Ultimate Achievements

Counter Strike 1.6 Achievement System

A Senior Project Presented for the  
Bachelor of Computer Science Degree  
Notre Dame University  
Lebanon

Tony Karam

June 2016

**Abstract**

Ultimate Achievements is a project that will help Counter Strike 1.6 players track and save their in-game progress to a database. That database can later be accessed from the website where a player can check the list of all the available achievements and see his own progress. The project is divided into three parts. The first part is the Plugin part. The plugin is installed on the servers, this plugin will be responsible of tracking the players’ progress and saving it to the database when the player is disconnected. The plugin will also have an authentication interface so that way no one can steal or manipulate any other player’s progress. The second part is the Database part. The database is where all the data regarding all the players and achievements are stored. There are several tables that each store a specified set of data. Two users have been created in the database, one for the Plugin and one for the Website. The third and final part is the Website part. The Website is where every player can go and check the list of all the available achievements. The ones that he earned and the ones he didn’t. A player can also see his current progress, that way he knows exactly how much of each achievement he needs to achieve it.

Table of Contents

[Chapter 1 – Introduction and General Information 5](#_Toc454077173)

[Introduction 5](#_Toc454077174)

[Problem Definition 5](#_Toc454077175)

[Chapter 2 – Tools and Methods 6](#_Toc454077176)

[Tools 6](#_Toc454077177)

[Amx Mod X 6](#_Toc454077178)

[WAMP Server 6](#_Toc454077179)

[Notepad++ 6](#_Toc454077180)

[PodBot 6](#_Toc454077181)

[Methods 7](#_Toc454077182)

[Hashing 7](#_Toc454077183)

[Sessions 8](#_Toc454077184)

[Chapter 3 - Functionality 9](#_Toc454077185)

[Database 9](#_Toc454077186)

[Website 10](#_Toc454077187)

[Plugin 10](#_Toc454077188)

[Chapter 4 - Code 12](#_Toc454077189)

[Plugin 12](#_Toc454077190)

[Enumerations 12](#_Toc454077191)

[Plugin Natives 13](#_Toc454077192)

[Client Natives 14](#_Toc454077193)

[MySQL Functions 15](#_Toc454077194)

[Menus 20](#_Toc454077195)

[Chapter 5 – User Interface 26](#_Toc454077196)

[Plugin 26](#_Toc454077197)

[Registration Message 26](#_Toc454077198)

[Registration Menu 26](#_Toc454077199)

[Password Entry 26](#_Toc454077200)

[Password Confirmation Menu 27](#_Toc454077201)

[Login Message 27](#_Toc454077202)

[Login Menu 27](#_Toc454077203)

[Successful Login 28](#_Toc454077204)

[Logout 28](#_Toc454077205)

[Website 28](#_Toc454077206)

[Sign Up Page 28](#_Toc454077207)

[Login Page 29](#_Toc454077208)

[Logout Page 30](#_Toc454077209)

[Achievements Page 30](#_Toc454077210)

[Profile Page 30](#_Toc454077211)

[Chapter 6 – Results 32](#_Toc454077212)

[Results 32](#_Toc454077213)

[Automation 32](#_Toc454077214)

[Security 32](#_Toc454077215)

[User Friendliness 32](#_Toc454077216)

[References 33](#_Toc454077217)

[Appendix 34](#_Toc454077218)

[SQL Tables Creation 34](#_Toc454077219)

# Chapter 1 – Introduction and General Information

## Introduction

The idea behind this project is to stimulate the sense of competition and achievement among Counter Strike players. That way the players will have an incentive to keep coming back to the server. This is what the project offers. This project offers a few achievements as proof of concept, there can always be room for many other achievements to be added and implemented. The goal of this project is to allow the player to join a server, choose whether he wishes to register or not and do so if he desires. Then he can join the website where he has to login with the information that he provides when he first registers on the server. After getting logged in, the person can then check his progress, available achievements and a small explanation of how the system works.

## Problem Definition

This senior project will help Counter Strike 1.6 players and server administrators easily track and monitor their progress while playing on the servers. Players will have a list of achievements that can be completed and earned. That person’s progress will be saved in a database every time the player disconnects from the server, that way the player can continue exactly where everything was left off. That way no progress is wasted, and each player will be able to see how far he has gone by logging in on the website. This will attract more players to the servers and give them a challenge that they can wish to complete.

# Chapter 2 – Tools and Methods

## Tools

### Amx Mod X

AmxModX is a versatile Half-Life metamod plugin which is targeted toward server administration. It has a wide array of scripting capabilities so people can write ‘plugins’, or files which add on to a mod’s functionality. Plugins can take form in administrative services, statistics generation, fun additions, gameplay changes, and much, much more! You can also write modules to expand the functionality of AmxModX and add to the scripting language.(Vincent Herbert, n.d.)

Ultimate Achievements uses AmxModX to take advantage of its forwards, hooks, and tools that it offers. The plugin would not be possible without AmxModX, therefore it is a requirement for all Counter Strike 1.6 servers that wishes to use Ultimate Achievements.

### WAMP Server

WampServer is a Windows web development environment. It allows you to create web applications with Apache2, PHP, and a MySQL database. Alongside, PHPMyAdmin that allows you to easily manage your databases. (Bourdon, n.d.)

WAMP Server was used to create a local Web Server on the local machine. That way the website can be hosted locally and show its functionality. This Web Server will also hold the database that both the Website and the Plugin will be accessing to store and read related data.

### Notepad++

Notepad++ is a free source code editor and Notepad replacement that supports several languages. Running in the MS Windows environment, its use is governed by GPL License.(Ho, n.d.)

All the written code, whether it’s for the Plugin or the Website, is written on Notepad++.

### PodBot

PODBot MetaMod is an open source (GPL) metamod plugin that adds computer players (bots) to a popular game called Counter-Strike. (Klinge, n.d.)

## Methods

### Hashing

The plugin as well as the website, uses md5 hashing to hash the password so it can be stored in the database in a non-readable format. Passwords are not stored in the database, however the hashed password is stored in the database. To increase security, a password is hashed with a randomly generated salt for each player. When a player registers on the server, the plugin will automatically generate a random 15 character string (called salt), concatenate it to the actual password, and finally hash the output string that is later stored in the database. That way, if two players use the same password, the hashed output will never be the same. When the player tries to login, the plugin will contact the database and get that player’s salt, then repeat the process that happened when the player registered. So the salt is concatenated to the actual password, and hashed. After getting the hashed password from the database, the two strings are compared. If they are equal, then the player will be logged in and his progress will be tracked, if not it means the player has input the wrong password. The same process for logging in is used on the website, however registering on the website is not allowed since the player must be in-game to retrieve his steam ID.

The following is a code snippet of the password hashing function in the Plugin:

HashPassword**(** strHashedPassword**[** **],** iHashedPasswordSize**,** strPassword**[** **],** strSalt**[** **]** **)** **{**

**new** strString**[** 128 **];**

formatex**(** strString**,** charsmax**(** strString **),** "%s%s"**,** strPassword**,** strSalt **);**

hash\_string**(** strString**,** Hash\_Md5**,** strHashedPassword**,** iHashedPasswordSize **-** 1 **);**

**}**

The following is a code snippet of the random generation of the salt:

GenerateRandomString**(** strSalt**[** **],** iSaltSize **)** **{**

**new** strAlphabet**[** **]** **=** "abcdefghijklmnopqrstuvwxyz"**;**

**new** strNumbers**[** **]** **=** "0123456789"**;**

**for(** **new** iLoop **=** 0**;** iLoop **<** iSaltSize**;** iLoop**++** **)** **{**

**if(** random\_num**(** 0**,** 1 **)** **)** **{**

strSalt**[** iLoop **]** **=** strNumbers**[** random\_num**(** 0**,** 9 **)** **];**

**}** **else** **{**

strSalt**[** iLoop **]** **=** strAlphabet**[** random\_num**(** 0**,** 25 **)** **];**

**if(** random\_num**(** 0**,** 1 **)** **)** **{**

strSalt**[** iLoop **]** **=** char\_to\_upper**(** strSalt**[** iLoop **]** **);**

**}**

**}**

**}**

**}**

### Sessions

The website uses sessions to store each player’s steam ID after he has logged in. After the player has logged in, his steam ID is stored in *‘$\_SESSION’*. That way we can know exactly who that player is, and display his appropriate information in his profile. The *‘$\_SESSION’* stays available until the player closes his browser, or logged out manually. In order to check if a player is logged in, we have to check if *‘$\_SESSION[ “steamid” ]’* is set, if so that means the user has logged in, if not that means he has yet to login. There is no way for the user to change the values inside *‘$\_SESSION’* so it is completely secure.

The following is a code snippet of the password check and session assignment that happens when the user logged in:

$row = **mysqli\_fetch\_array**( $sqlResult, MYSQL\_ASSOC );

$strDatabaseUserPassword = $row[ "Password" ];

$strDatabaseUserSalt = $row[ "Salt" ];

$strToHash = **sprintf**( "%s%s", $strUserPassword, $strDatabaseUserSalt );

**if**( **md5**( $strToHash ) == $strDatabaseUserPassword ) {

$\_SESSION[ "steamid" ] = $strUserSteamID;

**echo** '<div class="alert alert-success">';

**echo** '<strong>Sucess!</strong> <br><br>You have been successfully logged in.<br>';

**echo** 'You can now check your <a href="profile.php">profile</a>.';

**echo** '</div>';

}

# Chapter 3 - Functionality

## Database



The image above shows the EER of the database. Here is a small explanation of each element in the EER:

1. Player is the entity that represents each player that is registered (Primary Key = Steam ID):
   1. First Name (varchar 255): the first name of the player registered.
   2. Last Name (varchar 255): the last name of the player registered.
   3. Steam ID (varchar 18): the unique identifier that each player has when the player connects to the server.
   4. Password (varchar 255): the hashed password and salt of each player.
   5. Salt (varchar 255): the randomly generated 15 character string that is unique to each player.
2. Achievement is the entity that represents each achievement that is available (Primary Key = Achievement ID):
   1. Achievement ID (int): the unique auto incremented identifier that represents each achievement.
   2. Name (varchar 255): the name of the achievement in question.
   3. Description (varchar 512): a small piece of text that represents what the player is required to accomplish to acquire the achievement.
   4. Goal (int): the number of times each action should be accomplished in order to acquire the achievement.
3. Achieves is a many to many relationship between a player and an achievement (Primary Key = Steam ID and Achievement ID):
   1. Progress (int): the progress will be the number of how many times the player has accomplished the required action, when this number is equal to the goal of the achievement that means the player has acquired the achievement.

The database is created with a series of MySQL queries that the administrator will need to execute when installing the system. The code required to do so is present in the appendix labeled ‘SQL Tables Creation’.

For the sake of testing, sample data will be inserted in the database. The code to insert the data used is also available in the appendix labeled ‘SQL Sample Data’.

## Website

The website is divided into several pages:

1. Home Page:
   1. Display a small description of what the project is and what it aims to achieve.
   2. Three interesting features that the project delivers.
2. Achievements Page:
   1. List each achievement’s name.
   2. List each achievement’s description.
   3. List each achievement’s goal.
3. Profile Page:
   1. Check list of acquired achievements.
   2. Check list of un-acquired achievements.
   3. Option to reset player progress.
   4. Display information such as First Name, Last Name, and Steam ID.
4. Sign UP Page: a small note informing players that signing up is only available on the server.
5. Login Page:
   1. Form that a player can use to login to the website.
   2. Information box to notify user how to retrieve his Steam ID.
6. Logout Page: a page dedicated to logging the player out of his account.

## Plugin

The plugin achieves the following points:

* Catch when a player has connected to the server.
* Check if player has already registered:
  + Tell player to login if he desires.
  + Allow player to register if he desires.
* Logging in/out:
  + Allow player to login so his progress is saved.
  + Allow player to logout if he wishes to stop recording his progress.
* Catch when a player has disconnected from the server and save player’s progress.
* Catch several events and/or actions to update player’s achievement progress (amount of kills, amount of bomb defuses, amount of bomb plants … etc).

# Chapter 4 - Code

In this chapter, we will be taking a closer look at each part of the project, explaining the reasoning behind the most important situations and cases as we go along.

## Plugin

### Enumerations

/\* Enumerations \*/

enum \_**:**ACHIEVEMENT\_MAX**(** **)** **{**

ACHIEVEMENT\_AWARDIST **=** 1**,** ACHIEVEMENT\_PROMOTED**,**

ACHIEVEMENT\_BOMB\_EXPERT**,** ACHIEVEMENT\_DEFUSAL\_EXPERT**,**

ACHIEVEMENT\_CLUTCH\_MASTER**,** ACHIEVEMENT\_WAR\_BONDS**,**

ACHIEVEMENT\_SPENDING\_SPREE**,** ACHIEVEMENT\_ACE\_MASTER**,**

ACHIEVEMENT\_BODY\_BAGGER**,** ACHIEVEMENT\_BATTLE\_SIGHT\_ZERO**,**

ACHIEVEMENT\_POINTS\_IN\_YOUR\_FAVOR**,** ACHIEVEMENT\_MAKE\_THE\_CUT**,**

ACHIEVEMENT\_SOMEONE\_SET\_US\_BOMB**,** ACHIEVEMENT\_RITE\_OF\_FIRST\_DEFUSAL**,**

ACHIEVEMENT\_GIVE\_PIECE\_A\_CHANCE

**};**

enum \_**:**ACHIEVEMENT\_COLUMN\_MAX**(** **)** **{**

ACHIEVEMENT\_COLUMN\_ID **=** 0**,**

ACHIEVEMENT\_COLUMN\_NAME**,**

ACHIEVEMENT\_COLUMN\_DESCRIPTION**,**

ACHIEVEMENT\_COLUMN\_GOAL

**};**

enum \_**:**PLAYER\_COLUMN\_MAX**(** **)** **{**

PLAYER\_COLUMN\_STEAMID **=** 0**,**

PLAYER\_COLUMN\_FIRST\_NAME**,**

PLAYER\_COLUMN\_LAST\_NAME**,**

PLAYER\_COLUMN\_PASSWORD**,**

PLAYER\_COLUMN\_SALT

**};**

enum \_**:**ACHIEVES\_COLUMN\_MAX**(** **)** **{**

ACHIEVES\_COLUMN\_STEAMID **=** 0**,**

ACHIEVES\_COLUMN\_ID**,**

ACHIEVES\_COLUMN\_PROGRESS**,**

ACHIEVES\_COLUMN\_ACQUIRED

**};**

Instead of using plain integer numbers to represent each achievement, we use enumerations to increase the readability of the code. The upside of enumerations, other than readability, is how easy it is to change or other elements. For example, if we would like to add an achievement, all that is needed is to simply add it to the enum and then use it in the code.

### Plugin Natives

/\* Plugin Natives \*/

public plugin\_init**(** **)** **{**

register\_plugin**(** g\_strPluginName**,** g\_strPluginVersion**,** g\_strPluginAuthor **);**

register\_cvar**(** g\_strPluginName**,** g\_strPluginVersion**,** FCVAR\_SERVER **|** FCVAR\_EXTDLL **|** FCVAR\_UNLOGGED **|** FCVAR\_SPONLY **);**

register\_clcmd**(** "REGISTER\_PASSWORD"**,** "ClCmd\_RegisterPassword" **);**

register\_clcmd**(** "LOGIN\_PASSWORD"**,** "ClCmd\_LoginPassword" **);**

register\_clcmd**(** "FIRST\_NAME"**,** "ClCmd\_FirstName" **);**

register\_clcmd**(** "LAST\_NAME"**,** "ClCmd\_LastName" **);**

register\_clcmd**(** "say /register"**,** "ClCmd\_RegisterMenu" **);**

register\_clcmd**(** "say /login"**,** "ClCmd\_LoginMenu" **);**

register\_clcmd**(** "say /logout"**,** "ClCmd\_LogoutMenu" **);**

register\_clcmd**(** "say\_team /register"**,** "ClCmd\_RegisterMenu" **);**

register\_clcmd**(** "say\_team /login"**,** "ClCmd\_LoginMenu" **);**

register\_clcmd**(** "say\_team /logout"**,** "ClCmd\_LogoutMenu" **);**

RegisterHam**(** Ham\_Spawn**,** "player"**,** "Ham\_Spawn\_Player\_Post"**,** **true** **);**

RegisterHam**(** Ham\_Killed**,** "player"**,** "Ham\_Killed\_Player\_Pre"**,** **false** **);**

RegisterHam**(** Ham\_TakeDamage**,** "player"**,** "Ham\_TakeDamage\_Player\_Pre"**,** **false** **);**

register\_event**(** "SendAudio"**,** "Event\_SendAudio\_TerroristWin"**,** "a"**,** "2&%!MRAD\_terwin" **);**

register\_event**(** "SendAudio"**,** "Event\_SendAudio\_CounterTerroristWin"**,** "a"**,** "2&%!MRAD\_ctwin" **);**

set\_task**(** 0.1**,** "MySQL\_InitializeConnection" **);**

**}**

public plugin\_end**(** **)** **{**

SQL\_FreeHandle**(** g\_sqlTuple **);**

**}**

Just like in C++ where we have the *‘main( )’* function that is called at the start of the program, Amx Mod X offers several natives directly related to the plugin. One of those natives is the *‘plugin\_init( )’* that is called when the plugin is first loaded on the server. Here, in this function, all the commands registrations and hooks are created. For example, in the plugin, in order to bring up the registration menu, a player must type in chat *‘/register’*. This is where that command is registered. All *‘register\_clcmd( )’* natives take two arguments: the first is the actual command that the player has to type and the second is the function to call whenever that player issues that command.

### Client Natives

/\* Client Natives \*/

public client\_authorized**(** iPlayerID **)** **{**

ClearBit**(** g\_bitIsLoggedIn**,** iPlayerID **);**

ClearBit**(** g\_bitIsRegistered**,** iPlayerID **);**

g\_strPlayerFirstName**[** iPlayerID **]** **=** ""**;**

g\_strPlayerLastName**[** iPlayerID **]** **=** ""**;**

ClearPlayerAchievementProgress**(** iPlayerID **);**

MySQL\_PlayerConnected**(** iPlayerID **);**

**}**

public client\_disconnected**(** iPlayerID **)** **{**

MySQL\_SavePlayerStats**(** iPlayerID **);**

**}**

Amx Mod X also offers several client natives. These natives allows us to catch player connection and disconnection from the server and execute code accordingly. At first, we catch when the player is authorized (*client\_authorized( )* is different from *client\_connected( )* – the difference is that in *client\_authorized( )* the connected player is guaranteed to have a steam ID where in *client\_connected( )* it is not, and since the steam ID is important at this stage we use *client\_authorized( )*). Then we clear the player’s ‘*g\_bitIsLoggedIn’* and ‘*g\_bitIsRegistered’* bits. These bits will us to know if the player is registered or logged in. And since when a player disconnects from a server and another player connects, that new player might end up with the old player’s ID (in-game ID is different from the steam ID), so we clear those bits since that player is new. The strings that hold the player’s first and last name are also cleared since he is a new player. Lastly we call the *‘MySQL\_PlayerConnected( )’* function to start the authentication process (this function will be explained in detail later on in this chapter).

At *‘client\_disconnected( )’*, we are catching when the player has disconnected from the server. That means it’s time to save the player’s progress, and that’s why we call the *‘MySQL\_SavePlayerStats( )’* function that will deal with it (this function will be explained in detail later on in this chapter).

### MySQL Functions

All user defined (not natives) functions that contact the database start with MySQL\_. The functions will be responsible of reading, deleting, inserting, and updating the database records. Each set of MySQL functions will be explained shortly below.

#### Initialization

public MySQL\_InitializeConnection**(** **)** **{**

g\_sqlTuple **=** SQL\_MakeDbTuple**(** g\_strSQLHost**,** g\_strSQLUsername**,** g\_strSQLPassword**,** g\_strSQLDatabase **);**

**new** iErrorCode**;**

**new** Handle**:**sqlConnection **=** SQL\_Connect**(** g\_sqlTuple**,** iErrorCode**,** g\_strErrorMessage**,** charsmax**(** g\_strErrorMessage **)** **);**

**if(** sqlConnection **==** Empty\_Handle **)** **{**

set\_fail\_state**(** g\_strErrorMessage **);**

**}**

SQL\_FreeHandle**(** sqlConnection **);**

MySQL\_GetAchievementGoals**(** **);**

**}**

The above function is responsible of initializing the MySQL connection. First it takes the Host, Username, Password, and Database name and makes the tuple that will be used to make the connection. If for any reason, the connection was not successful, this function will stop the plugin from executing by calling *‘set\_fail\_state( )’* that will change the plugin’s state to failed and output the error message to the logs. Then it frees the handle so it can be used later by another function before calling *‘MySQL\_GetAchievementGoals( )’* that will read the database for the achievements’ goal and store it in a global variable.

#### Player Connection and Player Registration

MySQL\_PlayerConnected**(** iPlayerID **)** **{**

**new** strPlayerID**[** 1 **];**

strPlayerID**[** 0 **]** **=** iPlayerID**;**

**new** strPlayerAuthID**[** 36 **];**

get\_user\_authid**(** iPlayerID**,** strPlayerAuthID**,** charsmax**(** strPlayerAuthID **)** **);**

**new** strQuery**[** 512 **];**

formatex**(** strQuery**,** charsmax**(** strQuery **),** "SELECT \* FROM Player WHERE Player.SteamID = '%s'"**,** strPlayerAuthID **);**

SQL\_ThreadQuery**(** g\_sqlTuple**,** "MySQL\_Answer\_PlayerConnected"**,** strQuery**,** strPlayerID**,** **sizeof(** strPlayerID **)** **);**

**}**

public MySQL\_Answer\_PlayerConnected**(** iFailState**,** Handle**:**hQuery**,** strErrorMessage**[** **],** iErrorCode**,** strPlayerID**[** **],** iSize **)** **{**

**new** iPlayerID **=** strPlayerID**[** 0 **];**

**if(** iFailState **==** TQUERY\_CONNECT\_FAILED **)** **{**

log\_amx**(** "Could not connect to the MySQL Database." **);**

log\_amx**(** "[%d] %s"**,** iErrorCode**,** strErrorMessage **);**

**}** **else** **if(** iFailState **==** TQUERY\_QUERY\_FAILED **)** **{**

log\_amx**(** "Query failed" **);**

log\_amx**(** "[%d] %s"**,** iErrorCode**,** strErrorMessage **);**

**}** **else** **{**

**if(** SQL\_NumResults**(** hQuery **)** **)** **{**

SetBit**(** g\_bitIsRegistered**,** iPlayerID **);**

**}**

**}**

SQL\_FreeHandle**(** hQuery **);**

**}**

MySQL\_RegisterPlayer**(** iPlayerID**,** strPassword**[** **]** **)** **{**

**new** strPlayerID**[** 1 **];**

strPlayerID**[** 0 **]** **=** iPlayerID**;**

**new** strPlayerAuthID**[** 36 **];**

get\_user\_authid**(** iPlayerID**,** strPlayerAuthID**,** charsmax**(** strPlayerAuthID **)** **);**

**new** strSalt**[** 16 **];**

GenerateRandomString**(** strSalt**,** charsmax**(** strSalt **)** **);**

**new** strHashedPassword**[** 34 **];**

HashPassword**(** strHashedPassword**,** **sizeof(** strHashedPassword **),** strPassword**,** strSalt **);**

**new** strQuery**[** 512 **];**

formatex**(** strQuery**,** charsmax**(** strQuery **),** "INSERT INTO Player ( SteamID, FirstName, LastName, Password, Salt ) VALUES ( '%s', '%s', '%s', '%s', '%s' );"**,** strPlayerAuthID**,** g\_strPlayerFirstName**[** iPlayerID **],** g\_strPlayerLastName**[** iPlayerID **],** strHashedPassword**,** strSalt **);**

SQL\_ThreadQuery**(** g\_sqlTuple**,** "MySQL\_Answer\_RegisterPlayer"**,** strQuery**,** strPlayerID**,** **sizeof(** strPlayerID **)** **);**

**}**

public MySQL\_Answer\_RegisterPlayer**(** iFailState**,** Handle**:**hQuery**,** strErrorMessage**[** **],** iErrorCode**,** strData**[** **],** iDataSize **)** **{**

**new** iPlayerID **=** strData**[** 0 **];**

**if(** iFailState **==** TQUERY\_CONNECT\_FAILED **)** **{**

log\_amx**(** "Could not connect to the MySQL Database." **);**

log\_amx**(** "[%d] %s"**,** iErrorCode**,** strErrorMessage **);**

client\_print**(** iPlayerID**,** print\_chat**,** "[ Achievements ] There has been a problem registering your password. Please try again" **);**

**}** **else** **if(** iFailState **==** TQUERY\_QUERY\_FAILED **)** **{**

log\_amx**(** "Query failed" **);**

log\_amx**(** "[%d] %s"**,** iErrorCode**,** strErrorMessage **);**

client\_print**(** iPlayerID**,** print\_chat**,** "[ Achievements ] There has been a problem registering your password. Please try again" **);**

**}** **else** **{**

SetBit**(** g\_bitIsRegistered**,** iPlayerID **);**

client\_print**(** iPlayerID**,** print\_chat**,** "[ Achievements ] You have been successfully registered." **);**

client\_print**(** iPlayerID**,** print\_chat**,** "[ Achievements ] Please type /login to login." **);**

**}**

SQL\_FreeHandle**(** hQuery **);**

**}**

The first function *‘MySQL\_PlayerConnected( )’* is responsible of contacting the database and checking if the player already has a record in the Player table. If a record was found, that means the user has already registered, so it sets the player’s *‘g\_bitIsRegistered’* bit to 1. If a record was not found, nothing is done since the *‘g\_bitIsRegistered’* for that player is already set to 0 (it was cleared in *client\_authorized( )* when the player first connected).

The second function *‘MySQL\_RegisterPlayer( )’* is responsible for creating a record for that player after he has registered. The function starts by getting generating a random 15 character string (this string is used as the salt) by calling the *‘GenerateRandomString( )’* function where the *‘strSalt’* variable is automatically passed by reference. Then the function continues by generating the hashed password that will be stored in the database by calling the *‘HashPassword( )’* function where the *‘strHashedPassword’* variable is also automatically passed by reference. After that, the query is formatted and sent. When the answer comes back from the database, appropriate error messages are shown to the player if there are any, or there *‘b\_gitIsRegistered’* bit is set to true. Now the player can go ahead and login to his account that was just created.

#### Player Login

MySQL\_GetPlayerInfo**(** iPlayerID **)** **{**

**new** strPlayerID**[** 1 **];**

strPlayerID**[** 0 **]** **=** iPlayerID**;**

**new** strPlayerAuthID**[** 36 **];**

get\_user\_authid**(** iPlayerID**,** strPlayerAuthID**,** charsmax**(** strPlayerAuthID **)** **);**

**new** strQuery**[** 512 **];**

formatex**(** strQuery**,** charsmax**(** strQuery **),** "SELECT \* FROM Player WHERE ( Player.SteamID = '%s' )"**,** strPlayerAuthID **);**

SQL\_ThreadQuery**(** g\_sqlTuple**,** "MySQL\_Answer\_GetPlayerInfo"**,** strQuery**,** strPlayerID**,** **sizeof(** strPlayerID **)** **);**

**}**

public MySQL\_Answer\_GetPlayerInfo**(** iFailState**,** Handle**:**hQuery**,** strErrorMessage**[** **],** iErrorCode**,** strData**[** **],** iDataSize **)** **{**

**new** iPlayerID **=** strData**[** 0 **];**

**if(** iFailState **==** TQUERY\_CONNECT\_FAILED **)** **{**

log\_amx**(** "Could not connect to the MySQL Database." **);**

log\_amx**(** "[%d] %s"**,** iErrorCode**,** strErrorMessage **);**

client\_print**(** iPlayerID**,** print\_chat**,** "[ Achievements ] There has been a problem connecting to the database. Please try again." **);**

**}** **else** **if(** iFailState **==** TQUERY\_QUERY\_FAILED **)** **{**

log\_amx**(** "Query Failed" **);**

log\_amx**(** "[%d] %s"**,** iErrorCode**,** strErrorMessage **);**

client\_print**(** iPlayerID**,** print\_chat**,** "[ Achievements ] It seems that you are not registered. Please register first." **);**

**}**

**new** strPlayerHashedPassword**[** 34 **];**

SQL\_ReadResult**(** hQuery**,** PLAYER\_COLUMN\_PASSWORD**,** strPlayerHashedPassword**,** charsmax**(** strPlayerHashedPassword **)** **);**

SQL\_ReadResult**(** hQuery**,** PLAYER\_COLUMN\_SALT**,** g\_strPlayerSalt**[** iPlayerID **],** charsmax**(** g\_strPlayerSalt**[** **]** **)** **);**

CheckPasswordValidity**(** iPlayerID**,** strPlayerHashedPassword **);**

SQL\_FreeHandle**(** hQuery **);**

**}**

When a player has provided his password, *‘MySQL\_GetPlayerInfo( )’* is called where a query is executed that retrieves the player’s information. That information includes the player’s hashed password that is already stored in the database and the player’s salt. These two will be used to validate the password that the player has provided. When the results are in, the hashed password and the salt are stored in variables and then the *‘CheckPasswordValidity( )’* is called that checks if the passwords match.

#### Player Stats (Get, Save, and Insert)

MySQL\_GetPlayerStats**(** iPlayerID **)** **{**

**new** strPlayerID**[** 1 **];**

strPlayerID**[** 0 **]** **=** iPlayerID**;**

**new** strPlayerAuthID**[** 36 **];**

get\_user\_authid**(** iPlayerID**,** strPlayerAuthID**,** charsmax**(** strPlayerAuthID **)** **);**

**new** strQuery**[** 512 **];**

formatex**(** strQuery**,** charsmax**(** strQuery **),** "SELECT \* From Achieves WHERE ( Achieves.SteamID = '%s' )"**,** strPlayerAuthID **);**

SQL\_ThreadQuery**(** g\_sqlTuple**,** "MySQL\_Answer\_GetPlayerStats"**,** strQuery**,** strPlayerID**,** **sizeof(** strPlayerID **)** **);**

**}**

public MySQL\_Answer\_GetPlayerStats**(** iFailState**,** Handle**:**hQuery**,** strErrorMessage**[** **],** iErrorCode**,** strData**[** **],** iDataSize **)** **{**

**new** iPlayerID **=** strData**[** 0 **];**

**if(** iFailState **==** TQUERY\_CONNECT\_FAILED **)** **{**

log\_amx**(** "Could not connect to the MySQL Database." **);**

log\_amx**(** "[%d] %s"**,** iErrorCode**,** strErrorMessage **);**

client\_print**(** iPlayerID**,** print\_chat**,** "[ Achievements ] There has been a problem connecting to the database. Please try again." **);**

**}** **else** **if(** iFailState **==** TQUERY\_QUERY\_FAILED **)** **{**

log\_amx**(** "Query Failed" **);**

log\_amx**(** "[%d] %s"**,** iErrorCode**,** strErrorMessage **);**

client\_print**(** iPlayerID**,** print\_chat**,** "[ Achievements ] There has been a problem retrieving your stats. Please try again." **);**

**}** **else** **{**

**if(** **!**SQL\_NumResults**(** hQuery **)** **)** **{**

MySQL\_InsertPlayerAchieves**(** iPlayerID **);**

**}** **else** **{**

**new** iID**,** iProgress**;**

**for(** **new** iLoop **=** 0**;** iLoop **<** SQL\_NumResults**(** hQuery **);** iLoop**++** **)** **{**

iID **=** SQL\_ReadResult**(** hQuery**,** ACHIEVES\_COLUMN\_ID **);**

iProgress **=** SQL\_ReadResult**(** hQuery**,** ACHIEVES\_COLUMN\_PROGRESS **);**

g\_iPlayerAchievementProgress**[** iPlayerID **][** iID **]** **=** iProgress**;**

**}**

**}**

**}**

SQL\_FreeHandle**(** hQuery **);**

**}**

MySQL\_InsertPlayerAchieves**(** iPlayerID **)** **{**

**new** strPlayerID**[** 1 **];**

strPlayerID**[** 0 **]** **=** iPlayerID**;**

**new** strPlayerAuthID**[** 36 **];**

get\_user\_authid**(** iPlayerID**,** strPlayerAuthID**,** charsmax**(** strPlayerAuthID **)** **);**

**new** strQuery**[** 2048 **]** **=** ""**;**

**for(** **new** iLoop **=** 1**;** iLoop **<** ACHIEVEMENT\_MAX**;** iLoop**++** **)** **{**

format**(** strQuery**,** charsmax**(** strQuery **),** "%s INSERT INTO Achieves ( SteamID, ID, Progress, Acquired ) VALUES ( '%s', %d, 0, false );"**,** strQuery**,** strPlayerAuthID**,** iLoop **);**

**}**

SQL\_ThreadQuery**(** g\_sqlTuple**,** "MySQL\_Answer\_InsertPlayerStats"**,** strQuery**,** strPlayerID**,** **sizeof(** strPlayerID **)** **);**

**}**

public MySQL\_Answer\_InsertPlayerStats**(** iFailState**,** Handle**:**hQuery**,** strErrorMessage**[** **],** iErrorCode**,** strData**[** **],** iDataSize **)** **{**

**new** iPlayerID **=** strData**[** 0 **];**

**if(** iFailState **==** TQUERY\_CONNECT\_FAILED **)** **{**

log\_amx**(** "Could not connect to the MySQL Database." **);**

log\_amx**(** "[%d] %s"**,** iErrorCode**,** strErrorMessage **);**

client\_print**(** iPlayerID**,** print\_chat**,** "[ Achievements ] There has been a problem connecting to the database. Please try again." **);**

**}** **else** **if(** iFailState **==** TQUERY\_QUERY\_FAILED **)** **{**

log\_amx**(** "Query Failed" **);**

log\_amx**(** "[%d] %s"**,** iErrorCode**,** strErrorMessage **);**

client\_print**(** iPlayerID**,** print\_chat**,** "[ Achievements ] There has been a problem inserting your stats. Please try again." **);**

**}** **else** **{**

client\_print**(** iPlayerID**,** print\_chat**,** "[ Achievements ] Your stats has been inserted into our database. Start playing to increase your stats." **);**

**}**

SQL\_FreeHandle**(** hQuery **);**

**}**

MySQL\_SavePlayerStats**(** iPlayerID **)** **{**

**new** strPlayerID**[** 1 **];**

strPlayerID**[** 0 **]** **=** iPlayerID**;**

**new** strPlayerAuthID**[** 36 **];**

get\_user\_authid**(** iPlayerID**,** strPlayerAuthID**,** charsmax**(** strPlayerAuthID **)** **);**

**new** strQuery**[** 2048 **]** **=** ""**;**

**new** iProgress**;**

**for(** **new** iLoop **=** 1**;** iLoop **<** ACHIEVEMENT\_MAX**;** iLoop**++** **)** **{**

iProgress **=** g\_iPlayerAchievementProgress**[** iPlayerID **][** iLoop **];**

format**(** strQuery**,** charsmax**(** strQuery **),** "%s UPDATE Achieves SET Achieves.Progress = %d, Achieves.Acquired = %d WHERE Achieves.SteamID = '%s' AND Achieves.ID = %d;"**,** strQuery**,** iProgress**,** **(** g\_bPlayerAchievementAcquired**[** iLoop **]** **)** **?** 1 **:** 0**,** strPlayerAuthID**,** iLoop **);**

**}**

SQL\_ThreadQuery**(** g\_sqlTuple**,** "MySQL\_Answer\_SavePlayerStats"**,** strQuery**,** strPlayerID**,** **sizeof(** strPlayerID **)** **);**

**}**

public MySQL\_Answer\_SavePlayerStats**(** iFailState**,** Handle**:**hQuery**,** strErrorMessage**[** **],** iErrorCode**,** strData**[** **],** iDataSize **)** **{**

**if(** iFailState **==** TQUERY\_CONNECT\_FAILED **)** **{**

log\_amx**(** "Could not connect to the MySQL Database." **);**

log\_amx**(** "[%d] %s"**,** iErrorCode**,** strErrorMessage **);**

**}** **else** **if(** iFailState **==** TQUERY\_QUERY\_FAILED **)** **{**

log\_amx**(** "Query Failed" **);**

log\_amx**(** "[%d] %s"**,** iErrorCode**,** strErrorMessage **);**

**}**

SQL\_FreeHandle**(** hQuery **);**

**}**

The first function *‘MySQL\_GetPlayerStats( )’* is the function responsible for contacting the database and retrieving everything from the table Achieves regarding a specific player, then it saves the data into 2 two dimensional arrays. The first array is *‘g\_iPlayerAchievementProgress[ MAX\_PLAYERS ][ ACHIEVEMENT\_MAX ]’* where the player’s progress for each achievement is saved. The second array is *‘g\_bPlayerAchievementAcquired[ MAX\_PLAYERS ][ ACHIEVEMENT\_MAX ]’* where each player’s acquired status for each achievement is stored. The 2 arrays mentioned above are two dimensional arrays that has a total of MAX\_PLAYERS x ACHIEVEMENT\_MAX cells, which is one cell per achievement per player.

The second function *‘MySQL\_InsertPlayerAchieves( )’* is the function responsible for inserting all the necessary records in the Achieves table in the database for all the newly registered players. For example, when a player registers on the server, his record (First Name, Last Name, Password, and Salt) is added to the Player table in the database. However, his records (that should all be zeroed since he newly registered) are not yet inserted in the Achieves table. This is what this function does. It creates a big query that inserts each achievement with its progress column set to 0 and acquired column set to false for that player.

The third function *‘’MySQL\_SavePlayerStats( )’* is the function responsible for taking the player’s *g\_iPlayerAchievementProgress[ ][ ]’* and *‘g\_bPlayerAchievementAcquired[ ][ ]’* respective arrays and update that player’s records in the database with those values.

### Menus

The best way to interact with the player in game where data is exchanged between the plugin and player is by using menus. A menu is a text that is printed on the left side of the player’s screen with several options to choose from. A player can choose an option he desires by pressing the numerical buttons on his keyboard. Some snapshots will be posted in Chapter 5. Each menus has a title that describes what the menu is about with a small optional message that is intended for the player. A series of options with numbers on the left side that represents the number that the player can press on his keyboard to choose a desired option.

#### Login Menu

ShowPlayerLogInMenu**(** iPlayerID **)** **{**

static menuPlayerLogIn**;**

**if(** **!**menuPlayerLogIn **)** **{**

**new** strMenuTitle**[** 128 **]** **=** "\yAchievement Log In Menu:^n^n\yNote: \wIf you are not logged in,^nyour progress will not be saved."**;**

menuPlayerLogIn **=** menu\_create**(** strMenuTitle**,** "Handle\_PlayerLogInMenu" **);**

menu\_additem**(** menuPlayerLogIn**,** "\wEnter Password" **);**

menu\_additem**(** menuPlayerLogIn**,** "\wLog In Later" **);**

menu\_setprop**(** menuPlayerLogIn**,** MPROP\_NUMBER\_COLOR**,** "\y" **);**

menu\_setprop**(** menuPlayerLogIn**,** MPROP\_EXIT**,** MEXIT\_NEVER **);**

**}**

menu\_display**(** iPlayerID**,** menuPlayerLogIn**,** 0 **);**

**}**

public Handle\_PlayerLogInMenu**(** iPlayerID**,** iMenu**,** iKey **)** **{**

**switch(** iKey **)** **{**

**case** 0**:** **{**

client\_cmd**(** iPlayerID**,** "messagemode LOGIN\_PASSWORD" **);**

**}**

**case** 1**:** **{**

client\_print**(** iPlayerID**,** print\_chat**,** "[ Achievements ] You have chosen to login later. You can type /login to login later on." **);**

**}**

**}**

**}**

This menu will give the player two options. The first is to enter his password so he can be logged in, and the second is an option to close the menu if he decided not to register anymore. When the player chooses the first option, a command is executed by the plugin that opens a text message section on the top left corner of his screen. The player can then proceed to input his password that is then sent to the *‘MySQL\_GetPlayerInfo( )’* function that has been discussed in this chapter.

#### Registration Menu

ShowPlayerRegistrationMenu**(** iPlayerID **)** **{**

static menuPlayerRegistration**;**

**if(** **!**menuPlayerRegistration **)** **{**

**new** strMenuTitle**[** 128 **]** **=** "\yAchievement Registration Menu:^n^n\yNote: \wIf you want your progress to be tracked,^nyou must register first."**;**

menuPlayerRegistration **=** menu\_create**(** strMenuTitle**,** "Handle\_PlayerRegistrationMenu" **);**

menu\_additem**(** menuPlayerRegistration**,** "\wEnter Password" **);**

menu\_additem**(** menuPlayerRegistration**,** "\wEnter First Name" **);**

menu\_additem**(** menuPlayerRegistration**,** "\wEnter Last Name" **);**

menu\_additem**(** menuPlayerRegistration**,** "\wRegister Later" **);**

menu\_setprop**(** menuPlayerRegistration**,** MPROP\_NUMBER\_COLOR**,** "\y" **);**

menu\_setprop**(** menuPlayerRegistration**,** MPROP\_EXIT**,** MEXIT\_NEVER **);**

**}**

menu\_di splay**(** iPlayerID**,** menuPlayerRegistration**,** 0 **);**

**}**

public Handle\_PlayerRegistrationMenu**(** iPlayerID**,** iMenu**,** iKey **)** **{**

**switch(** iKey **)** **{**

**case** 0**:** **{**

**if(** equal**(** g\_strPlayerFirstName**[** iPlayerID **],** "" **)** **||** equal**(** g\_strPlayerLastName**[** iPlayerID **],** "" **)** **)** **{**

client\_print**(** iPlayerID**,** print\_chat**,** "[ Achievements ] Please input your First Name and Last Name before entering your password." **);**

ShowPlayerRegistrationMenu**(** iPlayerID **);**

**}** **else** **{**

client\_cmd**(** iPlayerID**,** "messagemode REGISTER\_PASSWORD" **);**

**}**

**}**

**case** 1**:** **{**

client\_cmd**(** iPlayerID**,** "messagemode FIRST\_NAME" **);**

**}**

**case** 2**:** **{**

client\_cmd**(** iPlayerID**,** "messagemode LAST\_NAME" **);**

**}**

**case** 3**:** **{**

client\_print**(** iPlayerID**,** print\_chat**,** "[ Achievements ] You have chosen to register later. You can type /register to register later on." **);**

**}**

**}**

**}**

The registration menu will ask the user to input three fields: password, first name, and last name. The player also has an option to opt out of the registration if he so desires. A player cannot enter the password if he did not enter his first name and last name. When the password is entered, a password confirmation menu will pop up so he can confirm the password that he chose (the password confirmation menu is explained below).

#### Password Confirmation Menu

ShowPasswordConfirmationMenu**(** iPlayerID**,** strPassword**[** **]** **)** **{**

**new** strMenuTitle**[** 128 **];**

formatex**(** strMenuTitle**,** charsmax**(** strMenuTitle **),** "\yPassword Confirmation Menu:^n^n\yIs this your password?^n\r%s"**,** strPassword **);**

**new** menuPasswordConfirmation **=** menu\_create**(** strMenuTitle**,** "Handle\_PasswordConfirmationMenu" **);**

menu\_additem**(** menuPasswordConfirmation**,** "\wYes"**,** strPassword **);**

menu\_additem**(** menuPasswordConfirmation**,** "\wNo"**,** "" **);**

menu\_setprop**(** menuPasswordConfirmation**,** MPROP\_NUMBER\_COLOR**,** "\y" **);**

menu\_setprop**(** menuPasswordConfirmation**,** MPROP\_EXIT**,** MEXIT\_NEVER **);**

menu\_display**(** iPlayerID**,** menuPasswordConfirmation**,** 0 **);**

**}**

public Handle\_PasswordConfirmationMenu**(** iPlayerID**,** iMenu**,** iKey **)** **{**

**new** strPassword**[** 64 **],** strAction**[** 32 **],** iAccess**,** iCallBack**;**

menu\_item\_getinfo**(** iMenu**,** iKey**,** iAccess**,** strPassword**,** charsmax**(** strPassword **),** strAction**,** charsmax**(** strAction **),** iCallBack **);**

**switch(** iKey **)** **{**

**case** 0**:** **{**

MySQL\_RegisterPlayer**(** iPlayerID**,** strPassword **);**

**}**

**case** 1**:** **{**

ShowPlayerRegistrationMenu**(** iPlayerID **);**

**}**

**}**

**}**

The password confirmation is a menu that lets the player double check if he typed his password correctly. If he chooses yes, that *‘MySQL\_RegisterPlayer( )’* function is called and the password is included as an argument. If the player chose no, then the *‘ShowRegistrationMenu( )’* function is called so the player can enter his password again.

#### Player Achievement Management

IncreasePlayerProgress**(** iPlayerID**,** iAchievementID**,** iProgress **=** 1 **)** **{**

**if(** **!**g\_bPlayerAchievementAcquired**[** iPlayerID **][** iAchievementID **]** **)** **{**

g\_iPlayerAchievementProgress**[** iPlayerID **][** iAchievementID **]** **+=** iProgress**;**

CheckGoalReached**(** iPlayerID**,** iAchievementID **);**

**}**

**}**

CheckGoalReached**(** iPlayerID**,** iAchievementID **)** **{**

**if(** **!**g\_bPlayerAchievementAcquired**[** iPlayerID **][** iAchievementID **]** **)** **{**

**if(** g\_iPlayerAchievementProgress**[** iPlayerID **][** iAchievementID **]** **>=** g\_iAchievementGoal**[** iAchievementID **]** **)** **{**

g\_iPlayerAchievementProgress**[** iPlayerID **][** iAchievementID **]** **=** g\_iAchievementGoal**[** iAchievementID **];**

g\_bPlayerAchievementAcquired**[** iPlayerID **][** iAchievementID **]** **=** **true;**

IncreasePlayerProgress**(** iPlayerID**,** ACHIEVEMENT\_AWARDIST **);**

**}**

**}**

**}**

These two functions handle the player’s achievement progress and goal. For example: to track the player’s kill count, we first hook when a player has died and check his killer, that means we must increase the killer’s Body Bagger Achievement progress (the achievement that requires 25 kills). To increase that achievement, all we have to do is call the *‘IncreasePlayerProgress( )’* function and pass to it the ID of the killer, the achievement ID that we would like to increase, and the optional third argument the amount of units the progress should increase. The third argument defaults to 1 since most of the achievements are incremented by 1 unit. After the player’s progress is increased, we must check if the player’s progress is equal to the achievement goal. If it is equal that means the player should acquire the achievement in question. The *‘CheckGoalReached( )’* will check for that as well as check if the player has acquired 10 or more achievements and if so increase the player’s Awardist progress (since Awardist is an achievement to acquire 10 achievements).

#### Achievement Tracking

Each achievement requires a different action. Therefore in order to check if the player accomplished a specific action, we must first hook or catch that action. For example: if we need to track how many kills a player has gotten, we must first catch when a player is killed and check his killer. The following code snippets explain how each action is hooked:

* This forward is to catch when a player is killed (knife kills, headshots, and normal deaths):

public Ham\_Killed\_Player\_Pre**(** iVictimID**,** iKillerID**,** iShouldGIB **)** **{**

static iWeaponID **=** get\_user\_weapon**(** iKillerID **);**

static strWeaponName**[** 32 **];**

get\_weaponname**(** iWeaponID**,** strWeaponName**,** charsmax**(** strWeaponName **)** **);**

**if(** equal**(** strWeaponName**,** "weapon\_knife" **)** **)** **{**

IncreasePlayerProgress**(** iKillerID**,** ACHIEVEMENT\_MAKE\_THE\_CUT **);**

**}**

**if(** get\_pdata\_int**(** iVictimID**,** m\_LastHitGroup**,** 5 **)** **==** HIT\_HEAD **)** **{**

IncreasePlayerProgress**(** iKillerID**,** ACHIEVEMENT\_BATTLE\_SIGHT\_ZERO **);**

**}**

IncreasePlayerProgress**(** iKillerID**,** ACHIEVEMENT\_BODY\_BAGGER **);**

**}**

* This forward is to catch when a player is taking damage:

public Ham\_TakeDamage\_Player\_Pre**(** iVictimID**,** iInflictorID**,** iAttackerID**,** Float**:**fDamage**,** iDamageBits **)** **{**

**if(** is\_user\_connected**(** iAttackerID **)** **)** **{**

IncreasePlayerProgress**(** iAttackerID**,** ACHIEVEMENT\_POINTS\_IN\_YOUR\_FAVOR**,** floatround**(** fDamage **)** **);**

**}**

**}**

* These two events are to catch when a specific team has won the round:

public Event\_SendAudio\_TerroristWin**(** **)** **{**

**new** iPlayers**[** 32 **],** iNum**,** iTempID**;**

get\_players**(** iPlayers**,** iNum**,** "ae"**,** "TERRORIST" **);**

**if(** iNum **==** 1 **)** **{**

IncreasePlayerProgress**(** iPlayers**[** 0 **],** ACHIEVEMENT\_CLUTCH\_MASTER **);**

**}**

**for(** **new** iLoop **=** 0**;** iLoop **<** iNum**;** iLoop**++** **)** **{**

iTempID **=** iPlayers**[** iLoop **];**

IncreasePlayerProgress**(** iTempID**,** ACHIEVEMENT\_PROMOTED **);**

**}**

**}**

public Event\_SendAudio\_CounterTerroristWin**(** **)** **{**

**new** iPlayers**[** 32 **],** iNum**,** iTempID**;**

get\_players**(** iPlayers**,** iNum**,** "ae"**,** "CT" **);**

**if(** iNum **==** 1 **)** **{**

IncreasePlayerProgress**(** iPlayers**[** 0 **],** ACHIEVEMENT\_CLUTCH\_MASTER **);**

**}**

**for(** **new** iLoop **=** 0**;** iLoop **<** iNum**;** iLoop**++** **)** **{**

iTempID **=** iPlayers**[** iLoop **];**

IncreasePlayerProgress**(** iTempID**,** ACHIEVEMENT\_PROMOTED **);**

**}**

**}**

* The following natives are to catch when the bomb has exploded or has been defused:

public bomb\_explode**(** iPlanterID**,** iDefuserID **)** **{**

IncreasePlayerProgress**(** iPlanterID**,** ACHIEVEMENT\_BOMB\_EXPERT **);**

**new** iPlayers**[** 32 **],** iNum**;**

get\_players**(** iPlayers**,** iNum**,** "ae"**,** "TERRORIST" **);**

**for(** **new** iLoop **=** 0**;** iLoop **<** iNum**;** iLoop**++** **)** **{**

IncreasePlayerProgress**(** iPlayers**[** iLoop **],** ACHIEVEMENT\_SOMEONE\_SET\_US\_BOMB **);**

**}**

**}**

public bomb\_defused**(** iDefuserID **)** **{**

IncreasePlayerProgress**(** iPlanterID**,** ACHIEVEMENT\_DEFUSAL\_EXPERT **);**

**new** iPlayers**[** 32 **],** iNum**;**

get\_players**(** iPlayers**,** iNum**,** "ae"**,** "CT" **);**

**for(** **new** iLoop **=** 0**;** iLoop **<** iNum**;** iLoop**++** **)** **{**

IncreasePlayerProgress**(** iPlayers**[** iLoop **],** ACHIEVEMENT\_RITE\_OF\_FIRST\_DEFUSAL **);**

**}**

**}**

# Chapter 5 – User Interface

In this chapter, we will be showing how the system works. How the player communicates with the plugin and website.

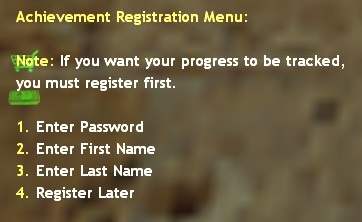
## Plugin

### Registration Message



When the player joins the server, the plugin checks if he is already registered or not. If not, the above chat message appears in his chat.

### Registration Menu



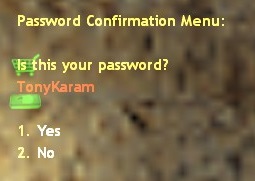
When the player types *‘/register’* in chat, the above menu will pop up. Then the player proceeds by entering his First Name, Last Name, and Password. The following image will show how a user would type his password.

### Password Entry



When the player chooses to enter his password (or First Name or Last Name), the plugin will automatically open a chat message dialog where the Player will input his desired text like shown above.

### Password Confirmation Menu



After the player has typed his password, a confirmation menu will pop up to confirm his chosen password like shown in the image above.

### Login Message



After the player has logged in, a message will be sent to the player’s chat notifying him of the result. Another message is also sent to remind the player that he needs to login after registering.

### Login Menu



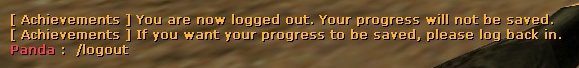
After the player has typed *‘/login’*, the above menu will pop up asking him to enter his password. Another chat message dialog will be automatically opened just like we already showed in the Password Entry section above.

### Successful Login



Another chat message is printed to the player notifying him of the login procedure. And then, since this is the first time the player has logged in, the plugin has inserting his new records in the Achieve table in the database, therefore the player is notified about it.

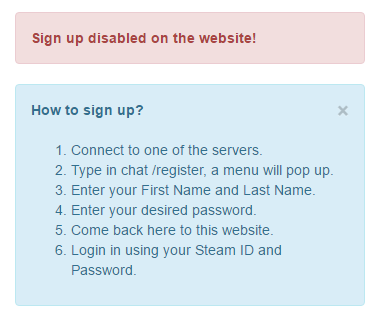
### Logout



The player has the ability to log out of his account. Therefore notify the player that he logged out, and tell him to log back in if he wishes his progress to be saved.

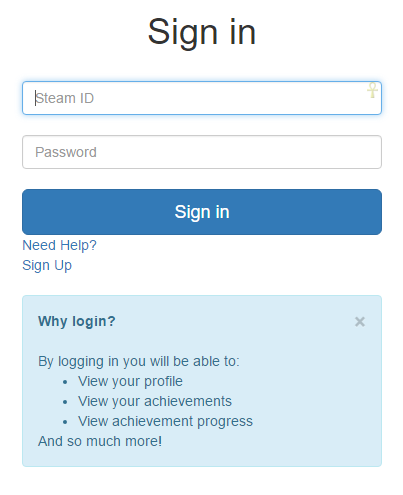
## Website

### Sign Up Page



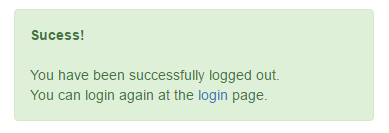
The image above shows how the sign up page looks like. Since we cannot retrieve the user’s steam ID to check, sign up has been disabled on the website. It is only allowed on the server by the plugin because on the server we can directly get the player’s steam ID.

### Login Page



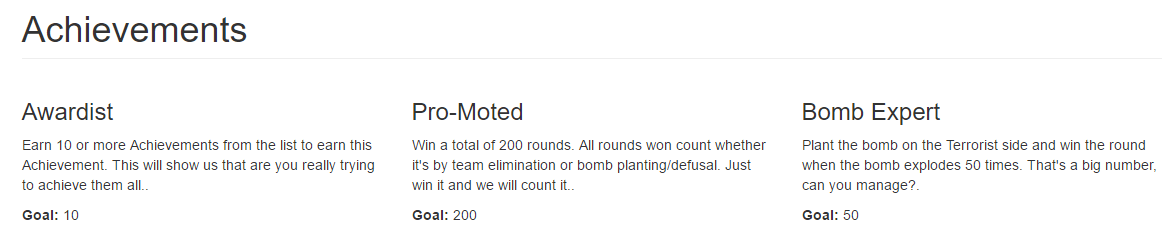
The sign in page is simple and easy. It allows the user to input his steam ID and password. It also shows a small information box that explains the benefits of signing in. Under the sign in button, there is small *‘Need Help?’* link that triggers a pop up that explain to the user how he can retrieve his steam ID if he does not know how to do so already.

### Logout Page



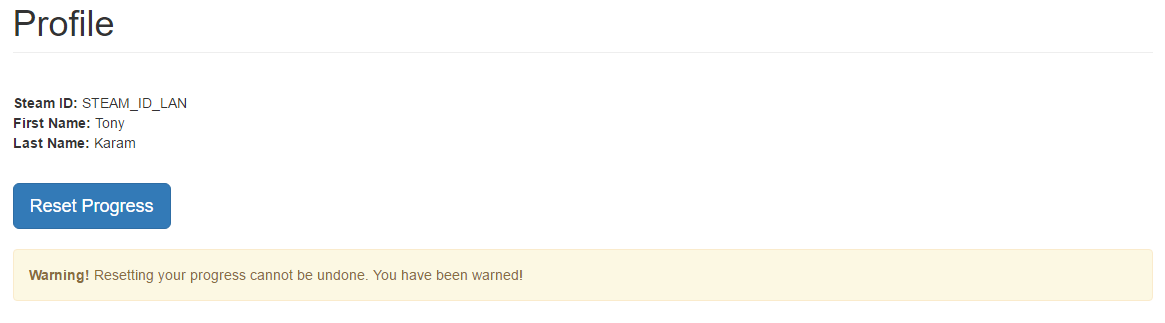
The logout page is a simple page that shows a success information box to notify the user that he has logged out simply by visiting the webpage.

### Achievements Page

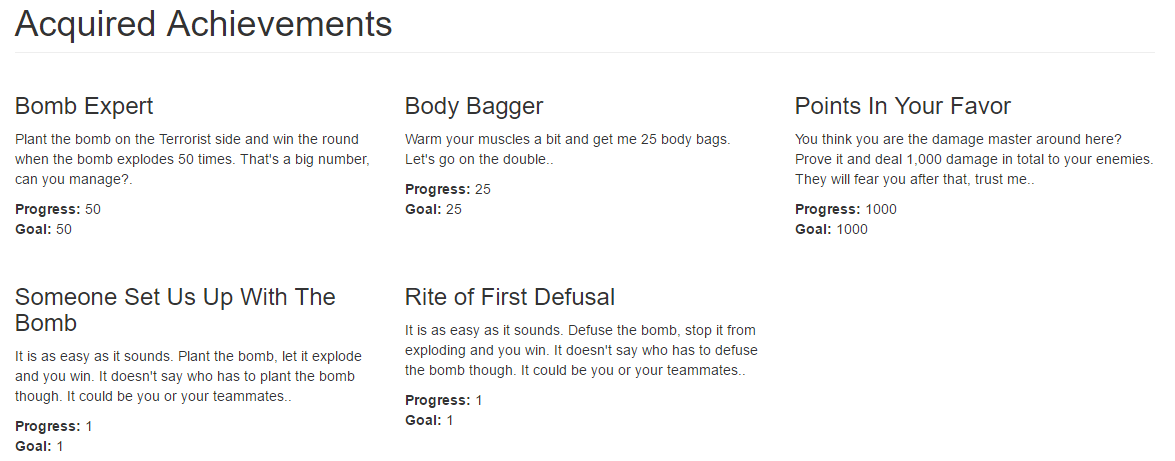


The achievements page is a page where all achievements are listed. Anyone can access this page, logged in and non-logged in users. There are a total of 5 rows like in the image shown above. Because the image would become so big if we included the entire set of achievements, only the first row is shown.

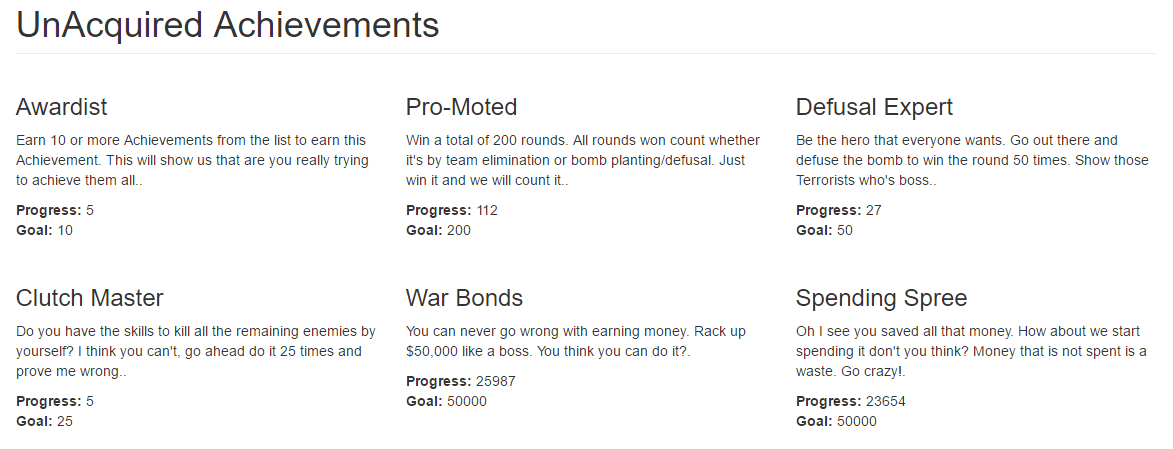
### Profile Page



The first part of the Profile page will display the user’s details such as the Steam ID, First Name, and Last Name. As well as a button to reset the user’s progress is he desires.



The second part is the list of achievements that the user has achieved.



The third and final part is the list of achievements that that the user has yet to achieve. In each achievement listing, the user can see his progress and the goal that he needs to hit in order to achieve the achievement in question.

# Chapter 6 – Results

## Results

### Automation

The end result is a fully automated system that can work with no human interaction at all. There is no need for administrators to keep an eye on the system, manually register players …etc. The only human interaction needed is at the very beginning where an administrator must add the achievements to the database. The administrator must add the Achievement’s name, ID, description and goal. The plugin will handle player registration, player achievement tracking, and player achievement status. Since Amx Mod X does not offer a good system to display all the achievements and progress for the players, the website is used to show each player the list of all achievements and his progress. This does not stop an administrator from making changes to the database or website if he desires.

### Security

When it comes to security, we mean by the measures taken to prevent hackers from stealing another player’s identity and possibly sabotage it. There are a few measures put in place to combat that. The first is the password hashing that is already explained in chapter 2. Password hashing with a randomly generated salt is there to prevent a hacker, if he managed to get access to the database, not to guess the player’s password. When it comes to steam IDs, they are unique to each player and they can never be changed. Steam ID management is handled by Valve and is out of my control.

### User Friendliness

The plugin and the website are very user friendly. If you have used a website before, that means you have all it takes to navigate through and use the website. The plugin is also user friendly, when a player joins the server it checks if he is registered or not. If he is not, a simple chat message is sent to the player explaining how to register. If he is already registered, he is sent a chat message telling that player to login. A player can also logout in game if he does not wish his progress to be tracked for the remainder of the game.

# References

Bourdon, R. (n.d.). Retrieved from WAMP Server: https://www.wampserver.com/

Ho, D. (n.d.). Retrieved from Notepad++: https://notepad-plus-plus.org/

Klinge, M. (n.d.). Retrieved from Bots United: http://podbotmm.bots-united.com/doc\_v3/html/pbmm\_credits.html

Vincent Herbert, S. E. (n.d.). Retrieved from Amx Mod X: http://www.amxmodx.org/about.php

# Appendix

## SQL Tables Creation

/\*

Player:

-------

@ SteamID is a string of at most 18 characters. It's format goes along the following:

- STEAM\_0:0:12345678

- STEAM\_0:1:87654321

@ FirstName is a normal string where the player inputs his first name.

@ LastName is a normal string where the player inputs his last name.

@ Password is a string of the hashed password of the player (it is not stored as plain text).

@ Salt is a random string of 16 characters to be used in the hashing of the password.

@ PRIMARY KEY of this table is the SteamID which is always unique

\*/

**CREATE** **TABLE** Player **(**

SteamID **varchar(** 18 **)** **NOT** **NULL,**

FirstName **varchar(** 255 **),**

LastName **varchar(** 255 **),**

Password **varchar(** 255 **),**

Salt **varchar(** 255 **),**

**PRIMARY** **KEY** **(** SteamID **)**

**)**

/\*

Achievement:

------------

@ ID is an integer that indentifies the achievement

@ Name is a string that gives a name to the achievement

@ Description is a string that explains a little bit about the achievement

@ Goal is the goal that the player has to reach to get the achievement

\*/

**CREATE** **TABLE** Achievement **(**

ID **int** **NOT** **NULL** AUTO\_INCREMENT**,**

Name **varchar(** 255 **),**

Description **varchar(** 512 **),**

Goal **int,**

**PRIMARY** **KEY** **(** ID **)**

**)**

/\*

Achieves:

---------

@ SteamID is the foreign key from Player

@ ID is the foreign key from Achievement

@ Progress is an integer number stating the progress of each Player with each Achievement

@ Acquired is a boolean variable that stores whether the player achieved each Achievement

\*/

**CREATE** **TABLE** Achieves **(**

SteamID **varchar(** 18 **),**

ID **int,**

Progress **int,**

Acquired **boolean,**

**FOREIGN** **KEY** **(** SteamID **)** **REFERENCES** Player **(** SteamID **),**

**FOREIGN** **KEY** **(** ID **)** **REFERENCES** Achievement **(** ID **),**

**PRIMARY** **KEY** **(** SteamID**,** ID **)**

**)**