Narrative

Software Reverse Engineering Artifact

Zane Russell Brown

Southern New Hampshire University

CS-499 Computer Science Capstone

GitHub Pages: https://github.com/BeardedArtist

Software Reverse Engineering

The software reverse engineering artifact is a project that was created in a software reverse engineering course. The goal of this project was to analyze assembly code, .out files and hex code and recreate the program as best as we can. The idea of this project was to give us real world experience when working with legacy systems and understanding the assembly code that makes up those programs and/or systems. The artifact includes three separate documents. The first is the assembly code analysis. Here I analyze the assembly code and discuss what each part of assembly does. The second document was created for this course and addresses the security vulnerabilities of the program and discusses any fixes that need to be made. This report will be included in the "enhancements made" section below. Finally, the final file included in this artifact is the c++ coding file that holds the program.

The software reverse engineering artifact was analyzed and gradually created during an eight-week period in the Software Reverse Engineering course. This course taught us how to use Linux systems to help understand and analyze the assembly code, hex code, and .out files. Once the analysis phases ended, we were taught how to take this knowledge of assembly and translate it to c++ language to recreate the program.

Justify the Inclusion of Artifact

As I have created several projects during my computer science journey, I struggled with choosing what project to showcase. However, the reverse engineering artifact quickly came to mind as it addresses a unique skill that I learned in my courses that could translate to a useful skill for future careers. Additionally, much like the Inventoy App artifact, I chose this project for my ePortfolio because it focuses on the major key computer science areas, software engineering and

design, and algorithms & data structures. The artifact relates to software engineering and design due to the meticulous analyzation of the code and understanding how it works both inside and out. Finally, this artifact relates to the algorithms and data structure area because we needed to fully understand the structures and algorithms of the assembly code and translate that understanding to the structures and algorithms of the c++ language.

I believe that this artifact is another strong entity in my ePortfolio. This is because it showcases the unique skills of reverse engineering that I have learned. This field of computer science is not the first thing people think of in regards to computer science. As our technology progresses, legacy systems gradually fall behind and businesses need to eventually fix and/or update these systems to keep up with the technological world. Software reverse engineering is a unique skill that can help these businesses and their technology. For this reason, I believed that including this artifact may showcase my unique skills and thus provide a unique portfolio for any future employers.

Reflection

Working on the improvements for the software reverse engineering artifact showcased how important reviewing and refactoring code is. During the original development of this artifact, I was still in the learning process and made several mistakes. However, now that I had some experience with coding and assembly language, I was able to see what problems the program had and fix them. Enhancing this project allowed me to see how much progress I had made in my skills as a programmer. Not only was I able to quickly identify the issues present in the program, but I also quickly fixed them while also staying as close to the assembly code as possible.

The program that was created from the assembly code is not too complex and therefore I had no issues re-creating this program during the enhancement phase. Reading assembly code was still slightly a challenge since assembly code doesn't always clearly represent what the program is doing. Regardless of this, I was able to easily re-create the program using both the assembly code and .out files.

This artifact went through some major changes. After re-analyzing the assembly code as well as re-analyzing my own program, I recorded the changes that needed to be made and updated the program. I did not to deviate too much from the original assembly code in order to showcase my skills in recreation. Therefore, I made certain decisions for this artifact. For example, I hardcoded the usernames and passwords instead of creating a database. I chose to do this because the user's were already recorded in the hex code. An informal review of the code and the enhancements made can be found below.

Artifact Issues and Improvements

A formal code review was not created in the second module for this artifact. Therefore, an informal code review was created for the original program that was made as well as documented enhancements. Most enhancements for this artifact have already been implemented during this week. The original assembly code documentation will also be included with this document and the c++ program file.

Block of C++ Code	Identified Issues

void CheckUserPermissionAccess()
{
 	
int userPassInput;
string usernameInput;
int Pass = 123;
 	
cout << "Enter your username: " << **endl**;
cin >> usernameInput;
cout << "Enter your password: " << **endl**;
cin >> userPassInput;
 	
if (userPassInput != Pass)
{
while (userPassInput != Pass) {
cout << "Invalid Password. Please try again" << **endl**;
cout << "Enter your username: " << **endl**;
cin >> usernameInput;
cout << "Enter your password: " << **endl**;
cin >> userPassInput;
}
 	
}
else
{
cout << "password accepted" << **endl**; // **This is used for testing purposes only. Not included in the Assembly code.**
}
 	
return;
 	
}
	The following highlighted sections of the **CheckUserPermissionAccess()** function contains security vulnerabilities. In fact, this function may contain the most important vulnerability within this program. The main issues here are that the program is not taking in any actual usernames. We can see the username's present in the hex when viewed through bless, but they are not used in the actual program. Also, the program only contains one password. Additional passwords will need to be created for each user. Note: I will hardcode in passwords in order to improve security, but to also keep the program as close to the assembly as possible.
	
	Additional vulnerabilities found. The next bug found was that the program did not allow for both names to be used due to the whitespace. Then, the program would let the user know that they made a mistake on the username/password the first time and then allow them to log into the program the second time even if the username/password was incorrect. Finally, an infinite display loop was found if the

	user enters the correct name the second time after failing the first log in. These will need to be addressed. **The first highlighted block:** - Add username variables to check for to increase security for the program - Also need to add passwords for each user. **The second highlighted block:** - The following cin inputs and if statements do not check for usernames. This means any user can easily access the program. - Only one password is being used for this program. This is highly dangerous and will need to be changed to increase security. - Both solutions can be done by adding a series of IF/ELSE statements to check both username and passwords - Fix program allowing only one name and not two. - Fix second time success error - Fix infinite loop issue.

void ChangeCustomerChoice()
{
int clientNumber;
int clientNewService;
 	
cout << "You chose 2" << **endl**;
cout << "Enter the number of the client that you wish to change" << **endl**;
cin >> clientNumber;
cout << "Please enter the client's new service choice (1 = Brokerage, 2 = Retirement)" << **endl**;
cin >> clientNewService;
 	
if (clientNumber == 1)
{
bobOption = clientNewService;
}
else if (clientNumber == 2)
{
sarahOption = clientNewService;
}
else if (clientNumber == 3)
{
	The following highlighted sections of the **ChangeCustomerChoice()** function contains security vulnerabilities. The user can change the customer's choice to be any integer instead of only 1 or 2. This can cause some serious issues if the company decides to implement a search filter into the program or if the user can receive any benefits for being in either option 1 or 2.
	
	The first highlighted block:
	- Character input can take in numbers more than 1 or 2. This can cause the customer's choice to become any number.

```
                amyOption = clientNewService;
        }
        else if (clientNumber == 4)
        {
                johnnyOption = clientNewService;
        }
        else if (clientNumber == 5)
        {
                carolOption = clientNewService;
        }

        return;
}
```

The second highlighted block:

- Should contain a filter (IF/ELSE) that checks for the ClientNewService variable for 1 or 2 only.

```
int main() {

        int userSelection;

        cout << "Hello! Welcome to our Investment Company" << endl;

        CheckUserPermissionAccess();

        while (userSelection != 3)
        {
            cout << "What would you like to do?" << endl;
            cout << "DISPLAY the client list (enter 1)" << endl;
            cout << "CHANGE a client's choice (enter 2)" << endl;
            cout << "Print Creators Name (enter 4)" << endl;
            cout << "Exit the program.. (enter 3)" << endl;
            cin >> userSelection;

            if (userSelection == 1)
            {
                    DisplayInfo();
            }
            else if (userSelection == 2)
            {
                    ChangeCustomerChoice();
            }
            else if (userSelection == 4)
            {
                    cout << "Zane Brown whom belongs to the Slytherin house made this." << endl;
            }
        }

        return 0;
}
```

The following highlighted sections of the **main()** function contains security vulnerabilities. Here, if the user enters an integer that goes beyond the max integer available in C++, it will cause an integer overflow and thus cause an infinite DisplayMenu() glitch. This is also caused if a float is entered.

We could add a simple filter that checks for only values 1-4. If none of these values are inserted, an error message is output. This could save us from user error and integer overflow.

Another security vulnerability that could exist in this program is the lack of additional username/password checks. If only certain employees should be able to change client's choices, then additional username/passwords might be necessary. Unfortunately, we do not currently have such

	information. Therefore this will be a record of such vulnerability and changed if said information is presented. **The first/second highlighted block:** - Can cause an integer overflow if user enters integer above c++ max int causing an infinite display menu bug. Also caused if float is entered.

Discussing Issues Found

CheckUserPermissionAccess() function

The following function contained security vulnerabilities. This was one of the most important functions that needed to be addressed to ensure the safety of the program. This function contained massive security vulnerabilities that pertained to the usernames and passwords. The original program seemed to only take one password which was “123”, this was obviously a major concern when it comes to passwords, but it also posed an issue due to the 5 other members sharing the same password. Additionally, the program did not take any kind of username. Anything could have been input into the program and no error messages would be output. It was obvious that this was a major vulnerability that needed to be fixed to help increase the security of the program. To fix this, I will hardcode in the current user’s username and corresponding passwords. I have decided to hardcode these values into the program at first in order to keep as close to the assembly that was analyzed as possible.

When testing the program, an additional vulnerability was found. It was discovered that if a user enters one of the available usernames (after implementing each username), the program

will only take the first name of the user's username. This is a big bug that needed to be addressed. Luckily a simple solution was found that will be discussed in the next section.

Finally, the last vulnerability found in this function was a bug that caused an infinite display loop when the user enters the correct username the second time. This means that if the user fails to log into the program the first time via a mistake with the username or password and attempts to log in a second time with the correct credentials, the program will enter an unstoppable loop that constantly displays the "error" message saying the user entered the incorrect username and password.

ChangeCustomerChoice() function

The following function contained security vulnerabilities. A major issue that was found in this function was that the user can change the customer's choice to be any integer instead of only 1 or 2. This is a big problem for a number of reasons. We can see that the function is focused around the customer's service choice which involves 1 equating to "Brokerage" and 2 equating to "Retirement". These are vital choices that the customer can make. If their option is accidentally changed or changed on purpose, it could cause problems in any benefits they may receive from being in that specific option. Also, this could cause the customer to be forgotten entirely if the entire program mainly uses a search/filter function. This specific vulnerability needs to be changed.

Quick addition, other end lines also need to be added to improve readability.

DisplayInfo()

No vulnerabilities were found in this function. However, additional end lines are needed to improve readability. Also, no comments were written in this function and will need to be updated.

Main() function

The following function contained security vulnerabilities. One specific vulnerability was an integer overflow if the user enters an integer that is higher (positive int) or lower (negative int) than the max integer that C++ can take. This caused a familiar bug in which the function would output an infinite DisplayMenu() function and would not stop. This made the program unusable. The same infinite display glitch was also caused when a float was entered into he system instead of a integer. Also, as a slight addition to this, if the user enters any value (not the max) outside of the values 1,2,3 or 4, the program will output the options again. This is great because this doesn't necessarily break the program and it is working as intended. However, an error message is not displayed. Thus, a user might become confused if the program continues to only output the menu without doing anything.

Another security vulnerability that could exist in this program is the lack of additional username/password checks. If only certain employees should be able to change client's choices, then additional username/passwords might be necessary. Unfortunately, we do not currently have such information. Therefore, this will be a record of such vulnerability and changed if said information is presented.

Fixes Made to the Program

CheckUserPermissionAccess() function

In the CheckUserPermissionAccess() function, we see a massive security vulnerability that pertained to the usernames and passwords. The original program seemed to only take one password which was “123” and no others. Additionally, the program did not take any kind of username. Anything could have been input into the program and no error messages would be output. To fix these vulnerabilities, I added additional variables that contain the current user’s usernames (found through bless) and gave those respective users their own passwords. These values are currently hardcoded in the program to maintain consistency with the analyzed assembly code.

When testing the program, an additional vulnerability was found. It was discovered that if a user enters one of the available usernames (after implementing each username), the program will only take the first name of the user’s username. To fix this specific vulnerability, I added a simple getline(cin, *variable*) which is used in place of the traditional cin statement. This is much better because cin statements only take the characters before the first white space. The getline() statement will take the entire string. Thus, allowing the user to enter their entire name.

Finally, the last vulnerability found in this function was a bug that caused an infinite display loop when the user enters the correct username the second time. This means that if the user fails to log into the program the first time via a mistake with the username or password and attempts to log in a second time with the correct credentials, the program will enter an unstoppable loop that constantly displays the “error” message saying the user entered the incorrect username and password. To fix this, a new bool variable was created (i.e.

CorrectPassword) and was set to false. A while loop can hold the IF/ELSE statements that check the username and passwords. This while loop contains the condition “CorrectPassword == false”. Then, inside the IF/ELSE statements, if the user enters the username and password correctly, the bool variable will be set to true and break the loop. Else, the program will output an error message and allow the users to try again.

ChangeCustomerChoice() function

In the ChangeCustomerChoice() function, a major issue that was found in this function was that the user can change the customer’s choice to be any integer instead of only 1 or 2. This was also be fixed with simple IF/ELSE statements. To enter only 1 or 2, we encased the current IF/ELSE IF statements that check which customer was selected and gave the if statement a condition that states if new client service is equal to 1 or is equal to 2, then the inside IF/ELSE IF statements may be executed. Else an error message is output letting the user know that their service choice value was not valid.

Quick addition, other end lines also need to be added to improve readability. To fix this, just simply and endl to add extra lines.

Main() function vulnerabilities

In the main() function, one specific vulnerability that was found was an integer overflow when the user enters an integer that is higher (positive int) or lower (negative int) than the max integer that C++ can take or a float. When this happens an infinite display glitch occurs. To fix this, a while loop encases the “choices” strings as well as the IF/ELSE IF statements that check

to see what the user chose. The while loop takes the condition that checks to see if the user's selection is NOT equal to 3. If not equal the loop executes. Else it will exit. Also, the IF/ELSE IF statements are encased in an if statement that checks to see if the user entered 1,2,3 or 4. If the values match, the inside IF/ELSE IF statements are executed. Else, an error message outputs letting the user know that their choice was invalid and to try again. The "choices" strings are then be output again. This error message also solves another vulnerability we found which causes the display to output as many times as it needs to if the user enters the wrong value. The error message lets them know that their input was wrong and to enter only 1,2,3 or 4.

ORIGINAL Program

```
//============================================================================
// Name        : Practice.cpp
// Author      : Zane Brown
//============================================================================

#include <iostream>
#include <string>
using namespace std;

int bobOption = 1;
int sarahOption = 2;
int amyOption = 1;
int johnnyOption = 1;
int carolOption = 2;


void CheckUserPermissionAccess()
{
	// Add the following username variables to check for to increase security for the program
	// Also need to add passwords for each user.

	// string user1 = "Bob Jones";
	// string user2 = "Sarah Davis";
	// string user3 = "Amy Friendly";
	// string user4 = "Johnny Smith";
	// string user5 = "Carol Spears";
```

```
	int userPassInput;
	string usernameInput;
	int Pass = 123;
	// more passwords needed to improve security

	cout << "Enter your username: " << endl;
	cin >> usernameInput;
	cout << "Enter your password: " << endl;
	cin >> userPassInput;

	// The following cin inputs and if statements do not check for usernames. This means any user can easily access the program.
	// Only one password is being used for this program. This is highly dangerous and will need to be changed to increase security.
	// Both solutions can be done by adding a series of IF/ELSE statements to check both username and passwords
	// Both names are not taken for user's full name. Will need to add functionality to include both names

	if (userPassInput != Pass)				// Need to update the IF statement to take username/password conditions and ELSE to take while loop.
	{					// While loop can accidentally cause an infinite loop when inputing correct name SECOND time after first mistake.
		while (userPassInput != Pass) {
			cout << "Invalid Password. Please try again" << endl;
			cout << "Enter your username: " << endl;
			cin >> usernameInput;
			cout << "Enter your password: " << endl;
			cin >> userPassInput;
		}

	}
	else
	{
		cout << "password accepted" << endl;  // **This is used for testing purposes only. Not included in the Assembly code.**
	}

	return;

}

// If this program is only accessed by employees, then the ChangeCustomerChoice() function is fine.
// Else if this program is also used by customers, this is very dangerous to allow them to change any customer's choice.

void ChangeCustomerChoice()
```

```
{
	int clientNumber;
	int clientNewService;

	cout << "You chose 2" << endl;
	cout << "Enter the number of the client that you wish to change" << endl;
	cin >> clientNumber;
	cout << "Please enter the client's new service choice (1 = Brokerage, 2 = Retirement)" << endl;
	cin >> clientNewService; // Can enter a number more than 1 or 2. This will cause the customer choice to change to any number.

	if (clientNumber == 1)  // Needs an IF/ELSE statement encasing the following IF/ELSE statements in order to check for selection of 1 or 2.
	{
		bobOption = clientNewService;
	}
	else if (clientNumber == 2)
	{
		sarahOption = clientNewService;
	}
	else if (clientNumber == 3)
	{
		amyOption = clientNewService;
	}
	else if (clientNumber == 4)
	{
		johnnyOption = clientNewService;
	}
	else if (clientNumber == 5)
	{
		carolOption = clientNewService;
	}

	return;
}

void DisplayInfo()
{
	cout << "You chose 1" << endl;

	cout << "   Client's Name   Service Selected (1 = Brokerage, 2 = Retirement)" << endl;
	cout << "1. Bob Jones selected option " << bobOption << endl;
	cout << "2. Sarah Davis selected option " << sarahOption << endl;
	cout << "3. Amy Friendly selected option " << amyOption << endl;
	cout << "4. Johnny Smith selected option " << johnnyOption << endl;
```

```
        cout << "5. Carol Spears selected option " << carolOption << endl;
    // Needs an extra end line to improve readability
        return;

}


int main() {

        int userSelection; // Can cause an integer overflow if user enters integer above C++ max int causing an infinite display menu bug.
                // This is also caused if a float is entered.
        cout << "Hello! Welcome to our Investment Company" << endl;

        CheckUserPermissionAccess();

        while (userSelection != 3)
        {
                cout << "What would you like to do?" << endl;
                cout << "DISPLAY the client list (enter 1)" << endl;
                cout << "CHANGE a client's choice (enter 2)" << endl;
                cout << "Print Creators Name (enter 4)" << endl;
                cout << "Exit the program.. (enter 3)" << endl;
                cin >> userSelection;
                                // Filter needed to check for values 1,2,3 or 4. Else error message needs to be displayed.
                if (userSelection == 1)
                {
                        DisplayInfo();
                }
                else if (userSelection == 2)
                {
                        ChangeCustomerChoice();
                }
                else if (userSelection == 4)
                {
                        cout << "Zane Brown whom belongs to the Slytherin house made this." << endl;
                }
        }

        return 0;
}
```

NEW Program (With Enhancements)

```
//============================================================================
// Name        : ProjectTwo.cpp
// Author      : Zane Brown
// Description : Project Two
//============================================================================

#include <iostream>
#include <string>
#include <limits>
using namespace std;

int bobOption = 1;
int sarahOption = 2;
int amyOption = 1;
int johnnyOption = 1;
int carolOption = 2;


void CheckUserPermissionAccess()
{

	int userPassInput;
	string usernameInput;
	bool CorrectPassword = false;


	// Add username variables to check for to increase security for the program
	// Also need to add passwords for each user.
	// (**FIXED -- Added usernames for each user as well as passwords for the respective user. This will increase this function's security.**)

	string user1 = "Bob Jones";
	string user2 = "Sarah Davis";
	string user3 = "Amy Friendly";
	string user4 = "Johnny Smith";
	string user5 = "Carol Spears";

	int BobPass = 9874;
	int SarahPass = 3478;
	int AmyPass = 2991;
	int JohnnyPass = 5503;
	int CarolPass = 2814;
```

```
	cout << "Enter your username: " << endl;
	getline(cin, usernameInput); // (**FIXED -- Added getline() to take in user's full name instead of only one word.**)
	cout << "Enter your password: " << endl;
	cin >> userPassInput;

	// The following cin inputs and if statements do not check for usernames. This means any user can easily access the program.
	// Only one password is being used for this program. This is highly dangerous and will need to be changed to increase security.
	// Both solutions can be done by adding a series of IF/ELSE statements to check both username and passwords

	while (CorrectPassword == false) //(**FIXED -- Added while loop to entire if/else block to manage if password matches TRUE. Else to continue loop if false.**)
	{
		if (usernameInput == user1 && userPassInput == BobPass)
			{
				cout << "password accepted" << endl;
				cout << endl;
				CorrectPassword = true;
			}
			else if (usernameInput == user2 && userPassInput == SarahPass)
			{
				cout << "password accepted" << endl;
				cout << endl;
				CorrectPassword = true;
			}
			else if (usernameInput == user3 && userPassInput == AmyPass)
			{
				cout << "password accepted" << endl;
				cout << endl;
				CorrectPassword = true;
			}
			else if (usernameInput == user4 && userPassInput == JohnnyPass)
			{
				cout << "password accepted" << endl;
				cout << endl;
				CorrectPassword = true;
			}
			else if (usernameInput == user5 && userPassInput == CarolPass)
			{
				cout << "password accepted" << endl;
				cout << endl;
				CorrectPassword = true;
			}
```

```
                else //(**FIXED -- While loop removed in ELSE due to infinite loop errors being caused when taking in multiple username and passwords.**)
                {
                        cin.clear();
                        cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Line 66 and 67 are used to clear the character input. If not done, an infinite loop of the below output happens.
                        cout << "Invalid Username or Password. Please try again" << endl;
                        cout << "Enter your username: " << endl;
                        getline(cin, usernameInput);
                        cout << "Enter your password: " << endl;
                        cin >> userPassInput;
                }
        }
        return;
}

// If this program is only accessed by employees, then the ChangeCustomerChoice() function is fine.
// Else if this program is also used by customers, this is very dangerous to allow them to change any customer's choice.

void ChangeCustomerChoice()
{
        int clientNumber;
        int clientNewService;

        cout << "You chose 2" << endl;
        cout << "Enter the number of the client that you wish to change" << endl;
        cin >> clientNumber;
        cout << "Please enter the client's new service choice (1 = Brokerage, 2 = Retirement)" << endl;
        cin >> clientNewService; // Can enter a number more than 1 or 2. This will cause the customer choice to change to any number.
                //(**FIXED -- encased the IF/ELSE statements for changing service within another IF/ELSE statement checking if only 1 or 2 was chosen.**)
        cout << endl;           //(**FIXED -- added endline to improve readability**)

        if (clientNewService == 1 || clientNewService == 2) // This will check if user only enters 1 or 2. Else it will output error message.
        {
                if (clientNumber == 1)
                {
                        bobOption = clientNewService;
                }
                else if (clientNumber == 2)
                {
                        sarahOption = clientNewService;
                }
```

```
                else if (clientNumber == 3)
                {
                        amyOption = clientNewService;
                }
                else if (clientNumber == 4)
                {
                        johnnyOption = clientNewService;
                }
                else if (clientNumber == 5)
                {
                        carolOption = clientNewService;
                }
        }
        else
        {
                cout << "**Invalid service choice. Please choose 1 or 2.**" << endl; //Error message if service choice does not match 1 or 2.
                cout << endl;
        }

        return;
}


void DisplayInfo()
{
        cout << "You chose 1" << endl;

        cout << "   Client's Name   Service Selected (1 = Brokerage, 2 = Retirement)" << endl;
        cout << "1. Bob Jones selected option " << bobOption << endl;
        cout << "2. Sarah Davis selected option " << sarahOption << endl;
        cout << "3. Amy Friendly selected option " << amyOption << endl;
        cout << "4. Johnny Smith selected option " << johnnyOption << endl;
        cout << "5. Carol Spears selected option " << carolOption << endl;
        cout << endl; //(**FIXED-- added endline to improve readability.**)

        return;

}


int main() {

        float userSelection; // Can cause an integer overflow if user enters integer above C++ max int causing an infinite display menu bug.
```

```
                // This is also caused if a float is entered.
                    //(**FIXED -- IF/ELSE now checking for specifically 1,2,3 or 4.**)
                    //(**Changed to Float to stop infinite display loop if float input.**)

        cout << "Hello! Welcome to our Investment Company" << endl;

        CheckUserPermissionAccess();

        while (userSelection != 3)
        {
                cout << "What would you like to do?" << endl;
                cout << "DISPLAY the client list (enter 1)" << endl;
                cout << "CHANGE a client's choice (enter 2)" << endl;
                cout << "Print Creators Name (enter 4)" << endl;
                cout << "Exit the program.. (enter 3)" << endl;
                cin >> userSelection;

                if (userSelection == 1 || userSelection == 2 || userSelection == 3 || userSelection == 4) //(**FIXED --
Program now checks for only 1,2,3 or 4. Else error is output.**)
                {
                        if (userSelection == 1)
                        {
                                DisplayInfo();
                        }
                        else if (userSelection == 2)
                        {
                                ChangeCustomerChoice();
                        }
                        else if (userSelection == 4)
                        {
                                cout << "Zane Brown whom belongs to the Slytherin house made this." << endl;
                        }
                }
                else
                {
                        cout << "User entered invalid choice. Please choose 1, 2, 3 or 4." << endl;
                        break; //(**FIXED -- added break to stop infinite display loop.**)
                }

        }

        return 0;
}
```