

Final Report



I. Introduction

In a first-meeting event for community members, an orientation session should be prepared and presented by the host. The host wants to make this session interactive, real-time, and systematic. **icebreaQ** makes this possible.

- **Interactive**
 - Participants can answer quiz questions and video chat with other participants
- **Real-time**
 - Answers are visible to everyone in the session.
- **Systematic**
 - Once created, the quiz can be reused for similar occasions.

iceBreaQ is a web-based real-time quiz platform for ice breaking. Take some time to share your thoughts and get close to each other through a quiz consisting of various questions.

II. Overview

1. Project Repository URL

Frontend: <https://github.com/Team-ILA/icebreaQ-front.git>

Backend: <https://github.com/Team-ILA/icebreaQ-server.git>

2. Functionalities

- a. Users can register and log in
- b. Registered users can create a new quiz instance and generate an associated url
- c. Registered users can join a quiz instance with the proper url

3. Open source projects

- a. Express.js
- b. React.js
- c. Node-gyp
- d. Node-bindings

III. Implementation

1. Title Page

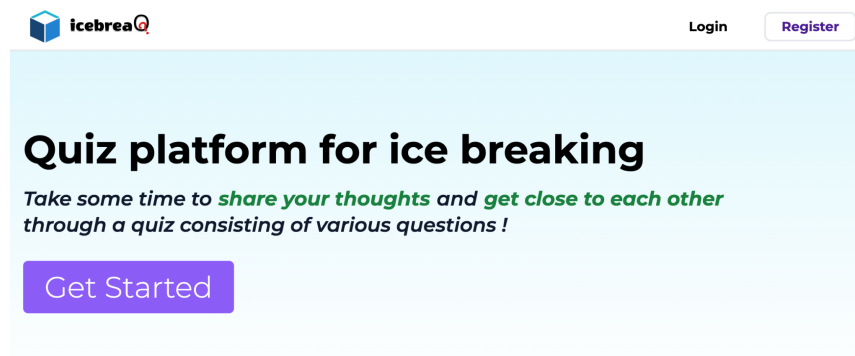


Fig.1. Title Page

This page provides a brief description of **icebreaQ**'s features and acts as a main page that helps users navigate to their destinations. The navigation bar located at the top of the page provides two buttons. 'Login' redirects the browser to the login page, while 'Register' redirects the browser to the registration page. Below the navigation bar is the 'Get Started' button that redirects the browser to the 'Quiz Join' Page.

2. User Login Page

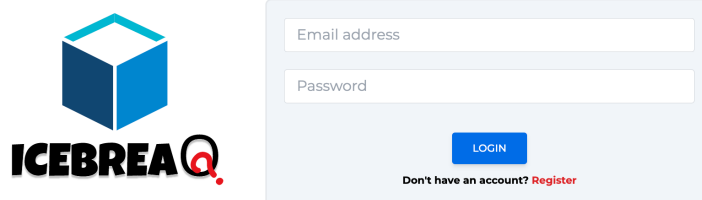


Fig.2. Login Page

Clicking on the 'Login' button at the navigation bar or 'Get Started' button without logging in will bring the user to this page. In this page the user can login to the service by typing their email address and password to login or register to the service. The user is required to login if they want to enjoy all features. The service provides format checking for both email and password to enhance security of the authentication process. Clicking on the 'Register' link below the 'LOGIN' button will redirect the user to the 'User Register' page.

3. User Register Page

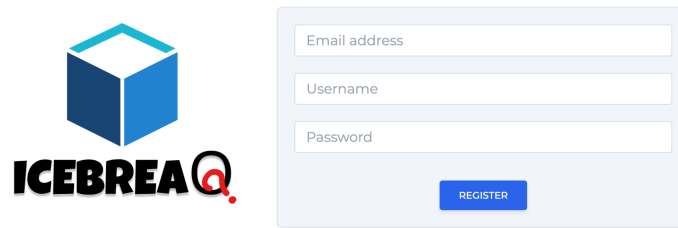
The image shows the user registration page for IcebreaQ. On the left is the IcebreaQ logo, which consists of a blue 3D cube icon above the text "ICEBREAQ" in a bold, black, sans-serif font. On the right is a light blue rectangular form. Inside the form, there are three white input fields stacked vertically, labeled "Email address", "Username", and "Password". Below these fields is a blue button with the word "REGISTER" in white capital letters.

Fig.3. Register Page

The user can register to the service by providing their email address, username and password. Format checking is performed on all input values when a create user request is sent to the server.

4. Quiz Join Page

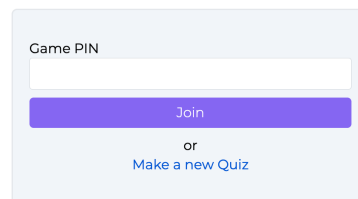
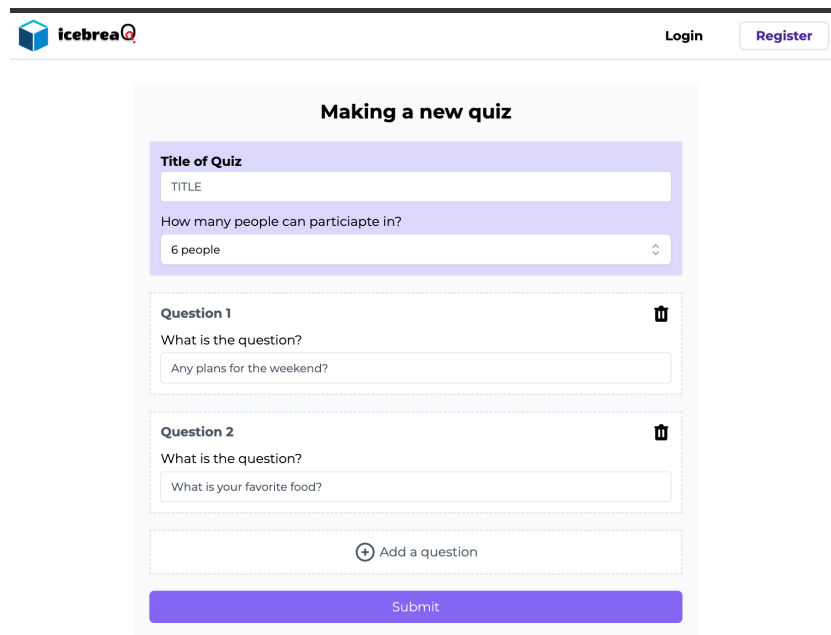
This image shows the header section of the Quiz Join Page. It features a horizontal line. On the left side of the line is the IcebreaQ logo. On the right side, there are two links: "Login" and "Register", both in a small, blue, sans-serif font.The image shows the main content area of the Quiz Join Page. It contains a light blue rectangular form. Inside the form, there is a white input field labeled "Game PIN". Below the input field is a blue button with the word "Join" in white capital letters. Below the button, the word "or" is centered, followed by the text "Make a new Quiz" in a blue, sans-serif font.

Fig.4. Quiz Join Page

The user can either join a quiz instance by typing the appropriate quiz PIN, or make a new quiz. If the user types an invalid PIN, an error message appears below the button. If the user types a valid PIN, the browser redirects to the quiz page of the appropriate PIN. The user's browser redirects to the 'Create Quiz' Page by clicking the 'Make a new Quiz' button.

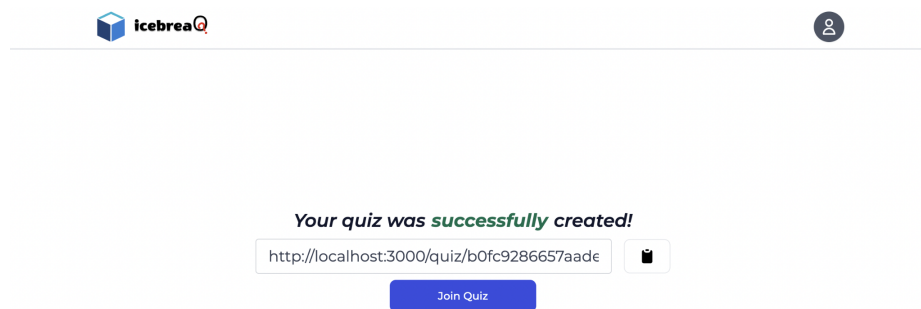
5. Create Quiz (Success) Page



The screenshot shows the 'Making a new quiz' form in the icebreaQ application. The form is titled 'Making a new quiz' and contains the following fields and buttons:

- Title of Quiz:** A text input field with the placeholder 'TITLE'.
- How many people can participate in?:** A dropdown menu with '6 people' selected.
- Question 1:** A text input field with the placeholder 'What is the question?' and the example text 'Any plans for the weekend?'. A trash icon is visible to the right.
- Question 2:** A text input field with the placeholder 'What is the question?' and the example text 'What is your favorite food?'. A trash icon is visible to the right.
- Add a question:** A button with a plus icon and the text 'Add a question'.
- Submit:** A large blue button with the text 'Submit'.

Fig.5. Create Quiz Page



The screenshot shows the success message 'Your quiz was successfully created!' in the icebreaQ application. The message is displayed in a green font. Below the message, there is a text box containing the URL 'http://localhost:3000/quiz/b0fc9286657aade' and a trash icon to its right. Below the text box is a blue button with the text 'Join Quiz'.

Fig.6. Create Quiz Success Page

In this page the user can create a new quiz instance. The user should type the quiz's title and select the total number of participants. The user can add at least one question for the quiz instance. If the user clicks the 'Submit' button, the browser requests the server to make a quiz instance which will be stored in the server's database. If the server successfully creates a new quiz instance in the database, it returns the unique PIN value for the created quiz. The user can easily join the Quiz room by clicking on the 'Join Quiz' button, or by using the url provided in the box at the middle of the page.

6. Quiz Page

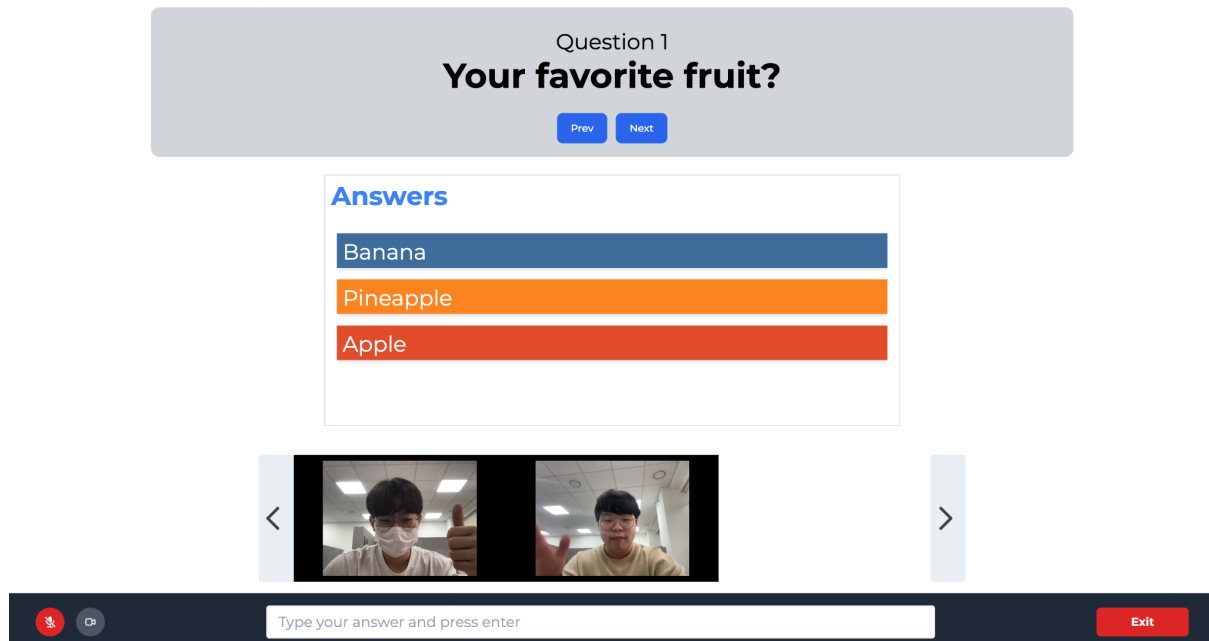


Fig.7. Quiz Page

In this page the user can see the current question and its answers by other participants in the quiz room. The user can add a new answer by typing their answer into the input box located at the bottom middle of the page and pressing enter. The user can also join a voice & video room by toggling the microphone and camera button at the bottom left corner of the page. The user should grant video and microphone control permission to the browser to join the voice & video room. The user can exit from the page by pressing the ‘Exit’ button at the bottom right corner of the page. The creator of the quiz instance can control the ‘Prev’ and ‘Next’ button to shift between questions.

IV. Project Demo

Video clip: <https://www.youtube.com/watch?v=X4PaFwVboRs>

V. Factors of Distinction

In this section, we examine what differentiates **icebreaQ** with other quiz platforms through empirical tests and user experience analysis. We compare **icebreaQ** with two other platforms(Kahoot!, QuizN).

Kahoot! is the most popular quiz platform in the world. It targets mostly educational use, supports various question types, and has multiple quiz modes. Although widespread and versatile, **Kahoot!** inherits some major shortcomings compared to **icebreaQ**. **Kahoot!**’s various features makes quiz building tiresome. Also, joining a quiz through **Kahoot!** takes much longer, even though **icebreaQ** and **Kahoot!** both use the same mechanisms for quiz joining. Lastly, **Kahoot!** lacks the means for participant communication. There is no real-time video chat or message chatting feature.

QuizN is a Korean variant of **Kahoot!**, which is mostly similar to it. Thus **QuizN** inherits the same shortcomings of **Kahoot!**.

1. Joining a Quiz

Joining a quiz on the other two platforms takes a long time. We measure the time to join a quiz in seconds on each platform and compare the results.

Platform	Time (seconds)
icebreaQ	<3
Kahoot!	10
QuizN	10

Table1. Quiz join time

Joining a quiz in other platforms takes significantly longer than **icebreaQ**. This would lead to bad user experience. **icebreaQ** can provide better experience with faster loading times.

2. Making a Quiz

Making a quiz on the other two platforms is very exhaustive. We measure the time to create a quiz consisting of one question in seconds on each platform and compare the results.

Platform	Time (seconds)
icebreaQ	<10
Kahoot!	35
QuizN	35

Table2. Quiz creation time

Making a quiz in other platforms also takes significantly longer than **icebreaQ**. This also leads to bad user experience. **icebreaQ** can provide better experience with faster quiz creation.

3. User Communication

As aforementioned, **icebreaQ** supports live video and text chatting. The other platforms lack these features. We believe that this is due to the different target users of other platforms. Most educational quizzes are held in offline events; participants meet up with each other, thus there is no need for online communication. However, many recent employee meetups are held online due to the pandemic. Thus, they need means to communicate online. Only **icebreaQ** can provide these features, which differentiates it from other platforms.

VI. GitCTF - Intended Vulnerabilities

1. Buffer Overflow (1)

```
bool validate(const char *str)
{
    // define a regular expression
    const regex pattern
```

```

        ("(\\w+)(\\.|_)?(\\w*)@(\\w+)(\\. (\\w+))+");

char buffer[BUF_SIZE];
strcpy(buffer, str);

string email = buffer;

// try to match the string with the regular expression
return regex_match(email, pattern);
}

```

Fig.8. validate() in emailValidator.cc

icebreaQ-server/src/lib/email-validator/emailValidator.cc contains code vulnerable to buffer overflow attack. validate() copies the input char *str to a buffer of size 100 bytes. The attacker can exploit this vulnerability by inserting a string that exceeds 100 bytes to this function. This kind of attack can be used to overwrite the stack frame, which can lead to unintended instruction executions, or disclosure of internal variables.

```

bool validate(const char *str)
{
    // define a regular expression
    const regex pattern
        ("(\\w+)(\\.|_)?(\\w*)@(\\w+)(\\. (\\w+))+");

    char buffer[BUF_SIZE];
    strncpy(buffer, str, BUF_SIZE);

    string email = buffer;

    // try to match the string with the regular expression
    return regex_match(email, pattern);
}

```

Fig.9. Fixed validate() in emailValidator.cc
(Commit ID: 6f58c3eb943c2221e76dd6cf03c6790ef12bb4b6)

This vulnerability can be resolved by replacing strcpy() with strncpy(), which prevents the program from copying more than 100 bytes to the buffer. strncpy() copies only up to 100 bytes to the buffer regardless of the input length.

2. Buffer Overflow (2)

```

bool validate(const char *str)
{
    char buffer[BUF_SIZE];
    strcpy(buffer, str);

    string password = buffer;
    int n = password.length();
    if(n < 11) return false;
    bool hasLower = false, hasUpper = false, hasDigit = false, hasSpecial = false;

```

```

for(int i = 0; i < n; i++)
{
    if(islower(password[i]))
        hasLower = true;
    else if(isupper(password[i]))
        hasUpper = true;
    else if(isdigit(password[i]))
        hasDigit = true;
    else
        hasSpecial = true;
}

if(hasLower && hasUpper && hasDigit && hasSpecial)
    return true;
else
    return false;
}

```

Fig.10. validate() in passwordValidator.cc

icebreaQ-server/src/lib/password-validator/passwordValidator.cc also contains code vulnerable to buffer overflow attack. validate() copies the input char *str to a buffer of size 50 bytes. The attacker can exploit this vulnerability by inserting a string that exceeds 50 bytes to this function. Possible consequences of an attack are similar to 1.

```

bool validate(const char *str)
{
    char buffer[BUF_SIZE];
    strncpy(buffer, str, BUF_SIZE);

    string password = buffer;
    int n = password.length();
    if(n < 11) return false;
    bool hasLower = false, hasUpper = false, hasDigit = false, hasSpecial = false;

    for(int i = 0; i < n; i++)
    {
        if(islower(password[i]))
            hasLower = true;
        else if(isupper(password[i]))
            hasUpper = true;
        else if(isdigit(password[i]))
            hasDigit = true;
        else
            hasSpecial = true;
    }

    if(hasLower && hasUpper && hasDigit && hasSpecial)
        return true;
    else
        return false;
}

```

Fig.11. Fixed validate() in passwordValidator.cc

(Commit ID: 3652e79dc12b08c4b7284d35341316ca65d495ce)

This vulnerability can be resolved in the same manner as the first one. `strcpy()` can be replaced with `strncpy()`.

```
int logger(const string& log_level, const string& message)
{
    ...
    FILE* fptr = fopen(file.c_str(), "a");
    if(fptr != NULL)
    {
        fprintf(fptr, dateFormat.c_str());
        fprintf(fptr, log_level.c_str());
        fprintf(fptr, "::");
        fprintf(fptr, message.c_str());
        fprintf(fptr, "\n");

        fclose(fptr);
    }
    ...
}
```

Fig.12. `logger()` in `logger.cc`

3. Race Condition (TOCTOU)

`icebreaQ-server/src/lib/logger/logger.cc` contains code that can cause a race condition. More specifically, the program incurs a time-of-check to time-of-use (TOCTOU) race condition in execution. The attacker can replace the opened file (`.log`) to induce an unintended file write. This can be observed in `logger()`, which opens a file to do a file write operation. This is not done atomically, which results in a TOCTOU race condition.

This vulnerability can be resolved by exploiting `flock(2)`, which is a Linux system call that applies or removes a lock on an open file. `flock(2)` locks the file so that our program can exclusively manipulate or read the file, which doesn't allow the attacker to change the file's contents or file pointer itself.

4. Format String

`icebreaQ-server/src/lib/logger/logger.cc` contains code vulnerable to format string attack. `logger()` takes two strings (`log_level`, `message`) as input, which are directly passed to `fprintf()` to execute a file write operation. By passing a format string (e.g. `%p`) as an argument, the attacker can exploit this vulnerability. This vulnerability can be exploited to obtain memory locations, which is fatal since the attacker can extract crucial values such as `%ebp` or return addresses.

This vulnerability can be resolved by providing more arguments to each `fprintf()` so that it doesn't interpret the input string. The input string will always be printed intact, although it contains a format string.

```
int logger(const string& log_level, const string& message)
{
    ...
    FILE* fptr = fopen(file.c_str(), "a");
    if(fptr != NULL)
    {
```

```

    int fd = fileno(fptr);
    if(flock(fd, 2))
    {
        fprintf(fptr, "%s", dateFormat.c_str());
        fprintf(fptr, "%s", log_level.c_str());
        fprintf(fptr, "::");
        fprintf(fptr, "%s", message.c_str());
        fprintf(fptr, "\n");
    }

    flock(fd, 8);

    fclose(fptr);
}
...
}

```

Fig.13. Fixed logger() in logger.cc

(Commit ID: 54d616436dca5ef9bcd97ecef1c5a777ed369664)

(Commit ID: 1a3690c5755aed6c39e4bb43eff74bcd36fad425)

5. Information Disclosure (reference: CVE-2022-32778)

```

...
app.use(
  session({
    secret: SESSION_KEY,
    resave: false,
    saveUninitialized: false,
  }),
);
...

```

Fig.14. index.ts

icebreaQ utilizes the http/https protocol as its server host and web client's communication protocol. However, the http protocol inherits a major security problem. The application's backend should maintain session states of the current connected users. This is done through session cookies. **icebreaQ**'s server transmits the generated session cookies to clients through the http protocol configured by default options. However, the 'secure flag' option is disabled by default. This means the session cookies are vulnerable to hijacking attacks: they are accessible by the attacker via JavaScript crafted http requests. This vulnerability can be specified as the information disclosure vulnerability.

This vulnerability can be resolved by simply adding the secure flag option. The server will now only allow secure connections, thus refusing hijacking attack requests.

```

...
app.use(
  session({
    secret: SESSION_KEY,
    resave: false,
    saveUninitialized: false,
    secure: process.env.NODE_ENV === DEPLOYMENT ? true : false,
  })
);
...

```

```

    }},
  );
  ...

```

Fig.14. Fixed index.ts

(Commit ID: bb21fb5c8e36f130cf2e486b479850f3a6de68d0)

VII. GitCTF - Unintended Vulnerabilities

No unintended vulnerabilities were found in our project during the GitCTF.

VIII. GitCTF - Reported Vulnerabilities

1. Team2

1.1 views.py - TOCTOU (Unintended)

(Commit ID: 1e10368c36390c1847ac1a169f558fca275731b3)

While writing into a file, an attacker can link this file to another file which leads to TOCTOU

2. Team3

2.1 mainwindow.cpp - SQL Injection (Intended)

(Commit ID: 19a3ad6def99280a7a6776c06f63c4d1387fb455)

As intended, the SQL statement searches a user with input ID and PW. However, attackers can execute their own SQL query with a malicious input. For example, if an attacker enters ID like `'OR '1'='1' OR '1'='1'`, this query can be transformed into `"SELECT * FROM user_info WHERE id = ' ' OR '1'='1' OR '1'='1' AND userpw='<input_password>'".` Whatever the result of this and id comparison is (even if the id and password is empty!), the condition should always be true because the first `'1'='1'` is true.

2.2 buyer_transaction.cpp - Buffer Overflow (Intended)

The string buffer `address_input` is assigned a limited size of `sizeof(char) * 300`, but input for `sprintf()` function is not limited. This could result in a buffer overflow situation.

2.3 deposit.cpp - Race Condition (Unintended)

(Commit ID: 19a3ad6def99280a7a6776c06f63c4d1387fb455)

The program decided to withdraw or deposit a certain amount of money into the user's account based on an user input. This might result in a race condition vulnerability, because if the user input value is changed in the middle of the deposit process, then money could be withdrawn rather than deposited.

2.4 mainwindow.cpp - Buffer Overflow (Intended)

(Commit ID: 19a3ad6def99280a7a6776c06f63c4d1387fb455)

The string buffer `buffer` is assigned a limited size of `sizeof(char) * 300`, but input for `sprintf()` function is not limited. This could result in a buffer overflow situation.

3. Team4

3.1 base.hpp - Buffer Overflow (Intended)

(Commit ID: fc82cac0503616475c06016000812aacfe4fcea6)

The string buffer `val` is assigned a limited size of `sizeof(char) * 1000`, but input for `strcpy()` function is not limited. This could result in a buffer overflow situation.

3.2 All raw queries in the `src/cpp/com/security/chat/dao/` directory - SQL Injection (Intended)
(Commit ID: 5fe473bd0338f502c1ee46195944fe67e9a5c782)

Every query in this folder does not check for special characters in SQL input. This might result in SQL injection if a malicious user deliberately exploits the characters to print every record in a certain table.

4. Team5

4.1 `Engine.cpp` - Buffer Overflow (Intended)

(Commit ID: 7a9260acaf331a41edf7018ccf32c32c32650b69)

The string buffer `val` is assigned a limited size of `sizeof(char) * 5`, but input for `strcpy()` function is not limited. This could result in a buffer overflow situation.

5. Team6

5.1 `invite.c` - CWE 401 (Missing Release of Memory after Effective Lifetime) (Unintended)

(Commit ID: 6c2277b1ec436af1696043572f1131015be7906a)

At the start of the file, three connections to the MySQL database are created, but only one or two are closed when the file is finished executing. This means memory space for connection instances are not freed, which result in inappropriate use of memory for the program.

5.2 `signup.c` - CWE 401 (Missing Release of Memory after Effective Lifetime) (Unintended)

(Commit ID: e0a734ab98b84b473618b04dcd7fb52936cbf0c)

Same as 4.1

5.3 `invite.c` - Buffer Overflow (Intended)

(Commit ID: 6b5bb585ffd3b8d52218c9c311f85666a3728e38)

The string buffer `username` is assigned a limited size of `sizeof(char) * 10`, but input for `strcpy()` function is not limited. This could result in a buffer overflow situation.

5.3 `login.c` - Format String (Intended)

(Commit ID: ef53b2fa85b990efa2f8f29a049b1448172cd4ac)

Directly putting a string argument inside of the `printf()` function can result in format string vulnerability. If the user puts `"%s"` as the string input value, then the `printf()` function will print a pointer value in the program.

6. Team7

7. Team8

7.1 `DocumentEdit.c` - Buffer Overflow (Intended)

(Commit ID: 4b150246b28bc37948de12ac69dae15d73362d63)

`strcpy()` takes a string `argv[1]`, `argv[2]` without size checking, and copies it to `wikiKeyword`, `wikiDescription`. This means the attacker can insert a string with size of `MAX_LEN=20` or greater. Through this the attacker can do a buffer overflow attack by overwriting the memory.

7.2 `login.c` - Format String (Intended)

(Commit ID: 4b150246b28bc37948de12ac69dae15d73362d63)

`printf()` takes a string `argv[1]` without formatting. This means the attacker can insert a format string such as `"%p"` to obtain memory addresses. Through this the attacker can do a format string attack to obtain crucial memory addresses.

7.3 DocumentEdit.c - Buffer Overflow (Intended)

(Commit ID: 4b150246b28bc37948de12ac69dae15d73362d63)

`strcpy()` takes a string `argv[2]` without size checking, and copies it to buffer. Same as 6.1.

7.4 search2.c - Buffer Overflow (Intended)

(Commit ID: 4b150246b28bc37948de12ac69dae15d73362d63)

`strcat()` takes a string `argv[1]` without size checking, and copies it to temp. Same as 6.1.

7.5 search2.c - Buffer Overflow (Intended)

(Commit ID: 4b150246b28bc37948de12ac69dae15d73362d63)

`strcat()` takes a string `argv[2]` without size checking, and copies it to temp. Same as 6.1.

7.6 login.c - SQL Injection (Unintended)

(Commit ID: 4b150246b28bc37948de12ac69dae15d73362d63)

An attacker can inject some sql statements to manipulate the result of the query. Same as 2.1.

8. Team9

8.1 moneyTransfer.c - Buffer Overflow (Intended)

(Commit ID: 3a2c8361e02724d82bdfcdc1a4e5558897c3cfc0)

`strcpy()` takes a string `argv[4]` without size checking, and copies it to money. Same as 6.1.

8.2 checkBalance.c - Format String (Intended)

(Commit ID: 80c6abe28d2879788d8be81426981811d94a7d34)

`printf()` takes a string `userID` without formatting. Same as 6.2.

8.3 checkBalance.c - SQL Injection (Intended)

(Commit ID: b8f3cc43b18aa96207b7a6087f92220270da164a)

An attacker can inject some sql statements to manipulate the result of the query. Same as 2.1.

8.4 login.c - SQL Injection (Unintended)

(Commit ID: 3a2c8361e02724d82bdfcdc1a4e5558897c3cfc0)

An attacker can inject some sql statements to manipulate the result of the query. Same as 2.1.

8.5 login.c - SQL Injection (Unintended)

(Commit ID: b8f3cc43b18aa96207b7a6087f92220270da164a)

An attacker can inject some sql statements to manipulate the result of the query. Same as 2.1.

8.6 monyTransfer.c - SQL Injection (Unintended)

(Commit ID: a7040419ded6a214d20ad2bb40ae06d86478d1b5)

An attacker can inject some sql statements to manipulate the result of the query. Same as 2.1.

8.7 moneyTransfer.c - SQL Injection (Unintended)

(Commit ID: b260a844c661343d10ca2a0461d6ef2481601a8d)

An attacker can inject some sql statements to manipulate the result of the query. Same as 2.1.

8.7 moneyTransfer.c - CWE-1284 (Improper Validation of Specified Quantity in Input) (Intended)

(Commit ID: a7040419ded6a214d20ad2bb40ae06d86478d1b5)

The program does not check the minimum value of balance to transfer, which a user can send a negative value that leads to taking money from other accounts.

IX. Lessons

1. FrontEnd

Tools and frameworks used for frontend development are ReactJS, Socket.io and WebRTC. Since our teammates had to develop frontend components together, we felt a need to modularize our development process. This led us to use a configuration tool, Git, and strictly divide development work into several React component files so that everyone can work on separate files simultaneously. In the case of Socket.io and WebRTC, we took online lectures at Udemy to familiarize ourselves with its concepts. After some trial and errors, we understood how sockets communicate with each other on networks, and successfully implemented realtime chatting and videophone features.

2. BackEnd

Teammates were unfamiliar with Express.js, so we conducted a group study about the framework and general background knowledge required for backend server development. During this process we came across a security option for browser cookies, and used this as one of our intended vulnerabilities. In addition, we had to include C or C++ source code files inside our project. However, our project mainly consisted of TypeScript files. To solve this problem we used a tool called NodeGYP to inject our C++ source inside of Node's environment library. We first wrote vulnerable code in C++ sources which were then compiled and injected inside of Node's library using NodeGYP so that our backend server could call them via Napi. During this process we could deepen our understanding of Node and JavaScript in general.