Audited by **SovaSlava**
08.05.2025

1. Introduction:

This document is an audit report of the banzai project. Audit has been done of *126df6e448ea84218d28f00726fa191eee046de1* commit *fedorDev/banzai-gamefi-solidity* repository with a focus on the security aspects of the application's smart contract implementation.

2. About Banzai.meme

Banzai - mini games ecosystem. Players send the same amount of stakes to smart contract, storing them in a pool. When the pool is filled with 10 stakes, after 2 blocks smart contract detects the winner and sends him 90% of the pool. Oracle sends a random number to the contract, the winner chooses, using the received number and after that, the winner receives his prize.

3. Risk classification

Severity	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1 Impact

- High - least to significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.
- Info - information issue. These are not security vulnerabilities but could cause confusion down the road.

3.2 Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of the funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

4. Security assessment summary

Commit hash - *126df6e448ea84218d28f00726fa191eee046de1*

Scope: contracts/GameA.sol

5. Executive summary

During the audit period of Banzai project a total of 5 issues have been found.

Protocol summary:

Name	Banzai
Repository	https://github.com/fedorDev/banzai-gamefi-solidity
Date	07.05.2025 - 08.05.2025
Protocol type	Lottery game

Findings count:

High	0
Medium	4
Low	6
Info	1
Total	10

Summary of findings:

ID	Title	Severity
[M-01]	Risk of centralization	Medium
[M-02]	Users could not receive their fund back, if the winner has not chosen for a long time	Medium
[M-03]	The creator could not receive accumulated funds, if users play the game very often	Medium
[M-04]	Hardcoded addresses of creator and oracle	Medium
[L-01]	Floating pragma	Low
[L-02]	Revert message does not shown, if sending reward to the creator has failed	Low
[L-03]	Missing event in state changes functions	Low
[L-04]	Hardcoded price of the game	Low
[L-05]	Using tx.origin narrows the circle of potential users	Low
[L-06]	Dust of funds will be on contract balance forever	Low
[I-01]	Project has no tests	Info
[G-01]	Unnecessary update storage variable lastBlock in each call of play() function.	Gas

6. Findings

M-01. Risk of centralization

Severity: Medium	Category: Centralization
Path: contracts/GameA.sol	

The current design of the contract grants significant centralized control to the oracle, which sends random numbers to the contract. This oracle calls the function detectWinner() and only after that, the winner will be chosen.

Recommendation: Use trusted solutions for obtaining random numbers. Most projects use Chainlink VRF. This well-known oracle, which could send random numbers to the game.

M-02. Users could not receive their fund back, if the winner has not chosen for a long time

Severity: Medium	Category: Missing logic
Path: contracts/GameA.sol	

If a user sends funds to the contract, and other users do not play in this game for a long time, or oracle does not send a random number, the user would like to receive his own funds back. But he could not do it, because there is no such function.

Recommendation: Add function, which will allow users to get their funds back, if round has not ended for the 5 hours, for example, or if the oracle has not sent the number durin 10 blocks after the last player took part in the game.

M-03. The creator could not receive accumulated funds, if users play the game very often

Severity: Medium	Category: Missing logic
Path: contracts/GameA.sol	

Function claimReward() sends accumulated funds to the creator, if there is not players in the current pool. But, if the game will become very popular, this restriction could be blocked this operation, because all time will be players in the pool. So, for the creator, it will be difficult to find the moment when the pool does not have players.

```
function claimReward() public payable {
    require(msg.sender == CREATOR, "Only creator can get rewards");
    require(pool.length < 1, "Pool must be empty");    // @audit
    (bool sentReward,) = CREATOR.call{ value: address(this).balance -
0.005 ether }("");
    require(sentReward, "Failed to send reward");
  }
```

Recommendation: Remove check of pool's length and add correct calculation fund amount, which will be transferred to creator and do not transfer user's bets.

M-04. Hardcoded addresses of creator and oracle

Severity: Medium	Category: Access Control
Path: contracts/GameA.sol	

If the creator finds out that his or oracle private key has corrupted, he could not change addresses in the contract, for transfer ownership of the contract to the other his address or to remove permission from oracle's address and add new oracle.

```
contract Banzai_game_A {
  ...
  address constant CREATOR = 0x90fe1986092Ec963C4e9368837D02CB297f545Fe;
  address constant ORACLE = 0xe7c161b9CCc53447E57d322138111AaBC322Dece;
```

Recommendation: Use ownable2step.sol contract from OpenZeppelin for to be able to transfer ownership to another address. And AccessControl for oracle.

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable2Step.sol
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/AccessControl.sol

L-01. Floating pragma

Severity: Low	Category: Compiler
Path: contracts/GameA.sol	

Contract uses a floating pragma version, which allows compilation with multiple versions of the Solidity compiler.
This approach can introduce unexpected behaviors if new compiler versions are released, potentially introducing changes or bugs that impact the contract.

Recommendation: Specify exactly solidity version

L-02. Revert message does not shown, if sending reward to the creator has failed

Severity: Low	Category: Code quality
Path: contracts/GameA.sol	

If sending funds to the creator in the claimReward function fails, the creator could not be able to see the reason for it, because the function does not accept the second return value of call function (returndata).

```
  function claimReward() public payable {
    ...
    (bool sentReward,) = CREATOR.call{ value: address(this).balance -
0.005 ether }("");
    require(sentReward, "Failed to send reward");
  }
```

Recommendation: read returned bytes of call and pass them to revert messages if call has failed.

L-03. Missing event in state changes functions

Severity: Low	Category: Logging
Path: contracts/GameA.sol	

In Solidity development, emitting events is a recommended practice when state-changing functions are invoked. Currently, functions play() and detectWinner() do not emit any events.

Recommendation: Consider declaring and emitting events in the contract to enhance transparency and facilitate off-chain monitoring.

L-04. Hardcoded price of the game

Severity: Low	Category: Business Logic
Path: contracts/GameA.sol	

Contract has a hardcoded price in native currency for play in the game. But it causes problems, if the price becomes very high in USD equivalent and the pool will not fill up, until price returns to previous value. If the price will be high for a long time, users, who have already taken part in the game, have to wait, when the price goes back. Or the creator should redeploy the contract with a different price value.

```
contract Banzai_game_A {
  uint256 constant STAKE = 0.01 ether;
  ...
  function play() public payable {
    require(tx.origin == msg.sender, "No stakes from smart contracts!");
    require(msg.value == STAKE, "Invalid amount!");
```

Recommendation: Add function for creator to change price of the game. New price should be applied only after the current round will be finished.

L-05. Using tx.origin narrows the circle of potential users

Severity: Low	Category: Business Logic
Path: contracts/GameA.sol	

The usage of tx.origin narrows the circle of potential users, because it is incompatible with existing standards, such as account abstraction (ERC4337), where the transaction initiator is not an externally owned account.

Recommendation: Allow participants in the game to users, who use smart contracts. Also does not send a reward in the same tx, when the oracle sends a random number.

L-06. Dust of funds will be on contract balance forever

Severity: Low	Category: Business Logic
Path: contracts/GameA.sol	

Function claimRewards send all funds from contract balance, except of 0.005 ether. This amount will be on contract balance forever and the creator could not send it to his address.

```
  function claimReward() public payable {
    require(msg.sender == CREATOR, "Only creator can get rewards");
    require(pool.length < 1, "Pool must be empty");
    (bool sentReward,) = CREATOR.call{ value: address(this).balance -
0.005 ether }("");
```

Recommendation: send all funds from contract balance in claimRewards function, except user's funds.

I-01. Project has no tests

Severity: Info	Category: Code quality
Path: contracts/GameA.sol	

The project has no tests. it is very important to test the project code in different situations and also have tests for integration with other projects.

Recommendation: Add unit and integration tests.

G-01. Unnecessary update storage variable lastBlock in each call of play() function.

Severity: Gas	Category:Gas optimization
Path: contracts/GameA.sol	

Value of lastBlock variable used, for checking condition in the detectWinner function. So, it is needed only when the pool is full. Does not need to update the variable every call of play() function.

```
 function play() public payable {
      ...
      lastBlock = block.number;
  }

function detectWinner() public payable {
      require(msg.sender == ORACLE, "Only oracle can call this method");
      require(pool.length > 9, "Pool is not full");
      require(block.number > lastBlock+2, "Please wait 2 more blocks");
     ...
```

Recommendation: update lastBlock variable in play() function only if pool.length == 10.