Secure Implementation of Blokus

Charles Tucker

Pablo Guerra

# Introduction

The game we are implementing for our ISA 681 Secure Software Design final project is Blokus. This project was the joint effort of Pablo Guerra and Charles Tucker. This project is due on Saturday 5/13/2023. Our project utilizes a framework called Flutter which is described briefly below.

## Flutter

Flutter is a DART framework developed by Google for the creation of secure web applications. Its flexible design allows it to be compiled and deployed to several environments depending on the needs of the user. Many modules exist to make creating and using secure code easier to manage.

# Design/Architecture

## TLS

The majority of our design rests on the Flutter framework and its security. A lot of the TLS and backend server implementations utilize the supabase module to ensure our crypto is up-to-date and secure. This allows us to rely on the tested modules of Flutter to ensure that our certificate is accepted.

## User Login and Authentication

Our login and authentication are also performed with supabase. Instead of storing passwords for user accounts, we utilize external email addresses to links to play the game. Once a user clicks on the link they are redirected to the webpage where they can play the game and access their account. This is more secure than relying on passwords, as we can trust that these email services are likely very secure, and most people have accounts with them making it safe to utilize them as our login services.

## Gameplay

Gameplay is controlled through the backend server of which the logic was configured by us. The user interacts by clicking and dragging pieces from their pool of pieces to the board on their turn. Turns are dictated by a ribbon on the top of the user’s icon to verify whose turn it is. We implement an entire service to ensure that users can only move on their turn.

## Game Ending Scenario

We implemented a few ways to end the game. First, when no player has any valid moves the game ends and points are tallied up to determine the winners. Additionally, we have implemented a session timeout function. This serves two purposes. First, it means that users who decide to sign out rather than finish a game, or who just have a long connection failure, cannot stop an opponent from winning. If your session times out, then it counts as a loss against you. Secondly, it stops users from holding onto resources that they are no longer using, whether intentionally or accidentally.

# Installation Instructions

To install and run our application you will need the DART framework. Instructions for how to install the framework can be found here: [Install | Flutter](https://docs.flutter.dev/get-started/install). Once installed either unzip the supplied code or clone the provided git repo link. Finally, open the project in a code editor of your choice. We recommend using IntelliJ as it has an easy-to-use DART plugin, which can be installed quickly and can handle most of the flutter package installation as well as building and running the code. If you choose to run the code yourself, or even through the plugin recommended above, it is important to install the dependencies. If using the plugin go to the tools header and under Flutter click “flutter pub get”. If doing it manually run “flutter pub get” in the directory with the pubspec.yaml file. You should now have the program installed and it should be ready to run.

# Operating Instructions

To play the game, open the project using IntelliJ. From there, select the device you wish to use, this means what environment you want to run the code in. Once selected, press the run button at the top to start the building and executing of the code. The launching of your preferred environment may take a few minutes to finish. Once up create your account by entering an email and getting your one-time password authentication link to begin playing. A minimum of two players are required to play the game, once two players have joined the lobby you can either start the game or wait for more players to join.

Once a game begins you make moves by clicking and dragging pieces onto the board in accordance with the game rules. Once a player has no more moves remaining, they pass until the game is over. Once no player has any valid moves, the game is finished, and a winner is declared. To prevent poor sportsmanship, if a player takes too long to make a move they forfeit and the other players continue on, or a winner is declared.

# Game Rules

Blokus is a game about placing colored blocks on a gameboard. Play begins with blue, followed by yellow, then red, and finally green. Pieces must be placed such that they are corner to corner with a piece of the same color and do not overlap any other piece of any color. Additionally, you may not touch edge-to-edge with any of your other pieces. At the end of the game every individual square you did not place counts as -1 point, i.e. a block with 4 total squares in it counts as -4 points if not placed. If a player places all their blocks they are awarded +15 points. At the end of the game, the player with the highest number of points wins.

If playing a two-person game, each player will control two colors instead of one and alternate which color’s “turn” it is when they get to their turn.

# Why It Is Secure

Several techniques were utilized to ensure the code is as reasonably secure as we could make it. This includes peer-reviewing all code written, performing code analysis, and reviewing common vulnerabilities to ensure we are not susceptible to them. The process and purpose of these security mitigations will be described below. Additionally, the code is written using a DART framework called Flutter, which assists in managing the security of the code.

## Flutter

Flutter, as mentioned above, is a DART framework designed by Google for the development of secure web applications. We utilized its built-in capabilities as well as a few additional modules to ensure that the security of the backend server was properly managed. Before development started, we made sure to research the security posture and history of Flutter. Since its release in 2017 there have been a total of 6 CVEs found affecting the Flutter framework [2]. Of those 6, a total of 4 were of Medium or Low severity. In addition, all 6 CVEs have since been patched and are no longer exploitable. Since we are running the latest version of Flutter, we are confident that the current existing CVEs for this framework would not affect our code. Additionally, Flutter has seen widespread usage in mobile development, as well as other areas, tying the other most common framework at a 42% usage rate. Despite its wide usage, only a total of 6 CVEs have been found in the previous 6 years meaning we are confident that their security practices are solid and, judging by this historical data, are unlikely to cause a large issue in the future.

## Peer Review

Throughout the entire software development process, all code was reviewed, and tested by the other group members to ensure best practices were maintained and all code was secure. Many vulnerabilities introduced into code are caused when a programmer spends hours working on a project causing fatigue, and sometimes frustration, to result in less-than-perfect code being written. Peer reviewing allows for a second set of eyes to ensure the code that was written does exactly as it is intended to do. Much like proofreading an essay, rereading code, and having someone else double-check your work allows for most bugs to be discovered before further testing is even performed.

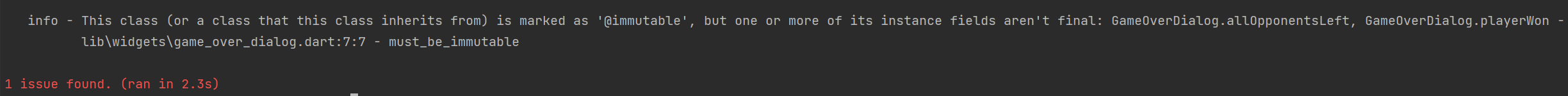
## Code Analysis

Flutter comes with an in-built ability to run static code analysis. After some research into alternatives, we decided that this would be the best analyzer for our code since it could best understand the intricacies of the Flutter framework. Below is the output of our code through its first run of static code analysis.

A picture containing screenshot, text

Description automatically generated

After looking into each of these issues we decided to fix 6 of them. The remaining problem was about field mutability and ensuring fields were set to final. After reviewing our code, we determined that the values of these fields were set only a single time and as such, did not require the final keyword despite the warning from the analyzer. This output can be verified by running “flutter analyze” in the root folder of the project. Here is the output of the second set of running the analyzer.



## Common Vulnerabilities

Throughout the entire development of the code, we remained cognizant of common vulnerabilities which are found in web applications. We reviewed the OWASP 10 Ten for web applications [1]. Below is a list of the most common vulnerabilities in web applications according to OWASP and why we are confident that our code could not be comprised by them.

## Broken Access Control

In our web application, the only access control that is needed is control over whom a player can act as and what stats a player is allowed to see. In order to view a player’s stats or make moves on another player’s behalf one would need to sign in as that user. Sign-on is done through an email system ensuring users do not have to remember a password, nor are they required to create a secure password. All that is required is a known secure email which the user can use to get the verification link.

## Cryptographic Failures

The cryptographic failures vulnerability occurs when any form of the cryptography of the application fails to do the job it is expected to do. As mentioned before, Flutter handles a large part of our encryption. Since Flutter itself has no known vulnerabilities and is run by a reputable organization we can be fairly confident that our application does not contain any cryptographic-related vulnerabilities. Additionally, anytime sensitive data is stored or transmitted, mainly passwords, we utilize appropriate cryptographically secure hash or Flutter’s capabilities to ensure that the information is secure.

## Injection

The overall web app is broken into two sections, the first is the game itself and the second is the login pages. The game itself does not have any place for a user to input data, instead, it requires dragging and dropping objects. This means that the main game itself does not have an opening for a user to perform any kind of injection attack.

The login page is the only aspect of the application which allows for user input. Login is done using an email verification method to sign in. We do strict input validation to enforce that only email-specific characters are used in the textbox input field. This has been thoroughly tested to ensure that any inputs which are invalidly input are either filtered or will not affect the backend server in any way.

## Insecure Design

Our web application contains very few components, namely the server and the clients. All communication between these two components is performed over HTTPS. This ensures that the communication is secured and not susceptible to eavesdropping or integrity concerns. Also, care was taken to ensure that only necessary information is passed between the two components. This allows for greater certainty with regard to the privacy of the data.

## Security Misconfigurations

There are very few configurations implemented in this web application. Even still we have ensured that best practices are followed throughout the development of the application. There are no default passwords, no unnecessary services or bloat from unused code, and all software is patched and at its most up to date.

## Vulnerable and Outdated Components

This type of vulnerability relates to out-of-date languages, modules, or other parts of the overall system. Since this is a relatively small project, we have been able to verify that all components of this code are at their most up-to-date and that none of these versions contain any known CVEs or exploits that need to be mitigated.

## Identification and Authentication Failures

Authentication and sign-on are handled through an email verification link. By utilizing this we ensure that so long as the user has a valid and secure email, they are able to use this app in a secure manner. Since no passwords are used, we do not have to worry about data breaches on our end and utilize the security of email accounts to ensure that our users cannot be imitated. Additionally, it is impossible to brute force a password since no passwords are needed.

## Software and Data Integrity Failures

This vulnerability occurs when software is deployed without proper verification of the underlying modules or code and without proper testing. A common example is the use of CICD pipelines which may automatically update modules without a human in the loop resulting in unexpected code changes. Since this is a relatively new project, all modules, and the framework itself were designed from the group up to the most updated version of the code. Additionally, we have no automation in our development pipeline meaning any changes would require manually updating the code. Given this, we are confident that all module and code changes will be verified by a human before being integrated into the final code.

## Security Logging and Monitoring Failures

All moves are stored and recorded as they take place. This is done entirely on the server so no clients can interfere or change the game history. This allows all moves to be viewed by all users and to be validated as correct if an issue were to come up. Since the moves of the games are meant to be public and valid for any to see, there is no concern that sensitive data would be leaked to an authorized user. Despite this, even if in the future sensitive data were to be logged in some way, these files are only visible to players of the game. This ensures that we have planned for the future in case things were to be changed in our requirements to log.

## Server-Side Request Forgery

Since we have no backend servers which are not otherwise accessible to the internet, we are not too worried about our Server-Side Request Forgery. Additionally, our entire webpage treats all users as untrusted and validates all data even if an attacker were to find a way to bypass our other defenses. Finally, none of our current endpoints, with the exception of user logins, utilize data passing through URLs, therefore according to OWASP we would not be susceptible to this kind of attack. For user logins, we pass a token that is used to authenticate a user, so if an attacker were to forge a link to attack a user the worst, they could do is trick the user into acting as the attacker which would be very obvious to the user.

## Conclusion

Since we have done our due diligence in reviewing the framework itself and the modules we have chosen to implement, we believe that our decision to go with a framework will not create a security risk. Additionally, given the practices we have chosen to implement, and our diligence in ensuring the more common web app vulnerabilities are not present. We can be reasonably confident that our code is well-designed and as secure as it could be.

# References

1. [OWASP Top Ten | OWASP Foundation](https://owasp.org/www-project-top-ten/)
2. [NVD - Results (nist.gov)](https://nvd.nist.gov/vuln/search/results?adv_search=true&isCpeNameSearch=true&query=cpe%3A2.3%3Aa%3Adart%3Adart_software_development_kit%3A-%3A*%3A*%3A*%3A*%3A*%3A*%3A*)