TENSET

Smart Contract Audit

FlipIt Game

Table of Contents

Executive Summary

"FlipIt" is an innovative crypto game that thrives on randomness, promising an engaging and rewarding experience for players. The core essence of the project lies in its dynamic gameplay. With a succinct motto of "Every time you play, you win," FlipIt captivates players by presenting them with a game of chance.

At the heart of FlipIt's gameplay are two distinct outcomes, driven by a random mechanism:
- Outcome 1: In this scenario, players may find themselves losing tokens, but the game compensates them with a unique and randomized ingredient NFT reward. This adds an element of excitement as players anticipate the intriguing and diverse ingredient NFTs they can acquire.
- Outcome 2: Alternatively, the game offers a scenario where players witness a token doubling effect. This means that participants not only keep their tokens but also see their quantity double. Moreover, alongside this token gain, players are treated to the allure of receiving a random ingredient NFT reward.

These ingredient NFTs serve as collectible assets within the game. They symbolize various components of a burger, such as top bun, bottom bun, lettuce, and meat. As players accumulate these ingredients and successfully gather all four, they gain the opportunity to mint a unique "BURGER" NFT. This signifies the completion of a burger and grants them access to an array of fascinating and diverse designs for their collection.

With an overarching goal of maximizing player engagement, FlipIt ensures that the number of Burger NFTs owned influences a player's ability to participate in the game. The more Burger NFTs in a player's possession, the greater their potential to utilize tokens and immerse themselves in the game's mechanics.

It's important to note that regardless of the outcome, FlipIt remains committed to providing players with at least one NFT per play session. However, participants should be aware of a 1% transaction tax applied to their in-game interactions, which plays a pivotal role in the project's tokenomics.

In summary, FlipIt introduces an enticing crypto gaming experience where randomness is the cornerstone of player engagement. The game's dual outcomes, encompassing both token losses and gains, coupled with the allure of ingredient NFT rewards, create an exciting and unpredictable atmosphere. This succinct summary captures the essence of FlipIt's innovative and rewarding gameplay.

Project Summary

Language	Codebase
Solidity	Private

Audit Scope

ID	File	SHA256
FG	FlipitGameV1.sol	1e9ee000978d29769de6dbbfb6a7f43 1ee8b02a286ebd1a280b82952f518a41a
RL	Random.sol	43f6d81198ef5b2697abd899a780d0a 1d0ebb1281e7e6d937102c576e8da51c3
IF	IFlipitMinter.sol	91538f09ae122ca1729dc5c91e114e6 d267c1aa4abd441bf013f3f3d53cf7beb
FM	FlipItMinter.sol	512cca69499844067e0eead160b4e6e 6d9680ae8646fed60f742b4287a737be1

General security assumptions

In conducting our security audit of the FlipIt Game, we have made a number of important security assumptions. These assumptions are fundamental to our analysis and should be considered when evaluating the overall security and potential vulnerabilities of the system:

Security of the Underlying Blockchain
Our security analysis is predicated on the assumption that the underlying blockchain (for example, Ethereum) is secure and functions as intended. Potential vulnerabilities or flaws in the blockchain protocol are outside the scope of this audit.

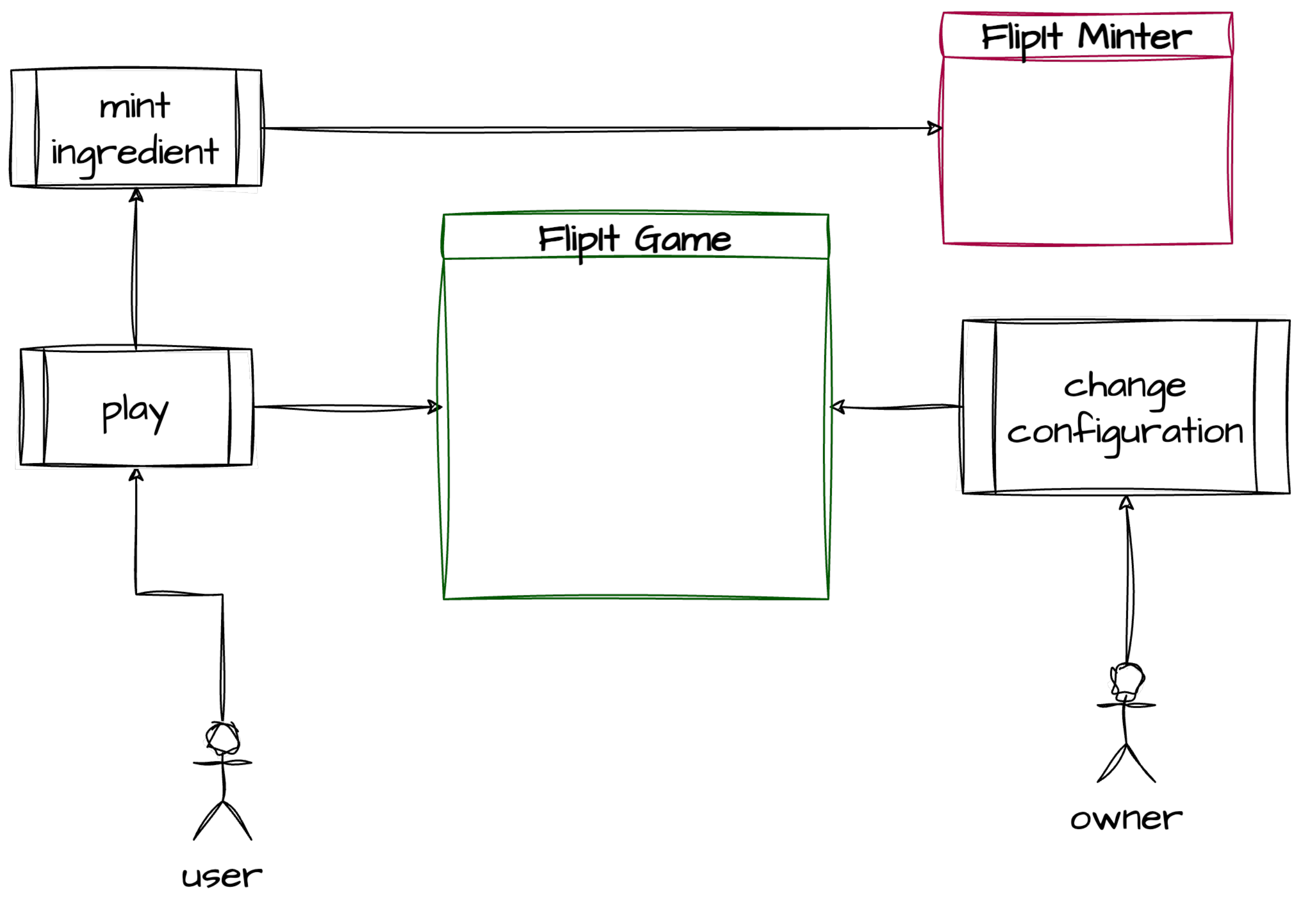
Trusted Environment
We assume that users of the smart contract will interact with it in a secure and trusted environment. This encompasses the use of secure, up-to- date software and hardware, which is free from malware or any potential interference from malicious third parties.

Private Key Security
Users of the smart contract are assumed to safeguard their private keys effectively. This involves not sharing private keys, securely storing key backups, and using hardware wallets or other secure means for key management.

Interactions with External Smart Contracts
The game smart contract interacts with token and NFT smart contracts which were not in the scope of this review, so we might/should add an assumption that these smart contracts work as intended.

These security assumptions form the basis of our audit. Deviations from these assumptions could potentially lead to risks or vulnerabilities not covered in this analysis. Therefore, the effective management of these factors is critical to ensure the ongoing security of the FlipIt Game.

Actors in the system



In our examination of the FlipIt Game, we've identified several key actors, each with their distinct roles, responsibilities, and potential influence within the protocol. These actors are as follows:

User
The "User" actor represents the default participant within the FlipIt Game ecosystem. Users engage with the game by playing it, aiming to win tokens or mint non-fungible tokens (NFTs) as rewards. They interact with the game contract by submitting transactions to play rounds of the game. Users do not have control over the contract's configuration or rules; they can only participate in the gameplay according to the established rules.

Owner
The "Owner" actor is a special wallet address that holds control over the FlipIt Game smart contract. This actor possesses additional permissions beyond those of a regular user. While the Owner can also participate in the game as a User, their primary role is to manage the contract's settings and configurations. The Owner can modify the threshold parameters that dictate the number of tokens required to play the game and the number of NFTs needed to participate. This ability to adjust the threshold parameters allows the Owner to fine-tune the gameplay experience and adapt to changing conditions.

Thread model for each actor

User

Uncertainty in Randomness
Users might be concerned about the randomness of the game's outcomes, especially if a Verifiable Random Function (VRF) is not used. The lack of a secure random number generation method could lead to doubts about the fairness of the game, as outcomes might be predicted or manipulated by all actors. This uncertainty might discourage users from participating due to a lack of trust in the randomness of the game.

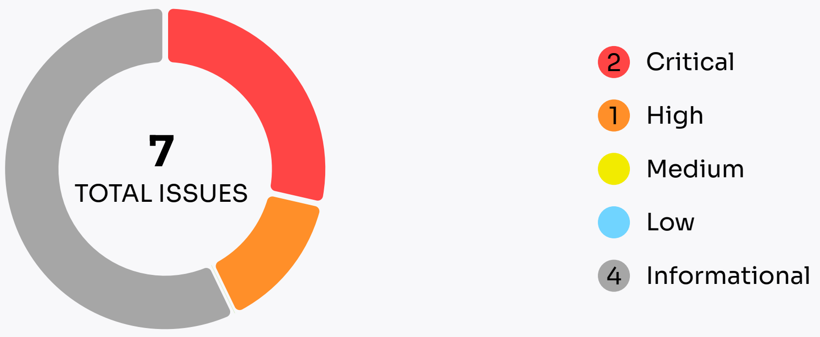
Owner manipulating game configuration
The owner could front-run user's transactions to manipulate game configuration, causing them to lose with a greater probability than at the moment a user was signing a transaction. This would quickly cause users' distrust discouraging them from playing, making this an invalid long-term tactic for an owner.

Owner

Private Key Loss or Compromise
The owner's control over the smart contract is tied to their private key. If the private key is lost, stolen, or compromised, it could have serious consequences. An attacker gaining access to the owner's private key could manipulate the game's parameters or alter the contract's behavior. It's crucial for the owner to maintain strict security measures for their private key to prevent unauthorized access and potential manipulation of the contract.

Findings

Audit Overview



Issues

Severity	Found	Resolved	Partially Fixed	Acknowledged
🟥 Critical	0	2	0	0
🟧 High	0	0	1	0
🟨 Medium	0	0	0	0
🟦 Low	0	0	0	0
● Informational	4	0	0	0
Total	**4**	**2**	**1**	**0**

FG-01
Lack of Approval Check in `play` Function.

● Critical ✓ Resolved

Description

During the audit, a vulnerability was identified in the `play` function of the FlipIt Game smart contract. The function appears to lack a proper check for user approval to spend tokens, potentially allowing users to drain the smart contract of tokens without appropriate authorization. This vulnerability poses a risk to the security and fairness of the game, as users might exploit the contract to unfairly double their tokens or deplete the smart contract`s token balance.

Recommendation

To address this vulnerability and ensure the integrity of the game, it is strongly recommended to implement a thorough approval check within the `play` function. Before executing any token transfers, the contract should verify that the user has granted the necessary approval to the contract for the specified amount of tokens. This can be achieved by using the `transferFrom` function along with the `allowance` mechanism provided by ERC-20 tokens. Implementing this check will prevent unauthorized token transfers and mitigate the risk of users draining the smart contract of tokens.

FG-02
Vulnerability to Reentrancy Attack in `play` Function.

● Critical ✓ Resolved

Description

During the audit, a critical vulnerability was identified in the `play` function of the FlipIt Game smart contract. The function is susceptible to a reentrancy attack, which could allow an attacker to exploit the external call to the `mintIngredient` function at line 135.

```
uint256[] memory rewards = minter.mintIngredient(player, 1);
```

By leveraging this vulnerability, an attacker can potentially trigger multiple calls to mint NFTs through the 'mintBatch' function from the ERC1155 standard.

Recommendation

To safeguard the `play` function against reentrancy attacks and unauthorized NFT minting, it is strongly recommended to implement the check-effects-interaction pattern. This pattern involves segregating state-changing operations from external function calls, thereby preventing unintended reentrant behavior. To apply the pattern, perform all state changes and validation checks before making any external calls to other contracts, ensuring that the contract's state is not modified after the external call

Comments

The team addressed an issue by implementing the `ReentrancyGuard` library from OpenZeppelin.

RL-01
Modulo Bias Impacting `Random` Library.

● High ✓ Partially Fixed

Description

During the audit, a vulnerability was discovered in the `Random` library, which utilizes a mechanism vulnerable to Modulo Bias. Modulo Bias is a flaw in random number generation that skews the distribution of random numbers towards certain values, leading to an imbalance in outcomes. In the context of the FlipIt Game, this bias shifts the odds of winning in favor of players, potentially causing excessive token payouts and depletion of the smart contract's token reserves. This vulnerability also introduces a risk of Denial-of-Service (DoS) attacks due to possible smart contract exhaustion.

Recommendation

To mitigate the risk of Modulo Bias and ensure a fair and secure gameplay experience, it is strongly recommended to transition from the current `Random` library to a more robust solution, such as a Verifiable Random Function (VRF). VRFs provide verifiably random and unbiased outputs, enhancing the randomness and unpredictability of outcomes. By integrating a reputable VRF solution, you can eliminate the bias issue and enhance the overall security and fairness of the FlipIt Game smart contract.

Comments

The team partially addressed the issue by reducing the winning chance, which is defined in the `UPPER_THRESHOLD_OF_CHANCE_TO_WIN` variable.

FM-01 Use Unchecked Block in `mintIngredient` Function.

● Informational Found

Description

During the audit, an opportunity for gas efficiency improvement was identified in the `mintIngredient` function at line 128 `amounts[indexOfIngredient] += 1;`. The current implementation includes an increment operation for a numerical value. Since an overflow is highly unlikely (although theoretically possible), using an unchecked block for this operation can lead to gas savings without introducing any viable security risks. The unchecked block allows the compiler to skip the overflow check, which can result in reduced gas consumption during contract execution.

Recommendation

To optimize gas consumption and enhance the efficiency of the contract, it is advised to wrap the increment operation within an unchecked block. By doing so, the compiler will omit the overflow check, resulting in lower gas costs for the `mintIngredient` function.

FM-02
Packing State Variables for Efficiency.

● Informational Found

Description

During the audit, a potential optimization opportunity was identified related to the state variables `MIN_MINT_CHANCE_THRESHOLD` and `MAX_MINT_CHANCE_THRESHOLD`. Presently, these variables are declared as `uint256`, though their intended purpose involves storing constant values that could be adequately accommodated within a `uint8` data type. Converting these variables to `uint8` will lead to a more compact packing of data, resulting in significant gas savings during contract execution.

Recommendation

To enhance the gas efficiency of the contract, it is advisable to modify the data types of `MIN_MINT_CHANCE_THRESHOLD` and `MAX_MINT_CHANCE_THRESHOLD` to `uint8`. Given the limited range of values these constants represent, the reduction in data storage and processing costs will contribute to an optimized and more economical smart contract.

FM-03
Packing Struct Variables for Efficiency.

● Informational Found

Description

During the audit, an opportunity for optimization was identified in the usage of the `Ingredient` struct. This struct currently holds several variables: `name` of type `string`, `tokenId` of type `uint256`, `minMintChanceThreshold` of type `uint256`, and `maxMintChanceThreshold` of type `uint256`. Given that the maximum values for `minMintChanceThreshold` and `maxMintChanceThreshold` are both capped at 100, and the `tokenId` is within the range of possible values for `uint64`, it is recommended to use more compact data types. This optimization will allow the struct to efficiently pack these variables into a single storage slot, reducing storage costs and enhancing gas efficiency.

Recommendation

To enhance the gas efficiency of the contract, it is advisable to modify the data types of `MIN_MINT_CHANCE_THRESHOLD` and `MAX_MINT_CHANCE_THRESHOLD` to `uint8`. Given the limited range of values these constants represent, the reduction in data storage and processing costs will contribute to an optimized and more economical smart contract.

FG-03
Packing Struct Variables for Efficiency.

● Informational   Found

Description

During the audit, an opportunity for gas optimization was identified in the variables `gameSerialId` and `thresholdSerialId`. Currently, both variables are declared as `uint256`, which provides a much larger range of values than what is required for storing IDs. Given that the maximum number of IDs to be stored is significantly lower and can be accommodated within the range of `uint64`, it is recommended to use `uint64` data types instead. This optimization will lead to reduced gas consumption during contract execution without sacrificing functionality.

Recommendation

To achieve gas efficiency, it is advised to update the data types of `gameSerialId` and `thresholdSerialId` from `uint256` to `uint64`. This change will ensure that the variables are appropriately sized to store IDs without unnecessary gas overhead. By making this adjustment, the contract will benefit from reduced gas costs while still maintaining compatibility with the expected ID values.

Disclaimer