ackee blockchain

Good-Ghosting Audit

9. 11. 2021

by Ackee Blockchain

Table of Contents

Document Revisions

Revisio n	Date	Description
1	8.10.2021	Initial revision
2	4.11.2021	Re-audit of fixed issues
2.1	9.11.2021	Updated trust model

1. Overview

This document presents our findings in reviewed contracts.

1.1 Ackee Blockchain

Ackee Blockchain is an auditing company based in Prague, Czech Republic, specialized in audits and security assessments. Our mission is to build a stronger blockchain community by sharing knowledge – we run a free certification course Summer School of Solidity and teach at the Czech Technical University in Prague. Ackee Blockchain is backed by the largest VC fund focused on blockchain and DeFi in Europe, Rockaway Blockchain Fund.

1.2 Audit Methodology

1. **Technical specification/documentation** - a brief overview of the system is requested from the client and the scope of the audit is defined.
2. **Tool-based analysis** - deep check with automated Solidity analysis tools MythX and Slither is performed.
3. **Manual code review** - the code is checked line by line for common vulnerabilities, code duplication, best practices and the code architecture is reviewed.
4. **Local deployment + hacking** - the contracts are deployed locally and we try to attack the system and break it.
1. **Unit testing and fuzzy testing** - run unit tests to ensure that the system works as expected, potentially write missing unit tests. Fuzzy testing is performed by Echidna.

1.3 Review team

The audit has been performed with a total time donation of 6 engineering days. The work was divided between the Chief Auditor who defined the methodology and performed manual code review. Automated tests were implemented by the second Auditor on the Chief Auditor’s assignment. The whole process was supervised by the Audit Guarantee.

Member’s Name	Position
Jan Kalivoda	Auditor
Štěpán Šonský	Chief Auditor
Josef Gattermayer, Ph.D.	Audit Guarantee

1.4 Disclaimer

We’ve put our best effort to find all vulnerabilities in the system, however our findings shouldn’t be considered as a complete list of all existing issues.

2. Scope

This chapter describes the audit scope, contains provided specification, used documentation and set main objectives for the audit process.

2.1 Coverage

Files being audited:

- GoodGhosting_Polygon_Quickswap.sol
- GoodGhosting.sol
- GoodGhostingPolygon.sol
- GoodGhostingPolygonWhitelisted.sol
- MerkleDistributor.sol
- Migrations.sol
- batched/GoodGhostingBatched.sol
- batched/GoodGhostingPolygonBatched.sol
- utils/Math.sol

Update rev. 2:
These files have been removed:
- batched/GoodGhostingBatched.sol
- batched/GoodGhostingPolygonBatched.sol
- GoodGhosting_Polygon_Quickswap.sol
- utils/Math.sol

Sources revision used during the whole auditing process:
- Repository: https://github.com/Good-Ghosting/goodghosting-smart-contracts
- Commit:
    - Revision 1: dfda8d5d5990abb7f352bc9358ac950dce1bd677
    - Fixed issues for re-audit (revision 2): b80271a65d16d6a72fd4fabce534d72d77a256d9

2.2 Supporting Documentation

We've used the official documentation, which was enough to understand the whole project.
- https://docs.goodghosting.com/goodghosting/

Update rev. 2: We have received additional information about the new changes:

1. *Games with 1 segment were not keeping track of winners*
   Solution: moved the logic to keep track of winners to the _transferDaiToContract function
2. *If a "winner" (player that made all deposits) early withdraws after making the last payment, but before the game ends, such player would still be considered a winner.*
   Solution:
   a) implemented a new variable named "winnerCount"
   b) added additional properties in the Player struct to help track if a player is a winner or not
   c) replace the player address (that made all deposits) in the winners array by a address(0) - this is to help clients reading data from the contract

2.3 Objectives

We've defined following main objectives of the audit:

- Check the code quality, architecture and best practices.
- Check if nobody unauthorized is able to steal funds.
- Check if redeem calculations are consistent and don't contain any mismatches.
- Check if nobody is allowed to cheat the game or manipulate the logic.

Update rev. 2:

- Check if the discovered issues were correctly fixed and correctness of newly implemented logic for winners declaration

3. System Overview

This chapter describes the audited system from our understanding. We'll focus mainly on `GoodGhosting.sol`, `GoodGhostingBatched.sol` and contracts which inherit from them.

3.1 GoodGhosting.sol

Following chapter describes our detailed understanding of GoodGhosting.sol and its parts.

`GoodGhosting.sol` is designed to run on the Ethereum network and also it's used as the base contract of `GoodGhostingPolygon.sol` and `GoodGhostingPolygonWhitelisted.sol`, which are designed for Polygon network.

Dependencies

The contract uses mainly proven OpenZeppelin libraries and interfaces to interact with AAVE. The contract inherits from `Ownable` and `Pausable` and uses `SafeMath` for uint256.

Structs

`Player`

- Structure for player data, contains address, flags whether the player already withdrawn funds and if can rejoin the game. Most recent paid segment and amount paid.

Functions

```
constructor(IERC20 _inboundCurrency, ILendingPoolAddressesProvider
_lendingPoolAddressProvider, uint256 _segmentCount, uint256
_segmentLength, uint256 _segmentPayment, uint256
_earlyWithdrawalFee, uint256 _customFee, address _dataProvider,
uint256 _maxPlayersCount, IERC20 _incentiveToken) public
```

- Initializes a new instance of the game with a set of parameters containing segment settings, max players, fees, token address, pool addresses provider...

`pause() external onlyOwner whenNotPaused`

- Pauses the game. When the game is paused, players can't join the game, deposit funds and early withdraw funds.

`unpause() external onlyOwner whenPaused`

- Resumes the game.

`adminFeeWithdraw() external virtual onlyOwner whenGameIsCompleted`

- Allows admin to withdraw fees and additional incentives if there is no winner.

`joinGame() external virtual whenNotPaused`

- Allows the player to join the game.

`earlyWithdraw() external whenNotPaused whenGameIsNotCompleted`

- Allows the player to withdraw funds before the game ends for a fee set during the deployment.

`withdraw() external virtual`

- Used for withdrawing funds after the game ends. Winners also obtain the reward share.

`makeDeposit() external whenNotPaused`

- Function for players to deposit tokens into the game every game segment.

`getNumberOfPlayers() external view returns (uint256)`

- Returns number of players in the game.

`redeemFromExternalPool() public virtual whenGameIsCompleted`

- Redeems funds from the lending pool. Calculates the admin fee and total game interest used for players' rewards. Can be called only after the game ends and only once.

`getCurrentSegment() public view returns (uint256)`

- Returns the number of current game segment.

`isGameCompleted() public view returns (bool)`

- Checks whether the game is running or completed.

`_transferDaiToContract() internal`

- Transfers the funds from the player to the contract and deposits funds into the pool.

`_joinGame() internal`

- Internal method to join players into the game.

3.2 GoodGhostingBatched.sol

Implementation is very similar to GoodGhosting.sol, but uses MerkleDistributor to allow only whitelisted players to join the game. Merkle root is passed into the constructor and a player has to provide Merkle proof and index to join the game. A batched version of the contract stores deposits for each game segment.

`GoodGhostingBatched.sol` is designed to run on the Ethereum network and also it’s used as the base contract of `GoodGhostingPolygonBatched.sol`, which is designed for the Polygon network.

4.Security Specification

This section specifies single roles and their relationships in terms of security in our understanding of the audited system. These understandings are later validated.

4.1 Actors

This part describes actors of the system, their roles and permissions.

Owner

Owner deploys the contract into the network. Contract also uses the Ownable pattern, so it's possible to transfer the ownership to another owner or renounce the ownership. Owner can pause/resume the game and withdraw fees.

Player

Player is an Ethereum account, which called joinGame() function. During the game, players can deposit funds every game segment. Also players can early withdraw their funds before the game ends, but in this case, a withdrawal fee is applied. Withdrawal after the game is not a subject of fee.

Lending Pool

GoodGhosting system uses external landing pools like AAVE to farm deposits. This actor is a black box in our trust model.

4.2 Trust model

Players have to trust the owner in terms of pausing the game. If the game is paused for more than 1 game segment, nobody will be the winner and the owner can harvest all pool's rewards. Another part of trust is put into landing pools, but GoodGhosting is not responsible for security of these external systems.

Update rev. 2.1:
The 1-segment game now supports winners, eliminating the previous problem with the trust model.

5. Findings

This chapter shows detailed output of our analysis and testing.

5.1 General Comments

Code is very well documented and every function has understandable comments. Code quality is also solid and follows basic good practices like `require` statements. Unit test coverage is also excellent.

5.2 Issues

Using our toolset, manual code review and unit testing we've identified the following issues.

Low

Low severity issues are more comments and recommendations rather than security issues. We provide hints on how to improve code readability and follow best practices. Further actions depend on the development team decision.

ID	Description	Contract	Line	Status
L1	Outdated compiler	All	3	Acknowledged
L2	Use of optimizer	All		Fixed
L3	Use of uint256 where uint8 is enough	GoodGhosting.sol GoodGhostingBatched.sol	51, 53 48, 50	Fixed
L4	Variable packing	Nearly all		Fixed

L1: Compiler version 0.6.11 is very outdated, we recommend using 0.8.0 at minimum. But also the newest one is also not recommended. Good practice is a compiler version established for 3-6 months.

L1 rev. 2: The compiler is going to be upgraded in a new version of this contract. It will not be corrected in this contract.

L2: Optimizer can contain bugs and it could lead to unexpected behavior or even vulnerabilities. It's not a direct threat, but we recommend optimizing the code manually.

L2 rev. 2: Optimizer was removed from the configuration file.

L3: Regarding `require` statements on lines 141 and 142, fee values are in percents, so it's safe to use `uint8` instead of `uint256`. This can save little gas in combination with L4.

L3 rev. 2: Variables (`earlyWithdrawalFee`, `customFee`) were retyped to `uint8`.

L4: State variables could have a more optimal ordering to save gas using the variable packing (32 bytes). E.g. packing boolean and address values together to save some storage.

L4 rev. 2: The initialization of the variables has been correctly rearranged.

Medium

Medium severity issues aren't security vulnerabilities, but should be clearly clarified or fixed.

ID	Description	Contract	Line	Status
~~M1~~	~~Unlimited allowance~~	~~GoodGhosting_Polygon_Quickswap.sol~~ ~~GoodGhostingBatched.sol~~	~~424~~ ~~454~~	Deprecated
M2	Renounce ownership	GoodGhosting.sol GoodGhostingBatched.sol GoodGhosting_Polygon_Quickswap.sol	16 17 19	Fixed

M1: Contract sets allowance for a router and a staking pool to `uint256` max value. This could lead to unlimited drain of funds in case the router or the staking pool gets hacked. We recommend setting a safer value, maximum which is necessary.

M1 rev. 2: All contracts addressed by this issue were deleted because they were no longer in use.

M2: Open Zeppelin's Ownable pattern allows the current owner to renounce the ownership of the contract, which could potentially cause inability to call `pause()`, `unpause()` and `adminFeeWithdraw()` methods. We recommend overriding the `renounceOwnership()` method to disable this unwanted feature.

M2 rev. 2: The issue was correctly fixed by adding an ownership control flag and overriding the `renounceOwnership()` method.

High

High severity issues are security vulnerabilities, which require specific steps and conditions to be exploited. These issues have to be fixed.

ID	Description	Contract	Line	Status
~~H1~~	~~First player doesn't receive the bonus~~	~~GoodGhosting_Polygon_Quickswap.sol~~ ~~GoodGhostingBatched.sol~~ ~~GoodGhostingPolygonBatched.sol~~	~~524~~ ~~399~~ ~~417~~	Deprecated

H1: First player, who calls the `withdraw()` method, is responsible for calling `redeemFromExternalPool()` to calculate rewards for all players. But in mentioned smart contracts, `redeemFromExternalPool()` method is called after the player's payout calculation, which leads to the following misbehaving situation. In case `redeemFromExternalPool()` would be called independently, before the first player calls `withdraw()`, then this issue would be invalid.

If the first player, who calls `withdraw()` is also the winner, then he will not receive rewards, because they're not calculated yet.

This issue has an easy fix, we suggest moving this block:

```
if (!redeemed) {
      redeemFromExternalPool();
}
```

ideally right below this line:

```
player.withdrawn = true;
```

H1 rev. 2: All contracts addressed by this issue were deleted because they were no longer in use.

Critical

Direct critical security threats, which could be instantly misused to attack the system. These issues have to be fixed.

✓ We haven't found any critical severity issues.

5.3 Unit testing

Provided tests are very well-written. Only branch coverage is not fully covered, because of ERC20 token transfers/approvals in `require` statements. Therefore, these statements were replaced (for testing purposes) with the `assert` keyword to reveal "real" coverage (98.48%), which is excellent. The resulting table can be viewed in Appendix A.

5.4 Fuzzy testing

Game mechanics were additionally tested with Echidna. Since time is taking a role in a contract, it was quite a challenging task for a fuzzer (with a given scope), so we decided at least to test various player count variables for possible misbehaving.

Overview

ID	Description	Contract/Function	Result
F1	There is always less winners than activePlayers	GoodGhosting	PASSED
F2	There is always less activePlayers than iterablePlayers	GoodGhosting	PASSED

6. Conclusion

We started with MythX and Slither tool analysis of the contract. Then the Chief Auditor performed a manual code review and a local smart contract deployment. In parallel, the second auditor checked coverage, the correct contract behavior and wrote fuzzy tests.

No direct security issues were found. However we've identified one high issue, which could lead to misbehavior in a specific situation and a winner could lose his reward. We have also found some medium issues which we recommend to get fixed and a few minor issues which are more good practice recommendations rather than issues.

Update rev. 2: All discovered issues in Revision 1 of the audit were correctly fixed by the client.

Appendix A

Coverage results

```
-------------------------------------|----------|----------|----------|----------|----------------
File                                 |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines
-------------------------------------|----------|----------|----------|----------|----------------
 contracts/                          |      100 |    98.48 |      100 |      100 |
  GoodGhosting.sol                   |      100 |      100 |      100 |      100 |
  GoodGhostingPolygon.sol            |      100 |    95.45 |      100 |      100 |
  GoodGhostingPolygonWhitelisted.sol |      100 |      100 |      100 |      100 |
  MerkleDistributor.sol              |      100 |      100 |      100 |      100 |
 contracts/merkle/                   |      100 |      100 |      100 |      100 |
  IMerkleDistributor.sol             |      100 |      100 |      100 |      100 |
 contracts/moola/                    |      100 |      100 |      100 |      100 |
  ILendingPoolCore.sol               |      100 |      100 |      100 |      100 |
  MAToken.sol                        |      100 |      100 |      100 |      100 |
  MILendingPool.sol                  |      100 |      100 |      100 |      100 |
 contracts/quickswap/                |      100 |      100 |      100 |      100 |
  IPair.sol                          |      100 |      100 |      100 |      100 |
  IRouter.sol                        |      100 |      100 |      100 |      100 |
  IStake.sol                         |      100 |      100 |      100 |      100 |
-------------------------------------|----------|----------|----------|----------|----------------
All files                            |      100 |    98.48 |      100 |      100 |
-------------------------------------|----------|----------|----------|----------|----------------
```

ackee blockchain

Thank You

Ackee Blockchain a.s.

Prague, Czech Republic

hello@ackeeblockchain.com

https://discord.gg/z4KDUbuPxq