Implementing Improved User Support for Streaming Audio

By

Amy Beam

Submitted to
the Faculty of the Information Engineering Technology Program
in Partial Fulfillment of the Requirements for
the Degree of Bachelor of Science
in Computer Science Technology

University of Cincinnati
College of Applied Science

June 2004
Implementing Improved User Support for Streaming Audio

by

Amy Beam

Submitted to
the Faculty of the Information Engineering Technology Program
in Partial Fulfillment of the Requirements
for
the Degree of Bachelor of Science
in Computer Science Technology

© Copyright 2004 Amy Beam

The author grants to the Information Engineering Technology Program permission
to reproduce and distribute copies of this document in whole or in part.

___________________________________________________  __________________
Amy Beam                                                Date

___________________________________________________  __________________
Tom Wulf, Faculty Advisor                                 Date

___________________________________________________  __________________
James F. Sullivan, Department Head                        Date
Acknowledgements

I would like to acknowledge WOXY.com for giving me the opportunity to complete this project. Bryan Jay Miller, information technology director, and the owners of WOXY.com gave me the cooperation I needed for researching the project, the resources required, and gave me the freedom to complete the project as I saw necessary.

I would like to give thanks to my advisor, Professor Tom Wulf, for supporting me with this project. Professor Wulf was always able to find time for me when I needed last minute help from him. Professor Mark Stockman was also a huge help during the final stages of my presentation preparation. Without Professor Stockman’s help I would have not been able to give a final presentation that demonstrated my project completion as well as I did. I would also like to thank the entire faculty of the Math Physics and Computer Technology program for giving me the knowledge needed to complete this project.

Finally, I need to thank my family and friends who have been so patient with me over the last year. I need to specifically thank Kyle Miller because without his constant support I would have experienced multiple breakdowns each day. To everyone who kept telling me I could do it and I did not believe you, thank you!
Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgements</td>
<td>iii</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>iv</td>
</tr>
<tr>
<td>List of Figures</td>
<td>v</td>
</tr>
<tr>
<td>Abstract</td>
<td>vi</td>
</tr>
<tr>
<td>1.0 Statement of the Problem</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Definition of the Need</td>
<td>1</td>
</tr>
<tr>
<td>2.0 Description of the solution</td>
<td>2</td>
</tr>
<tr>
<td>2.1 Streaming Media Defined</td>
<td>2</td>
</tr>
<tr>
<td>2.2 Chosen Solution</td>
<td>2</td>
</tr>
<tr>
<td>2.3 Icecast Description</td>
<td>3</td>
</tr>
<tr>
<td>2.4 User Profile</td>
<td>3</td>
</tr>
<tr>
<td>2.5 Organizational Scheme</td>
<td>3</td>
</tr>
<tr>
<td>2.6 Authentication Modification</td>
<td>6</td>
</tr>
<tr>
<td>2.7 Web Interface</td>
<td>7</td>
</tr>
<tr>
<td>2.8 Modified Source Code</td>
<td>10</td>
</tr>
<tr>
<td>2.8.1 File Changes in icecast.xml</td>
<td>10</td>
</tr>
<tr>
<td>2.8.2 File Changes in auth.c</td>
<td>11</td>
</tr>
<tr>
<td>2.8.3 File Changes in auth.h</td>
<td>12</td>
</tr>
<tr>
<td>2.8.4 File Changes in client.c</td>
<td>12</td>
</tr>
<tr>
<td>2.8.5 File Changes in client.h</td>
<td>13</td>
</tr>
<tr>
<td>2.8.6 File Changes in connection.c</td>
<td>13</td>
</tr>
<tr>
<td>3.0 Objectives</td>
<td>14</td>
</tr>
<tr>
<td>4.0 Design and Development</td>
<td>15</td>
</tr>
<tr>
<td>4.1 Budget</td>
<td>15</td>
</tr>
<tr>
<td>4.2 Timeline</td>
<td>16</td>
</tr>
<tr>
<td>4.3 Software</td>
<td>16</td>
</tr>
<tr>
<td>4.4 Hardware</td>
<td>17</td>
</tr>
<tr>
<td>5.0 Proof of Design</td>
<td>17</td>
</tr>
<tr>
<td>5.1 Testing Procedures</td>
<td>17</td>
</tr>
<tr>
<td>6.0 Conclusions and Recommendations</td>
<td>18</td>
</tr>
<tr>
<td>6.1 Conclusions</td>
<td>18</td>
</tr>
<tr>
<td>6.2 Recommendations</td>
<td>18</td>
</tr>
<tr>
<td>Appendix A: Code Snippets</td>
<td>19</td>
</tr>
<tr>
<td>1.0 Logon PHP File</td>
<td>19</td>
</tr>
<tr>
<td>2.0 Play PHP File</td>
<td>20</td>
</tr>
<tr>
<td>3.0. icecast.XML modification</td>
<td>20</td>
</tr>
<tr>
<td>4.0 auth.c modification</td>
<td>20</td>
</tr>
<tr>
<td>5.0 auth.h modification</td>
<td>24</td>
</tr>
<tr>
<td>6.0 client.c modification</td>
<td>24</td>
</tr>
<tr>
<td>7.0 client.h modification</td>
<td>24</td>
</tr>
<tr>
<td>8.0 connection.c modification</td>
<td>24</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure Number</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1. Main Organization Scheme for Application</td>
<td>4</td>
</tr>
<tr>
<td>Figure 2. Dataflow within Server</td>
<td>5</td>
</tr>
<tr>
<td>Figure 3. Dataflow upon Disconnection</td>
<td>6</td>
</tr>
<tr>
<td>Figure 4. Log in to Administration Interface</td>
<td>8</td>
</tr>
<tr>
<td>Figure 5. Manage Authentication Link</td>
<td>9</td>
</tr>
<tr>
<td>Figure 6. Show Defined Users</td>
<td>10</td>
</tr>
<tr>
<td>Figure 7. Icecast.xml Example Authentication Section</td>
<td>11</td>
</tr>
</tbody>
</table>
Abstract

Implementing Improved User Support for Streaming Audio is focused on taking the current Icecast server and adding the ability to have users login to the server to receive an audio stream. This project was designed to fill a need for this type of server at a local radio station, but many small radio stations have a need to stream audio content on a subscription based model.

Icecast provides access to listeners who would normally fall outside a radio station’s broadcasting radius. In this project, the original Icecast source code has been modified to give the program the ability to allow each user to have a unique username and password assuring that no more than one person can listen using a username/password pair.
Implementing Improved User Support for Streaming Audio

1.0 Statement of the Problem

1.1 Definition of the Need

The local independent radio station WOXY currently has a presence on the World Wide Web at http://www.woxy.com. From this Web site users have been able to listen to the live feed from the radio through streaming audio. At the time this project was conceived users were only able to listen to this streaming audio at 56 Kbs maximum. For many casual listeners or people with slower Internet connections this connection speed was fine. However, there was a group of users that asked the radio station to increase the streaming audio to 128 Kbs because they had a high-speed Internet connection or they listen to the radio station for many hours at a time and desired to have a better quality feed. This request could not be met because the Internet service provider (ISP) subscription that provides WOXY with its current Internet capabilities could not handle it. The station has considered changing ISP subscriptions and giving the listeners what they want, but financially this has not been possible. At the time this project was completed WOXY had sold the rights to its terrestrial broadcast and had contemplated going to an Internet only forum. However, due to funding issues that move was unable to be made and as of May 15, 2004 WOXY had ceased broadcasting all together. Even with this decision the project continued so as to provide other small broadcasters with the tools to provide subscription based Internet radio streams through an open source server.
2.0 Description of the solution

2.1 Streaming Media Defined

Streaming media is the term used to describe the real-time delivery of moving images, moving text and sound, over the internet. Streaming software allows internet audiences to listen or watch types of media, which have, up until now, been considered too large and bulky for consumption over the Internet.

Streaming media techniques work in the following way: as you listen or watch one portion of content, the next portion is downloading at the same time. The ability to simultaneously load and play distinguishes streaming from other types of internet media. Streaming allows for live transmission over the Internet, which enables a transformation of the internet into broadcasting medium.

To stream files live over the internet the files first have to be encoded into an appropriate streaming format. This is done with the various encoding tools. In simple terms an audio signal, for example a microphone plugged into a computer, is converted by the encoding software into a continuous data ‘stream’ over the internet. This conversion involves compressing the data, which will reduce its quality and file size, and converting the data into the technology providers usually propriety streaming format. Codec’s are the algorithms that encode and decode the audio or visual signals for delivery over the internet, and are the core technology of streaming media.

2.2 Chosen Solution

I utilized my knowledge in application and Web development to implement an Icecast II server for WOXY and published the code under the GNU General Public
License (GPL). I modified the original source code of the icecast program to allow secure client authentication. This modification gave the program the ability to allow each user to have a unique username and password assuring that no more than one person can listen using a username/password pair.

2.3 Icecast Description

Icecast is an audio broadcasting system that streams music in both MP3 and Ogg Vorbis format. It is available under the terms of the GNU GPL. The main home page for the Icecast Project is located at http://www.icecast.org/. Source code for the icecast program can be downloaded from off this site. Please note that when referring to the Project itself, Icecast is capitalized. The Project includes a variety of other programs. If you are talking about the icecast program only, then the name is not capitalized.

The icecast program also requires libraries and a streaming application to which listeners connect and then "stream" the audio from off icecast server to their home machines. All of these programs together comprise the Icecast Project, though many of them fall outside the realm of the developer’s control.

2.4 User Profile

The audience for this project is all listeners of small radio stations that might implement this Icecast modification. These users vary in technology expertise from novice to expert. The current interface will be the model for the new interface so that users will not need to learn to use the new service.

2.5 Organizational Scheme

The flow of the application has seven main sections as seen in Figure 1.
Figure 1. Main Organization Scheme for Application

The first element in the flowchart is the Web Page Login. This is where the user logs to his or her account. Next, the login information is sent to the authentication module which verifies if the username and password pair is valid. The client authentication uses the basic authentication as defined by the HTTP specification. Authentication is defined on a per-mountpoint basis. The administrator needs to specify the users that have access to the mountpoint, and then specify the passwords for the member. Configuration is done via a web administration interface. If this information is valid then the user can access the rest of the Web page. When the user chooses, the user is able to decide to listen to the streaming audio with high bandwidth. Since the user is already logged in there is no need
to check and see if they are able to access the stream. At this point a PHP function will serve the client a link to the audio feed. Once the user clicks on the served link the client’s audio player of choice will start and request the stream from the Icecast server.

![Dataflow diagram](image)

**Figure 2. Dataflow within Server**

Once the server has received the stream request it checks the database to make sure that this user is not currently logged in. If the user is not currently logged in the server setups the thread to serve to the client. Once the thread is created the server will set a flag in the database to mark this user as logged in. The flag will be able to see which user has just logged in by taking the user name from the randomly generated link which was passed to the client from the PHP script.
When Icecast determines that a connection has been broken it receives the disconnection and retrieve the URL with the user’s name embedded within it. With the user’s name the flag in the database can be removed allowing the user to log in again and repeat the process. Thus the user can only log in one time with any given set of credentials and cannot share their account information with others for concurrent streaming feeds.

2.6 Authentication Modification

After this project’s modification Icecast 2.0 supports a file-based authentication scheme. To support listener authentication an administrator must now provide at a minimum <mount-name> and <authentication> values in the icecast.xml file. The mount-name is the name of the mountpoint that you will use to connect your source client with and authentication configures what type of Icecast 2.0 authenticator to use. Currently, only a single type "htpasswd" is implemented. Each authenticator has a variable number
of options that are required and these are specified as shown in the example in Figure 7. The htpasswd authenticator requires only a single parameter, filename, which specifies the name of the file to use to store users and passwords. All requests for this mountpoint will require that a user and password be supplied for authentication purposes. These values are passed in via normal HTTP Basic Authentication meaning through a Web address like: http://user:password@stream:port/mountpoint.ogg. Users and Passwords are maintained through the Web administration interface.

2.7 Web Interface

Access to the Web administration interface can be found on the same server as the Icecast project with the “/admin” added to the end of the Icecast URL. Then the administrator will be asked to log in as seen in Figure 4.
Figure 4. Log in to Administration Interface

After the administrator is authenticated then he/she needs to click on the manage authentication link as highlighted in red in Figure 5.
Once the administrator clicks on the “Manage Authentication” link then he/she will be taken to the “Show Defined Users” screen where users can be added or deleted. (See Figure 6.) To add a user to the mountpoint then the administrator needs to fill in the user ID and the password in the appropriate boxes and press the “Add New User” button. From this screen administrators are also able to Show the Listeners, Move Listeners, and Kill the source.
2.8 Modified Source Code

The modified source code can be viewed in detail in Appendix A at the end of the report. In total after the modification the Icecast source folder contained 634 files totaling up to about five megabytes. All my modifications were done in the c source code files and the corresponding header files. In total the .c and .h files were 329Kb and contained 46 files. The following modules were modified to allow for user authentication:

icecast.xml, auth.h, auth.c, client.h, client.c.

2.8.1 File Changes in icecast.xml

The htpasswd authenticator requires a few parameters. The first, filename, specifies the name of the file to use to store users and passwords. Note that this file need not exist (and probably will not exist when you first set it up). Icecast has built-in support
for managing users and passwords via the web admin interface. The second option, allow_duplicate_users, if set to 0, will prevent multiple connections using the same username. Setting this value to 1 will enable multiple connections from the same username on a given mountpoint.

![Image](image.png)

**Figure 7. Icecast.xml Example Authentication Section**

### 2.8.2 File Changes in auth.c

The first module added was the auth_is_listener_connected module. This module is passed the stream and the username of the user desiring to listen to the stream. Then this module checks to see if the username is currently connected to the stream. If the user is connected then the module will return an error code, otherwise the client_node variable receives the next available node on the mountpoint.

The next module only needed a small tweaking to make sure that the user was authenticated correctly. The module is called auth_result auth_check_client and the line that ensures the correct authentication is:

```c
result = authenticator->authenticate(authenticator, source, username, password);
```

The next change occurred in the structure definition. Within this definition the allow_duplicate_users variable needed to be added.

The next major change within auth.c was to the htpasswd_auth module. This module needed to have the username and password passed to it and it then could return
the authentication result through the auth_result variable. The following lines of code were changed:

```c
if (!state->allow_duplicate_users) {
    if (auth_is_listener_connected(source, username)) {
        thread_rwlock_unlock(&state->file_rwlock);
        return AUTH_FORBIDDEN;
    }
}
```

These lines of code allow the program to send an error if a user tries to log in while the username is locked.

The final change made in auth.c was to the auth_get_hpasswd_auth module. This change checks to make see if the allow_duplicate_users variable is on or off. This is done in the following lines of code:

```c
state->allow_duplicate_users = 1;
while(options) {
    if(!strcmp(options->name, "filename"))
        state->filename = strdup(options->value);
    if(!strcmp(options->name, "allow_duplicate_users"))
        state->allow_duplicate_users = atoi(options->value);
    options = options->next;
}
```

2.8.3 File Changes in auth.h

The only line changed in the auth header file is the following: “source_t *source, char *username, char *password.” With this line I was able to specify where the username/password file was located and what username and password were entered for the user.

2.8.4 File Changes in client.c
The changes to the file client.c are to take care of the 403 error that is sent to the client when the username is already logged in to the server. This error is the error that auth.c sends when it returns the AUTH_FORBIDDEN variable to the server.

2.8.5 File Changes in client.h

The only change made in client header file is to add the client_send_403 module.

2.8.6 File Changes in connection.c

The first change made to connection.c was to add in the auth header file. Next I added the ability to move clients from one mountpoint to another in the connection_initialize. Then in the connection_shutdown module I destroyed the variable I just created.

The connection_complete_source module is completely new to connection.c. This module is called when activating a source. It verifies that the source count is not exceeded and applies any initial parameters. Then it creates the client late in the module because Icecast can not use this specific client to return any error codes. So the client is only created once Icecast knows that it is going to accept that source.

The _check_pass_http module does exactly what the module name says, it checks the password when sent through to http authentication. Since the client authentication is new to Icecast so is this module. Some code was borrowed from administrator’s authentication check that was already built into Icecast.

The module connection_check_source_pass was modified to call the _check_pass_http module to check the authentication for the client. This module also
contains other authentication checks not used in this project, but left in the source for other users to that may have it implemented.

3.0 Objectives

- Modify the original Icecast II Server to include user authentication
- Assure that no more than one person can listen using a username/password pair.
- User only open source server technology that is licensed under the terms of the GNU General Public License (GPL)
- Include ability to stream multiple streams via HTTP on port 80
- Support for multiple, cross-platform streaming clients
- Support the Winamp, iTunes and XMMS client applications
### 4.0 Design and Development

#### 4.1 Budget

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Cost</th>
<th>URL</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sound card</td>
<td>$79.88</td>
<td><a href="http://www.amazon.com">http://www.amazon.com</a></td>
<td>Creative Labs Sound Blaster Audigy 2 Sound Card</td>
</tr>
<tr>
<td>64 Mb RAM</td>
<td>22.99</td>
<td><a href="http://www.ramemory.com">http://www.ramemory.com</a></td>
<td>64MB DIMM Memory</td>
</tr>
<tr>
<td>400 MHz PII</td>
<td>70.00</td>
<td><a href="http://www.ibm.com">http://www.ibm.com</a></td>
<td>CPU - 400MHZ PII</td>
</tr>
<tr>
<td>10 GB Hard drive</td>
<td>40.00</td>
<td><a href="http://www.amazon.com">http://www.amazon.com</a></td>
<td>Western Digital 10.1 GB HDD Internal Drive</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Software</th>
<th>Cost</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Free BSD</td>
<td>0.00</td>
<td><a href="http://www.freebsd.org/">http://www.freebsd.org/</a></td>
</tr>
<tr>
<td>Icecast</td>
<td>0.00</td>
<td><a href="http://www.icecast.org/">http://www.icecast.org/</a></td>
</tr>
<tr>
<td>PHP</td>
<td>0.00</td>
<td><a href="http://www.php.net/">http://www.php.net/</a></td>
</tr>
</tbody>
</table>

**Total** $212.87
4.2 Timeline

Senior Design I
- Problem Analysis
- Research
- Examine Existing Code

Senior Design II
- Logical Design
  - Ensure Icecast Developers are not currently planning this upgrade
  - Visit Station to learn how station functions
  - Flow chart new server functions
- Physical Design
  - Design Database
  - Determine methodology to use
- Preparation for Construction
  - Setup new server at WOXY for development
  - Gain FTP access to the new server

Senior Design III
- Construction
  - Install Icecast on OpenBSD server
  - Create database
  - Develop module
    - Ensure Icecast can talk to the database
    - Ensure audio stream is being output to logged in user
    - Ensure database locks correctly
    - Ensure database unlocks correctly
- Testing
  - Create help document
  - Allow users to test new functionality

4.3 Software

All software being used in this project will be open source and licensed under the GNU GPL. The operating system (OS) on the server will be the free version of Red Hat Linux. Finally all Web development will be in PHP which is an open source development language.
4.4 Hardware

For this project an IBM Intellistation Workstation was running Redhat Linux 8.0. The workstation had the following hardware configuration:

- 333 MHz Pentium III
- 64 MB RAM
- 6 GB IDE hard drive

5.0 Proof of Design

5.1 Testing Procedures

Testing of this new application has been done by a group of carefully selected users. The users were selected by Bryan Jay Miller, the IT professional at the radio station. These users were selected based on their participation in previous testing procedures done by Mr. Miller. The users were required to test the application from within a heavy firewall to make sure that the stream will have the ability to penetrate the firewall in large companies. The users were also selected for their variety in OS platforms. I needed to ensure that the users are able to connect to the server using Macintosh, Windows, and Linux. The users were required to report their findings to Mr. Miller who acted as the liaison between the testers and I.

In all cases the users were able to access the Icecast server since it has been put on port 80 even through a heavy company firewall. Linux users used the XMMS client application, Macintosh users used iTunes, and Windows users used WinAmp as their client application. To complete the testing I logged into the Icecast server through the Web interface and created a user for each tester. Then I opened up a stream for User1 and kept that stream running through the testing. All users were asked to log in as User1 and to see if a stream was received. In all cases the testers were unable to log in as User1.
Then I asked each user to log in with their specific user names and keep the stream up all night. No streams were disconnected and all users reported a successful test.

6.0 Conclusions and Recommendations

6.1 Conclusions

As I moved through this project I gained more and more understanding of the world of streaming audio. I would recommend to students in the future who take on this project to talk to the experts in the particular field and get book recommendations from them and read them in Senior Design I. I do know that I do not fully understand all the functionality of streaming audio, however extending my research during Senior Design II helped me gain a full understanding of the power of Icecast. I would also recommend to future students doing a project in this line of technology to take an extra quarter and study all the source code within the server. Without this time I do not believe I would have been able to have completed my project as efficiently as I have. I am extremely glad that I have been able to gain as much knowledge as I have about streaming audio so that I can take this into the business world and share it with my colleagues.

6.2 Recommendations

Through the implementation of this project I came across many enhancements that could be made to the Icecast server. The first enhancement would definitely be a documentation process. The hardest part of this project was understanding exactly what each piece of the server was doing and where I needed to insert the authentication code.
Appendix A: Code Snippets

1.0 Logon PHP File

```php
<?php

if( isset( $_POST['user'] ) && isset( $_POST['pass'] ) )
{
    echo "Thanks for logging in!<br><br>
    Please click on the link below to begin listening.<br><br>
    <a href="play.php?user=\".stripslashes($_POST['user'])."&pass=\".stripslashes($_POST['pass']).">Listen</a>;

    /*
    echo "<a
    href="http://\".stripslashes($_POST['user']).":\".stripslashes($_POST['pass'])."@12.47.104.146/woxy128\"">Listen</a>";
    */

} else
{
    echo "<p><strong>Please log in.</strong><p>\n
    echo "<form action="\"$_SERVER['PHP_SELF']" method="POST">\n
    Username: <input type=text name=user><br><br>
    Password: <input type=password name=pass><br><br>

    <input type=submit><input type=reset>";

}

?>
```

```html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <title>97X Listener Log-in 2</title>
</head>
<body>

<?php

    if( isset( $_POST['user'] ) && isset( $_POST['pass'] ) )
    {
        echo "Thanks for logging in!<br><br>
        Please click on the link below to begin listening.<br><br>
        <a href="play.php?user=\".stripslashes($_POST['user'])."&pass=\".stripslashes($_POST['pass']).">Listen</a>;

        /*
        echo "<a
        href="http://\".stripslashes($_POST['user']).":\".stripslashes($_POST['pass'])."@12.47.104.146/woxy128\"">Listen</a>";
        */
    }
    else
    {
        echo "<p><strong>Please log in.</strong><p>\n
        echo "<form action="\"$_SERVER['PHP_SELF']" method="POST">\n
        Username: <input type=text name=user><br><br>
        Password: <input type=password name=pass><br><br>

        <input type=submit><input type=reset>";
    }

?>
```
2.0 Play PHP File

```php
<?php
header('Content-type: audio/x-mpegurl');
echo "http://".stripslashes($_GET['user']).":".stripslashes($_GET['pass'])."@12.47.104.146/woxy128";
?>
```

3.0. icecast.XML modification

```xml
<fallback-mount>/example2.ogg</fallback-mount>
  <authentication type="htpasswd">
    <option name="filename" value="myauth"/>
    <option name="allow_duplicate_users" value="0"/>
  </authentication>
</mount>
```

4.0 auth.c modification

```c
int auth_is_listener_connected(source_t *source, char *username)
{
    client_t *client;
    avl_node *client_node;

    avl_tree_rlock(source->client_tree);

    client_node = avl_get_first(source->client_tree);
    while(client_node) {
        client = (client_t *)client_node->key;
        if (client->username) {
            if (!strcmp(client->username, username)) {
                avl_tree_unlock(source->client_tree);
                return 1;
            }
        }
        client_node = avl_get_next(client_node);
    }

    avl_tree_unlock(source->client_tree);
    return 0;
}
```

```c
auth_result auth_check_client(source_t *source, client_t *client)
{
    auth_t *authenticator = source->authenticator;
```
auth_result result;

if(authenticator) {
    /* This will look something like "Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==" */
    char *header = httpp_getvar(client->parser, "authorization");
    char *userpass, *tmp;
    char *username, *password;

    if(header == NULL)
        return AUTH_FAILED;

    if(strncmp(header, "Basic ", 6)) {
        INFO0("Authorization not using Basic");
        return 0;
    }

    userpass = util_base64_decode(header+6);
    if(userpass == NULL) {
        WARN1("Base64 decode of Authorization header "%s" failed",
             header+6);
        return AUTH_FAILED;
    }

    tmp = strchr(userpass, ':');
    if(!tmp) {
        free(userpass);
        return AUTH_FAILED;
    }

    *tmp = 0;
    username = userpass;
    password = tmp+1;

    result = authenticator->authenticate(
               authenticator, source, username, password);

    if(result == AUTH_OK)
        client->username = strdup(username);

    free(userpass);

    return result;
}
else
    return AUTH_FAILED;
typedef struct {
  char *filename;
  int allow_duplicate_users;
  rwlock_t file_rwlock;
} htpasswd_auth_state;

static auth_result htpasswd_auth(auth_t *auth, source_t *source, char *username, char *password)
{
  htpasswd_auth_state *state = auth->state;
  FILE *passwdfile = NULL;
  char line[MAX_LINE_LEN];
  char *sep;

  thread_rwlock_rlock(&state->file_rwlock);
  if (!state->allow_duplicate_users) {
    if (auth_is_listener_connected(source, username)) {
      thread_rwlock_unlock(&state->file_rwlock);
      return AUTH_FORBIDDEN;
    }
  }
  passwdfile = fopen(state->filename, "rb");
  if(passwdfile == NULL) {
    WARN2("Failed to open authentication database \\
            %s: %s", state->filename, strerror(errno));
    thread_rwlock_unlock(&state->file_rwlock);
    return AUTH_FAILED;
  }

  while(get_line(passwdfile, line, MAX_LINE_LEN)) {
    if(!line[0] || line[0] == '#')
      continue;

    sep = strchr(line, ':');
    if(sep == NULL) {
      DEBUG0("No seperator in line");
      continue;
    }

    *sep = 0;
    if(!strcmp(username, line)) {
      /* Found our user, now: does the hash of password match hash? */
      char *hash = sep+1;
      char *hashed_password = get_hash(password, strlen(password));
      if(!strcmp(hash, hashed_password)) {
        fclose(passwdfile);
      }
    }
  }
}

thread_rwlock_unlock(&state->file_rwlock);
if (passwdfile == NULL) {
  WARN2("Failed to open authentication database "
          "%s": %s", state->filename, strerror(errno));
  thread_rwlock_unlock(&state->file_rwlock);
  return AUTH_FAILED;
}

while(get_line(passwdfile, line, MAX_LINE_LEN)) {
  if(!line[0] || line[0] == '#')
    continue;

  sep = strchr(line, ':');
  if(sep == NULL) {
    DEBUG0("No seperator in line");
    continue;
  }

  *sep = 0;
  if(!strcmp(username, line)) {
    /* Found our user, now: does the hash of password match hash? */
    char *hash = sep+1;
    char *hashed_password = get_hash(password, strlen(password));
    if(!strcmp(hash, hashed_password)) {
      fclose(passwdfile);
    }
  }
}
free(hashed_password);
thread_rwlock_unlock(&state->file_rwlock);
return AUTH_OK;
}
free(hashed_password);
/* We don't keep searching through the file */
break;
}
}

fclose(passwdfile);

thread_rwlock_unlock(&state->file_rwlock);
return AUTH_FAILED;
}

static auth_t *auth_get_htpasswd_auth(config_options_t *options)
{
    auth_t *authenticator = calloc(1, sizeof(auth_t));
    htpasswd_auth_state *state;

    authenticator->authenticate = htpasswd_auth;
    authenticator->free = htpasswd_clear;

    state = calloc(1, sizeof(htpasswd_auth_state));

    state->allow_duplicate_users = 1;
    while(options) {
        if(!strcmp(options->name, "filename"))
            state->filename = strdup(options->value);
        if(!strcmp(options->name, "allow_duplicate_users"))
            state->allow_duplicate_users = atoi(options->value);
        options = options->next;
    }

    if(!state->filename) {
        free(state);
        free(authenticator);
        ERROR0("No filename given in options for authenticator.");
        return NULL;
    }

    authenticator->state = state;
    DEBUG1("Configured htpasswd authentication using password file %s", state->filename);
thread_rwlock_create(&state->file_rwlock);

return authenticator;
}

5.0 auth.h modification

/* Authenticate using the given username and password */
auth_result (*authenticate)(struct auth_tag *self,
    char *username, char *password);
    source_t *source, char *username, char *password);
    void (*free)(struct auth_tag *self);
    void *state;
    void *type;

6.0 client.c modification

void client_send_403(client_t *client) {
    int bytes = sock_write(client->con->sock,
            "HTTP/1.0 403 Forbidden\r\n"
            "\r\n"
            "Access restricted.\r\n");
    if(bytes > 0) client->con->sent_bytes = bytes;
    client->respcode = 403;
    client_destroy(client);
}

7.0 client.h modification

void client_send_403(client_t *client);

8.0 connection.c modification

#include "auth.h"

void connection_initialize(void)
{
    if (!_initialized) return;

    thread_mutex_create(&_connection_mutex);
    thread_mutex_create(&_queue_mutex);
    thread_mutex_create(&move_clients_mutex);
    thread_rwlock_create(&_source_shutdown_rwlock);
    thread_cond_create(&_pool_cond);
    thread_cond_create(&global.shutdown_cond);
}
void connection_shutdown(void)
{
    if (!_initialized) return;

    thread_cond_destroy(&global.shutdown_cond);
    thread_cond_destroy(&_pool_cond);
    thread_rwlock_destroy(&_source_shutdown_rwlock);
    thread_mutex_destroy(&_queue_mutex);
    thread_mutex_destroy(&_connection_mutex);
    thread_mutex_destroy(&move_clients_mutex);

    _initialized = 0;
}

int connection_complete_source (source_t *source)
{
    ice_config_t *config = config_get_config();

    global_lock();
    DEBUG1("sources count is %d", global.sources);

    if (global.sources < config->source_limit)
    {
        char *contenttype;
        mount_proxy *mountproxy = config->mounts;
        format_type_t format_type;

        /* setup format handler */
        contenttype = httpp_getvar (source->parser, "content-type");
        if (contenttype != NULL)
        {
            format_type = format_get_type (contenttype);

            if (format_type == FORMAT_ERROR)
            {
                global_unlock();
                config_release_config();
                if (source->client)
                    client_send_404 (source->client, "Content-type not supported");
                WARN1("Content-type \\
                        %s\n                        not supported, dropping source", contenttype);
                return -1;
            }
        }
    }
}
else
{
    WARN0("No content-type header, falling back to backwards compatibility mode ",
    "for icecast 1.x relays. Assuming content is mp3.");
    format_type = FORMAT_TYPE_MP3;
}

source->format = format_get_plugin (format_type, source->mount, source->parser);

if (source->format == NULL)
{
    global_unlock();
    config_release_config();
    if (source->client)
        client_send_404 (source->client, "internal format allocation problem");
    WARN1 ("plugin format failed for \"%s\", source->mount);
    return -1;
}

global.sources++;
global_unlock();

/* set global settings first */
source->queue_size_limit = config->queue_size_limit;
source->timeout = config->source_timeout;

if (source->client == NULL)
    source->client = client_create (source->con, source->parser);

while (mountproxy)
{
    if (strcmp (mountproxy->mountname, source->mount) == 0)
    {
        source_apply_mount (source, mountproxy);
        break;
    }
    mountproxy = mountproxy->next;
}
config_release_config();

source->shutdown_rwlock = &_source_shutdown_rwlock;
DEBUG0 ("source is ready to start");

return 0;
}
WARN1("Request to add source when maximum source limit"
"reached %d", global.sources);

global_unlock();
config_release_config();

if (source->client)
    client_send_404 (source->client, "too many sources connected");

    return -1;
}

static int _check_pass_http(http_parser_t *parser,
    char *correctuser, char *correctpass)
{
    /* This will look something like "Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==" */
    char *header = httpp_getvar(parser, "authorization");
    char *userpass, *tmp;
    char *username, *password;

    if(header == NULL)
        return 0;
    if(!strncmp(header, "Basic ", 6))
        return 0;

    userpass = util_base64_decode(header+6);
    if(userpass == NULL) {
        WARN1("Base64 decode of Authorization header \"%s\" failed",
            header+6);
        return 0;
    }

    tmp = strchr(userpass, ':');
    if(!tmp)
        free(userpass);
    return 0;
}

static int _check_pass_http(http_parser_t *parser,
    char *correctuser, char *correctpass)
{
    /* This will look something like "Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==" */
    char *header = httpp_getvar(parser, "authorization");
    char *userpass, *tmp;
    char *username, *password;

    if(header == NULL)
        return 0;
    if(!strncmp(header, "Basic ", 6))
        return 0;

    userpass = util_base64_decode(header+6);
    if(userpass == NULL) {
        WARN1("Base64 decode of Authorization header \"%s\" failed",
            header+6);
        return 0;
    }

    tmp = strchr(userpass, ':');
    if(!tmp)
        free(userpass);
    return 0;
}

    return 0;
}

free(userpass);
return 1;
}

int connection_check_source_pass(http_parser_t *parser, char *mount)
{
    ice_config_t *config = config_get_config();
    char *pass = config->source_password;
    char *user = "source";
    int ret;
    int ice_login = config->ice_login;
    char *protocol;

    mount_proxy *mountinfo = config->mounts;
    thread_mutex_lock(&(config_locks()->mounts_lock));

    while(mountinfo) {
        if(!strcmp(mountinfo->mountname, mount)) {
            if(mountinfo->password)
                pass = mountinfo->password;
            if(mountinfo->username)
                user = mountinfo->username;
            break;
        }
        mountinfo = mountinfo->next;
    }

    thread_mutex_unlock(&(config_locks()->mounts_lock));

    if(!pass) {
        WARN0("No source password set, rejecting source");
        config_release_config();
        return 0;
    }

    protocol = http_getvar(parser, HTTPP_VAR_PROTOCOL);
    if(protocol != NULL && !strcmp(protocol, "ICY")) {
        ret = _check_pass_icy(parser, pass);
    } else {
        ret = _check_pass_http(parser, user, pass);
        if(!ret && ice_login)
            ret = _check_pass_ice(parser, pass);
        if(ret)
            WARN0("Source is using deprecated icecast login");
    }
}
}  
config_release_config();  
return ret;  
}