забудьте его (особенно, если вы не будете его удалять). Удалить пароль можно командой

```
openssl rsa -in pass.key -out nopass.key
```

Учтите, что незащищенный ключ должен быть доступен для чтения **только** суперпользователю.

Далее нам необходимо создать файл запроса сертификата. Он нужен как для получения платного стороннего сертификата, так и своего, самоподписанного. Запрос генерируется командой

```
openssl req -new -key nopass.key -out server.csr
```

Вам придется ввести некоторую информацию (страна, область, город, название компании, название отдела, название хоста, почтовый адрес) и дополнительную (необязательную) информацию: пароль на запрос и дополнительное название компании. При вводе названия

хоста проследите, чтобы оно в точности соответствовало доменному имени компьютера, иначе будут выдаваться предупреждения о том, что сертификат выдан другому компьютеру. В дальнейшем этот файл предполагается использовать для получения платного стороннего сертификата. Нам же это не нужно, поэтому генерируем сертификат самостоятельно:

```
openssl x509 -req -days 365 -in server.csr -signkey nopass.key -out  
server.cert
```

Если вы хотите, чтобы сертификат прослужил не год, а другое время, измените значение параметра `days`.

Остается лишь разместить файлы сертификата и ключа в директориях, указанных директивами `SSLCertificateFile` и `SSLCertificateKeyFile`.

Теперь перезапускаем веб-сервер:

```
apachectl restart
```

Если все было сделано без ошибок, `apache` перезапустится, и вы получите возможность соединения со своим сервером по протоколу `https` с SSL-шифрованием.

1.3 Проверяем https-соединение

Для проверки работы `https` создайте в директории, указанной директивой `DocumentRoot` вашего виртуального `https`-сервера, индексный файл вида

```
<html><body>  
SSL  
</body></html>
```

И наберите в адресной строке браузера `https://localhost`. Если у вас возникнет предупреждение о том, что соединение является недоверенным, значит, все было выполнено верно. Согласитесь с тем, что понимаете риск. Добавьте исключение безопасности для своего сайта. Далее вы получите предупреждение о том, что сертификат является самоподписанным— но мы об этом и сами прекрасно знаем. Поставьте отметку «Постоянно хранить исключение» и подтвердите исключение безопасности.

Если настройки доступа к корневой директории вашего `https`-сервера были выполнены правильно, на экране вы увидите надпись «SSL». Значит, наш сервер настроен и работает верно. Чтобы убедиться, что наше соединение зашифровано, посмотрите свойства веб-страницы в своем браузере. Вы должны увидеть надпись «Зашифрованное соединение: стойкое шифрование...».

Нам остается лишь создать директорию для скриптов CGI нашего `https`-сервера (в данном случае это `/var/www/SSL/cgi-bin`).

1.4 Установка CGI и вспомогательных сервисов

Для того, чтобы наши сервисы поддерживали аутентификацию и авторизацию, необходимо добавить веб-страницу аутентификации, CGI для работы с базой данных пользователей и оперирующие с файлами cookie сценарии на JavaScript.

1.4.1 Компиляция

Распакуйте куда-нибудь содержимое архива HTTP[S]_authentication.tgz. Войдите в директорию HTTP[S]_authentication. В ней вы увидите несколько файлов на C и директорию html. Так как база данных должна быть доступна не только из-под защищенного соединения, в CGI выхода из системы путь к базе должен быть полным. Измените в начале файла exit.c строковую константу db_path на полный путь к базе относительно вашей файловой системе, т.е. если ваша директория cgi-bin для https-сервера /var/www/SSL/cgi-bin, то путь надо указать так:

```
static char *db_path = "/var/www/SSL/cgi-bin/user-pass";
```

Для компиляции CGI аутентификации и работы с базой данных запустите в текущей директории команду make:

```
make
gcc -Wall -D_XOPEN_SOURCE=501 -c -o auth.o auth.c
gcc -Wall -D_XOPEN_SOURCE=501 auth.o -lsqlite3 -lcrypt -o auth
gcc adduser.c -Wall -D_XOPEN_SOURCE=501 -lsqlite3 -lcrypt -o adduser
gcc exit.c -Wall -D_XOPEN_SOURCE=501 -lsqlite3 -lcrypt -o exit
gcc deluser.c -Wall -D_XOPEN_SOURCE=501 -lsqlite3 -lcrypt -o deluser
/bin/rm -f *.o *~
```

Для компиляции исходных текстов необходимо, чтобы в вашей системе были установлены библиотеки разработки sqlite3 и crypt (см. Makefile на стр. 17).

В директории с исходниками будут созданы исполняемые файлы, два из которых — exit и auth необходимо скопировать в директорию cgi-bin вашего https-сервера. Кроме того, CGI exit нужно скопировать в директорию cgi-bin вашего http-сервера, т.к. это — скрипт выхода, который может запускаться как из под защищенных, так и не защищенных страниц. Файлы adduser и deluser служат для добавления и удаления пользователей из базы данных. Их не нужно копировать вместе с CGI. Еще один файл — bash-скрипт INIT_DB служит для первичной инициализации базы данных.

1.4.2 Инициализация базы данных и работа с ней

Для инициализации базы данных запустите скрипт INIT_DB. В текущей директории будет создана база данных user-pass. Режим доступа к ней — 0666, что не очень хорошо с точки зрения безопасности, поэтому есть смысл от имени суперпользователя после операций с ней изменить ее владельца на apache (или www — т.е. пользователя, от имени которого запускается web-сервер).

По умолчанию в базу заносится один-единственный пользователь с логином sampleuser, паролем pass и уровнем доступа 3. Для добавления или удаления пользователя используйте утилиты adduser и deluser. В качестве первого аргумента обеих задается полный путь к базе данных.

Итак, для того, чтобы добавить пользователя, запустим утилиту adduser. Если запустить ее без аргументов, она будет искать базу данных в текущей директории. Создадим пользователя с логином adm, паролем \$am0u4ka и уровнем доступа 100:

```
Имя пользователя :
adm
Пароль :
```

```
$am0u4ka
Уровень доступа:
100
Проверим:
    имя пользователя - adm;
    пароль - $am0u4ka;
    доступ - 100.
Верно (y/n)?
y
```

Если все данные правильны, в ответ на запрос введите «y», и пользователь будет создан.

Так как пароль отображается открытым текстом, желательно проследить, чтобы никто его не подсмотрел.

Для удаления пользователя используйте утилиту `deluser`. Ее обязательно вызывать с двумя аргументами: первый является путем к базе данных, второй — именем удаляемого пользователя. Если вы запустите эту утилиту без аргументов, появится подсказка об использовании:

```
./deluser user-pass
Использование:
deluser <database> <user>

где      <user> - имя пользователя,
         <database> - полный путь к базе данных SQLite
```

Удалим пользователя по умолчанию:

```
./deluser user-pass sampleuser
Удаляю пользователя sampleuser
```

Все, пользователь удален из базы данных. Помимо этой утилиты вы можете удалять пользователей и вручную — при помощи `sqlite3`.

После того, как вы добавите нужное количество пользователей, перенесите базу данных в директорию с CGI-скриптами и, в случае необходимости, смените режим доступа на 0600 и владельца на `apache`.

1.4.3 Подключение и тестирование сервиса аутентификации

Теперь внесем некоторые изменения в файлы `pass.html` и `pass.js` и скопируем их в корень `https`-сервера, а также скопируем `pass.js` в корень `http`-сервера (это — основной сценарий авторизации, который вы будете подключать к своим `web`-страницам). Изменения касаются только названия вашего сервера.

В сценарии `pass.js` замените константы `PASSURL` и `EXURL` на расположение CGI аутентификации и CGI выхода из системы. Обратите внимание, что в сценарии, который вы поместите в корень `http`-сервера, путь к `EXURL` должен указывать на незащищенный `cgi-bin`.

В файле `pass.html` замените константу `CGI_PATH`, определяемую в самом начале ввездного сценария JavaScript, на адрес CGI аутентификации вашего сервера.

Далее скопируйте файлы по назначению.

Проверим, как работает наш сервис. Для этого в файл `index.html` корня нашего `https`-сервера внесем изменения:

```
<html>
<head>
<script src="pass.js" type="text/javascript" language="javascript"></
  script>
</head>
<body onload="getcookie();">
Доступ разрешен!
<br>
<button onclick="exit();">Выход</button>
</body>
</html>
```

Т.е. подключим сценарий авторизации, добавим надпись «Доступ разрешен», которая появится в случае успешной аутентификации, а также кнопку выхода.

Подключимся к своему серверу через https-протокол. Функция `getcookie` проверяет, есть ли у нас файл cookie с ключом авторизации. Если его нет, осуществляется перенаправление на страницу аутентификации. Вы увидите поля ввода имени пользователя и пароля. Если введенные вами данные будут верными, осуществится перенаправление на страницу, с которой вы только что ушли для аутентификации. Иначе — появится сообщение об ошибке.

Если все было выполнено верно, после ввода имени пользователя и пароля вы увидите надпись «Доступ разрешен» и кнопку «Выход». Проверим, как работает сохранение файлов cookie: перезапустите свой браузер и попробуйте вновь войти на свой сайт. Если настройки вашего браузера позволяют сохранять файлы cookie, вам не придется авторизоваться повторно. Для одной сессии cookie хранится в течение 20 часов. По истечении этого времени вам необходимо будет вновь пройти процедуру аутентификации.

Теперь нажмите на кнопку «Выход». Если CGI выхода работает верно, cookie и запись с ключем сессии в базе данных будут удалены, и вы вновь перенаправитесь на страницу аутентификации.

1.4.4 Подключение сервиса авторизации к своим web-сценариям

В директории с исходными кодами есть файл `CGI_auth.c`, который можно подключить к своим CGI для поддержки сервиса авторизации. Этот файл экспортирует лишь одну функцию `get_auth_level()`, возвращающую целое число, равное уровню доступа, либо завершающую работу CGI с одним из кодов ошибки. Если вы хотите иметь возможность регистрировать ошибки с текстовым комментарием, прокомментируйте в этом файле функцию `die(int error)` и подключите файл `auth.h`.

Для того, чтобы иметь возможность использовать функцию `get_auth_level()`, добавьте в заголовочный файл своего CGI строку

```
extern int get_auth_level();
```

в список исходных файлов своего `Makefile` добавьте `CGI_auth.c`, а в список подключаемых библиотек добавьте `-lsqlite3 -lcrypt`. Если же ваш CGI состоит из одного файла, например, `mycgi.c`, скомпилировать его с поддержкой авторизации можно так:

```
gcc -lsqlite3 -lcrypt mycgi.c CGI_auth.c -o mycgi
```

Проследите, чтобы константа `db_path`, инициализируемая в начале этого файла, содержала правильный полный путь к базе данных.

В `html`-файл, вызывающий этот сценарий, надо внести изменения, аналогично описанным в разделе 1.4.3. Не забудьте, что если ваш сервис работает по незащищенному протоколу, в корневую директорию этого сервера нужно скопировать файл `pass.js`, а в соответствующую директорию `cgi-bin` скопировать скрипт `exit`, как описано в том же разделе.

Для примера сделаем простую веб-страницу, при переходе на которую происходит перенаправление пользователя для аутентификации (в случае, если он еще не прошел эту процедуру, или истек срок действия его ключа). После прохождения аутентификации пользователь перенаправляется обратно, и страница запускает CGI, который отображает на странице надпись «Добро пожаловать, администратор», если уровень доступа пользователя имеет значение больше 100, надпись «Здравствуйте!», если уровень доступа лежит в пределах $21 \div 100$, и надпись «До свидания!» при уровне доступа 20 и ниже.

Так как наш CGI не должен анализировать поступающие к нему запросы, а лишь выдавать одну строку текста в ответ на свой вызов, его код будет достаточно простым:

```
#include <stdio.h>
extern int get_auth_level();
int main(){
    char *msg;
    int level = get_auth_level();
    if(level > 100) msg = "Добро пожаловать , администратор";
    else if(level > 20) msg = "Здравствуйте!";
    else msg = "До свидания!";
    printf("Content-type: text/html;\n\n%s\n", msg);
    return 0;
}
```

Сохраним этот файл как `msg.c` и скомпилируем:

```
gcc -lsqlite3 -lcrypt msg.c CGI_auth.c -o msg
```

Получившийся файл `msg` необходимо скопировать в директорию `cgi-bin` вашего незащищенного `web`-сервера.

Теперь в корневой директории сервера создадим `html`-файл с содержимым

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=koi8-r">
<title>test</title>
<script src="/pass.js" type="text/javascript" language="javascript"></script>
<script language="javascript">
function start(){
    var timeout_id;
    getcookie();
    var request = new XMLHttpRequest();
    request.open("POST", "/cgi-bin/msg", true);
    request.onreadystatechange=function(){
        if(request.readyState == 4){
            if(request.status == 200){
```

```

        clearTimeout(timeout_id);
        document.getElementById("message").
            innerHTML =
                request.responseText;
    }
    else alert("Ошибка передачи запроса. Попробуйте
        еще раз.");
    }
}
request.send(null);
timeout_id = setTimeout(function(){
    alert("Таймаут передачи запроса. Попробуйте еще раз.");},
    3000);
}
</script>
</head>
<body onload="start();">
<h1 align=center id="message"></h1>
<h3 onclick="exit();">для выхода щелкните по этой надписи</h3>
</body>
</html>

```

Добавим при помощи функции `adduser` пользователей `guest` с уровнем доступа 1, `user` с уровнем доступа 50 и `admin` с уровнем доступа 150. Как это сделать описано в разделе 1.4.2.

Теперь зайдите на страницу `msg` своего сервера. Вы должны будете перенаправиться на сервис аутентификации. Начните с пользователя `guest`. Если вы все сделали правильно, должна появиться надпись «До свидания» и ниже нее «для выхода щелкните по этой надписи», щелкнув по которой вы вновь должны оказаться на странице аутентификации. Зайдите под двумя остальными регистрационными именами. Сообщения на странице должны измениться.

А Исходные коды

А.1 CGI аутентификации, его заголовочный файл и CGI выхода из системы

auth.c

```

#define _XOPEN_SOURCE 501
#include <unistd.h>
#include <stdlib.h>
#include <strings.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <sqlite3.h>
#include "auth.h"

// #define __CONSOLE__ // для тестирования в консоли

```



```

char *qs = NULL;
char* get_qs(){
    char *m, *tmpbuf;
    if((m = getenv("REQUEST_METHOD")) && strcasecmp(m, "POST") == 0){
        tmpbuf = (char*)calloc(2048, 1);
        fgets(tmpbuf, 2048, stdin);
        qs = strdup(tmpbuf);
        free(tmpbuf);
    }
    else
#ifdef __CONSOLE__
        if( (tmpbuf = getenv("QUERY_STRING")) )
            qs = strdup(tmpbuf);
#else
        die(NO_QS);
#endif
    if(qs && strlen(qs) < 1)
        die(NO_QS);
    return qs;
}

char* get_qs_param(char *param){
    char *tok, *val, *par, *str, *ret = NULL;
    if(!qs) return NULL;
    str = strdup(qs);
    tok = strtok(str, "& \n");
    do{
        if((val = strchr(tok, '=')) == NULL) continue;
        *val++ = '\0';
        par = tok;
        if(strcasecmp(par, param)==0){
            if( strlen(val) > 0 )
                ret = strdup(val);
            break;
        }
    }while((tok = strtok(NULL, "& \n"))!=NULL);
    free(str);
    return ret;
}

char *SQLbuf = NULL;
int callback(void *NotUsed, int argc, char **argv, char **azColName){
    if(!*argv) return 1;
    SQLbuf = strdup(*argv);
    return 0;
}

char *SQLexec(char *q_str){
    sqlite3 *db;

```

```

    char *zErrMsg = NULL, *ret = NULL;
    int rc = sqlite3_open("user-pass", &db);
    if(rc){ sqlite3_close(db); die(NO_DB);}
    if(SQLbuf){ free(SQLbuf); SQLbuf = NULL;}
    rc = sqlite3_exec(db, q_str, callback, NULL, &zErrMsg);
    if(rc){
        fprintf(stderr, "SQL error: %s\n", zErrMsg);
        //die(SQL_ERR);
    }
    else if(SQLbuf)
        ret = strdup(SQLbuf);
    sqlite3_close(db);
    return ret;
}

char *get_passwd(char *login, int *level){
    char *q_str = (char*)calloc(512, 1), *ret=NULL, *tmp;
    snprintf(q_str, 512, "select pass from users where login=\"%s\";"
        , login);
    ret = SQLEXec(q_str);
    snprintf(q_str, 512, "select level from users where login=\"%s\";"
        , login);
    if( !(tmp = SQLEXec(q_str)) ) *level = 0;
    else *level = atoi(tmp);
    free(q_str);
    free(tmp);
    return ret;
}

void insert2db(char *id, char *key, int level){
    char *q_str = (char*)calloc(512, 1);
    snprintf(q_str, 512, "delete from keyid where time < julianday('"
        now')-1;");
    SQLEXec(q_str);
    snprintf(q_str, 512, "insert into keyid values(\"%s\", \"%s\", %d,"
        julianday('now')));", id, key, level);
    SQLEXec(q_str);
}

char *sha(char *str, char *salt){
    char *tmp = crypt(str, salt);
    return strdup(tmp);
}

char *getpath(char *url){
    char *ptr, *ptr1;
    ptr = strchr(url, '/');
    if(!ptr) return NULL;
    ptr1 = strchr(ptr+1, '/');
    if(!ptr1) return NULL;
}

```

```

    ptr = strchr(ptr1+1, '/');
    if(!ptr){
        ptr = (char*)calloc(2,1);
        *ptr = '/';
    }
    return ptr;
}

int main(int argc, char **argv){
    char *IP, *URL, *id, *login, *pass_raw, *pass_ori, *pass, *key, *
        salt;
    int level;
    printf("Content-type: text/html\n\n");
    salt = calloc(128, 1);
    IP = getenv("REMOTE_ADDR");
    if(!IP) die(BAD_IP);
    if( !get_qs() ) die(NO_QS);
    if(!(URL = get_qs_param("URL"))) die(BAD_URL);
    if(!(login = get_qs_param("login"))) die(NO_LOGIN);
    if(!(pass_raw = get_qs_param("passwd"))) die(NO_PASSWD);
    pass = sha(pass_raw, "$6$pass-enc$");
    free(pass_raw);
    if( !(pass_ori = get_passwd(login, &level)) ) die(WRONG_PASSWD);
    if(strcmp(pass_ori, pass)) die(WRONG_PASSWD);
    srand(time(NULL));
    snprintf(salt, 128, "$5%s$", IP);
    id = sha(URL, salt);
    id = strchr(id, '$') + 1;
    snprintf(salt, 128, "$5%.%d.$", rand());
    key = sha(id, salt);
    key = strchr(key, '$') + 1;
    free(salt);
    insert2db(id, key, level);
    printf("KEY=%s; path=%s\n", key, getpath(URL));
    return(0);
}

```

auth.h

```

#define NO_QS 0
#define NO_LOGIN -1
#define NO_PASSWD -2
#define WRONG_PASSWD -3
#define BAD_URL -4
#define BAD_IP -5
#define NO_DB -6
#define NO_REG -7
#define BAD_SERV -8
#define SQL_ERR -9

```

```

char    *no_qs = "Отсутствует строка запроса",
        *no_login = "Не указано имя пользователя",
        *no_passwd = "Не указан пароль",
        *wrong_passwd = "Неверная пара имя-пароль",
        *bad_url = "Отсутствует наименование сервиса для
                    регистрации",
        *bad_ip = "Не могу определить ваш IP-адрес",
        *no_db = "Повреждена база данных пользователей",
        *no_reg = "Вы не авторизованы",
        *bad_serv = "Вы не авторизованы на этом сервисе",
        *sql_err = "Ошибка базы данных",
        *unknown = "Неизвестная ошибка";

static void die(int error){
    char *msg;
    switch(error){
        case NO_QS: msg = no_qs; break;
        case NO_LOGIN: msg = no_login; break;
        case NO_PASSWD: msg = no_passwd; break;
        case WRONG_PASSWD: msg = wrong_passwd; break;
        case BAD_URL: msg = bad_url; break;
        case BAD_IP: msg = bad_ip; break;
        case NO_DB: msg = no_db; break;
        case NO_REG: msg = no_reg; break;
        case BAD_SERV: msg = bad_serv; break;
        case SQL_ERR: msg = sql_err; break;
        default: msg = unknown; break;
    }
    printf("%s\n", msg);
    fprintf(stderr, "Error %d: %s\n", error, msg);
    exit(error);
}

```

exit.c

```

#define _XOPEN_SOURCE 501
#include <unistd.h>
#include <stdlib.h>
#include <strings.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <sqlite3.h>
#include "auth.h"
// в приложении, которое должно работать с аутентификацией, добавить
// extern int get_auth_level();

static char *db_path = "/var/www/SSL/cgi-bin/user-pass";

char* get_key(){

```

```

    char *tok, *val, *par, *ret = NULL, *cookie;
    cookie = getenv("HTTP_COOKIE");
    if(!cookie) return NULL;
    tok = strtok(cookie, "; \n");
    do{
        if((val = strchr(tok, '=')) == NULL) continue;
        *val++ = '\0';
        par = tok;
        if(strcasecmp(par, "key")==0){
            if( strlen(val) > 0 )
                ret = strdup(val);
            break;
        }
    }while((tok = strtok(NULL, "; \n"))!=NULL);
    return ret;
}

char *SQLbuf = NULL;
int callback(void *NotUsed, int argc, char **argv, char **azColName){
    if(!*argv) return 1;
    SQLbuf = strdup(*argv);
    return 0;
}

char *SQLexec(char *q_str){
    sqlite3 *db;
    char *zErrMsg = NULL, *ret = NULL;
    int rc = sqlite3_open(db_path, &db);
    if(rc){ sqlite3_close(db); die(NO_DB);}
    if(SQLbuf){ free(SQLbuf); SQLbuf = NULL;}
    fprintf(stderr, "SQL: %s;\n", q_str);
    rc = sqlite3_exec(db, q_str, callback, NULL, &zErrMsg);
    if(rc){
        fprintf(stderr, "SQL error: %s\n", zErrMsg);
        die(SQL_ERR);
    }
    else if(SQLbuf)
        ret = strdup(SQLbuf);
    sqlite3_close(db);
    return ret;
}

char *get_id(char *key, int *level){
    char *ret=NULL, *tmp;
    char *q_str = (char*)calloc(512, 1);
    snprintf(q_str, 512, "select id from keyid where key=\"%s\"", key
    );
    ret = SQLexec(q_str);
    snprintf(q_str, 512, "select level from keyid where key=\"%s\"",
    key);

```

```

        if( !(tmp = SQLexec(q_str)) ) *level = 0;
        else *level = atoi(tmp);
        free(q_str);
        free(tmp);
        return ret;
}

char *sha(char *str, char *salt){
    char *tmp = crypt(str, salt);
    return strdup(tmp);
}

int main(int argc, char **argv){
    char *IP, *URL, *id, *id_ori, *key, *salt, *q_str;
    int level;
    printf("Content-type: text/html\n\n");
    salt = calloc(128, 1);
    IP = getenv("REMOTE_ADDR");
    if(!IP) die(BAD_IP);
    URL = getenv("HTTP_REFERER");
    fprintf(stderr, "URL=%s\n", URL);
    if(!URL) die(BAD_URL);
    key = get_key();
    if(!key) die(BAD_SERV);
    snprintf(salt, 128, "$$%s$", IP);
    id = sha(URL, salt);
    free(salt);
    id = strchr(id, '$') + 1;
    id_ori = get_id(key, &level);
    if(!id_ori) die(NO_REG);
    if(strcmp(id, id_ori)) die(BAD_SERV);
    q_str = (char*)calloc(512, 1);
    snprintf(q_str, 512, "delete from keyid where key=\"%s\";", key);
    SQLexec(q_str);
    free(q_str);
    return(0);
}

```

А.2 Утилиты для редактирования базы данных

adduser.c

```

#define _XOPEN_SOURCE 501
#include <unistd.h>
#include <stdlib.h>
#include <strings.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <sqlite3.h>

```

```
char *sqlbase = "user-pass";
char *SQLbuf = NULL;
int callback(void *NotUsed, int argc, char **argv, char **azColName){
    if(!*argv) return 1;
    SQLbuf = strdup(*argv);
    return 0;
}

char *SQLexec(char *q_str){
    sqlite3 *db;
    char *zErrMsg = NULL, *ret = NULL;
    int rc = sqlite3_open(sqlbase, &db);
    if(rc){ sqlite3_close(db); printf("Нет БД\n"); exit(1);}
    if(SQLbuf) free(SQLbuf);
    rc = sqlite3_exec(db, q_str, callback, NULL, &zErrMsg);
    if(rc)
        fprintf(stderr, "SQL error: %s\n", zErrMsg);
    else if(SQLbuf)
        ret = strdup(SQLbuf);
    sqlite3_close(db);
    return ret;
}

void insert2db(char *user, char *pass, int level){
    char *q_str = (char*)calloc(512, 1);
    snprintf(q_str, 512, "insert into users values(\"%s\", \"%s\", %d);",
        user, pass, level);
    SQLexec(q_str);
}

char *sha(char *str, char *salt){
    char *tmp = crypt(str, salt);
    return strdup(tmp);
}

char *getl(char *buf, int n){
    fgets(buf, n, stdin);
    buf[strlen(buf)-1] = 0;
    return buf;
}

int main(int argc, char **argv){
    char *login, *pass_raw, *pass, *tmp, ch;
    int level;
    login = calloc(128, 1);
    pass_raw = calloc(32, 1);
    tmp = calloc(32, 1);
    if(argc > 1) sqlbase = strdup(argv[1]);
    do{
```

```

        printf("Имя пользователя:\n");
        getl(login, 128);
        printf("Пароль:\n");
        getl(pass_raw, 32);
        printf("Уровень доступа:\n");
        getl(tmp, 32);
        level = atoi(tmp);
        free(tmp);
        printf("\nПроверим:\n\тима пользователя - %s;\n\тпароль -
                %s;\n\тдоступ - %d.\nВерно (y/n)?\n",
                login, pass_raw, level);
        ch = getchar();
    } while(ch != 'y');
    pass = sha(pass_raw, "$6$pass-enc$");
    insert2db(login, pass, level);
    return(1);
}

```

deluser.c

```

#define _XOPEN_SOURCE 501
#include <unistd.h>
#include <stdlib.h>
#include <strings.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <sqlite3.h>

char *sqlbase;
char *SQLbuf = NULL;
int callback(void *NotUsed, int argc, char **argv, char **azColName){
    if(!*argv) return 1;
    SQLbuf = strdup(*argv);
    return 0;
}

char *SQLEXec(char *q_str){
    sqlite3 *db;
    char *zErrMsg = NULL, *ret = NULL;
    int rc = sqlite3_open(sqlbase, &db);
    if(rc){ sqlite3_close(db); printf("Нет БД\n"); exit(1);}
    if(SQLbuf) free(SQLbuf);
    rc = sqlite3_exec(db, q_str, callback, NULL, &zErrMsg);
    if(rc)
        fprintf(stderr, "SQL error: %s\n", zErrMsg);
    else if(SQLbuf)
        ret = strdup(SQLbuf);
    sqlite3_close(db);
    return ret;
}

```



```

}

void rm_from_db(char *user){
    char *q_str = (char*)calloc(512, 1);
    snprintf(q_str, 512, "delete from users where login=\"%s\";",
             user);
    SQLexec(q_str);
}

void help(){
    printf("Использование:\ndeluser <database> <user>\n\nгде\t<user>
- имя пользователя,\n\t<database> - полный путь к базе данных
SQLite\n\n");
    exit(0);
}

int main(int argc, char **argv){
    char *login;
    if(argc != 3) help();
    sqlbase = strdup(argv[1]);
    login = strdup(argv[2]);
    printf("Удаляю пользователя %s\n", login);
    rm_from_db(login);
    return(1);
}

```

A.3 Makefile

```

PROGRAM = auth
LOADLIBES = -lsqlite3 -lcrypt
CXX.SRCS = auth.c
CC = gcc
DEFINES = -D_XOPEN_SOURCE=501
CXX = gcc
CPPFLAGS = -Wall $(DEFINES)
OBJS = $(CXX.SRCS:.c=.o)
all : $(PROGRAM) clean
$(PROGRAM) : $(OBJS)
    $(CC) $(CPPFLAGS) $(OBJS) $(LOADLIBES) -o $(PROGRAM)
    $(CC) adduser.c $(CPPFLAGS) $(LOADLIBES) -o adduser
    $(CC) exit.c $(CPPFLAGS) $(LOADLIBES) -o exit
    $(CC) deluser.c $(CPPFLAGS) $(LOADLIBES) -o deluser
clean:
    /bin/rm -f *.o *~
depend:
    $(CXX) -MM $(CXX.SRCS)

### <DEPENDENCIES ON .h FILES GO HERE>
# name1.o : header1.h header2.h ...

```

А.4 Скрипт для инициализации базы данных

INIT_DB

```
#!/bin/bash
# пароль - pass
rm -f user-pass
cat > .tmp_ << EOF
create table users(login TEXT, pass TEXT, level INTEGER);
insert into users values("ed", "\$6\$pass-enc\$I1.wNxb1TAttpehC22XRfh1LQ/
K.dkJEhORKjoluWk4zJgTm8a/EdrUKw4wU43S.ROTEj3V54WJIPfE5cOWyM/", 3);
create table keyid(id TEXT, key TEXT, level INTEGER, time REAL);
.quit
EOF
sqlite3 user-pass < .tmp_
rm -f .tmp_
```

А.5 Подключаемый файл для контроля авторизации

CGI_auth.c

```
#define _XOPEN_SOURCE 501
#include <unistd.h>
#include <stdlib.h>
#include <strings.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <sqlite3.h>
// в приложении, которое должно работать с аутентификацией, добавить
// extern int get_auth_level();

#define NO_QS 0
#define NO_LOGIN -1
#define NO_PASSWD -2
#define WRONG_PASSWD -3
#define BAD_URL -4
#define BAD_IP -5
#define NO_DB -6
#define NO_REG -7
#define BAD_SERV -8

static void die(int error){
    fprintf(stderr, "Auth error %d\n", error);
    exit(error);
}
```

```

static char *db_path = "/var/www/SSL/cgi-bin/user-pass";

static char* get_key(){
    char *tok, *val, *par, *ret = NULL, *cookie;
    cookie = getenv("HTTP_COOKIE");
    if(!cookie) return NULL;
    tok = strtok(cookie, "; \n");
    do{
        if((val = strchr(tok, '=')) == NULL) continue;
        *val++ = '\0';
        par = tok;
        if(strcasecmp(par, "key")==0){
            if( strlen(val) > 0 )
                ret = strdup(val);
            break;
        }
    }while((tok = strtok(NULL, "; \n"))!=NULL);
    return ret;
}

static char *SQLbuf = NULL;
int callback(void *NotUsed, int argc, char **argv, char **azColName){
    if(!*argv) return 1;
    SQLbuf = strdup(*argv);
    return 0;
}

static char *SQLEXec(char *par, char *key){
    sqlite3 *db;
    char *zErrMsg = NULL, *ret = NULL;
    char *q_str = (char*)calloc(512, 1);
    int rc = sqlite3_open(db_path, &db);
    if(rc){ sqlite3_close(db); die(NO_DB);}
    if(SQLbuf){ free(SQLbuf); SQLbuf = NULL;}
    snprintf(q_str, 512, "select %s from keyid where key=\"%s\"", par,
        key);
    rc = sqlite3_exec(db, q_str, callback, NULL, &zErrMsg);
    if(rc)
        fprintf(stderr, "SQL error: %s\n", zErrMsg);
    else if(SQLbuf)
        ret = strdup(SQLbuf);
    free(q_str);
    sqlite3_close(db);
    return ret;
}

static char *get_id(char *key, int *level){
    char *ret=NULL, *tmp;
    ret = SQLEXec("id", key);
    if( !(tmp = SQLEXec("level", key)) ) *level = 0;
}

```

```
        else *level = atoi(tmp);
        free(tmp);
        return ret;
    }

static char *sha(char *str, char *salt){
    char *tmp = crypt(str, salt);
    return strdup(tmp);
}

int get_auth_level(){
    char *IP, *URL, *id, *id_ori, *key, *salt;
    int level;
    salt = calloc(128, 1);
    IP = getenv("REMOTE_ADDR");
    if(!IP) die(BAD_IP);
    URL = getenv("HTTP_REFERER");
    if(!URL) die(BAD_URL);
    key = get_key();
    if(!key) die(BAD_SERV);
    snprintf(salt, 128, "$5$s$", IP);
    id = sha(URL, salt);
    id = strchr(id, '$') + 1;
    id_ori = get_id(key, &level);
    if(!id_ori) die(NO_REG);
    if(strcmp(id, id_ori)) die(BAD_SERV);
    free(salt);
    return(level);
}
```