ackee blockchain

Good-Ghosting Polygon Curve Audit

17 January 2022

by Ackee Blockchain

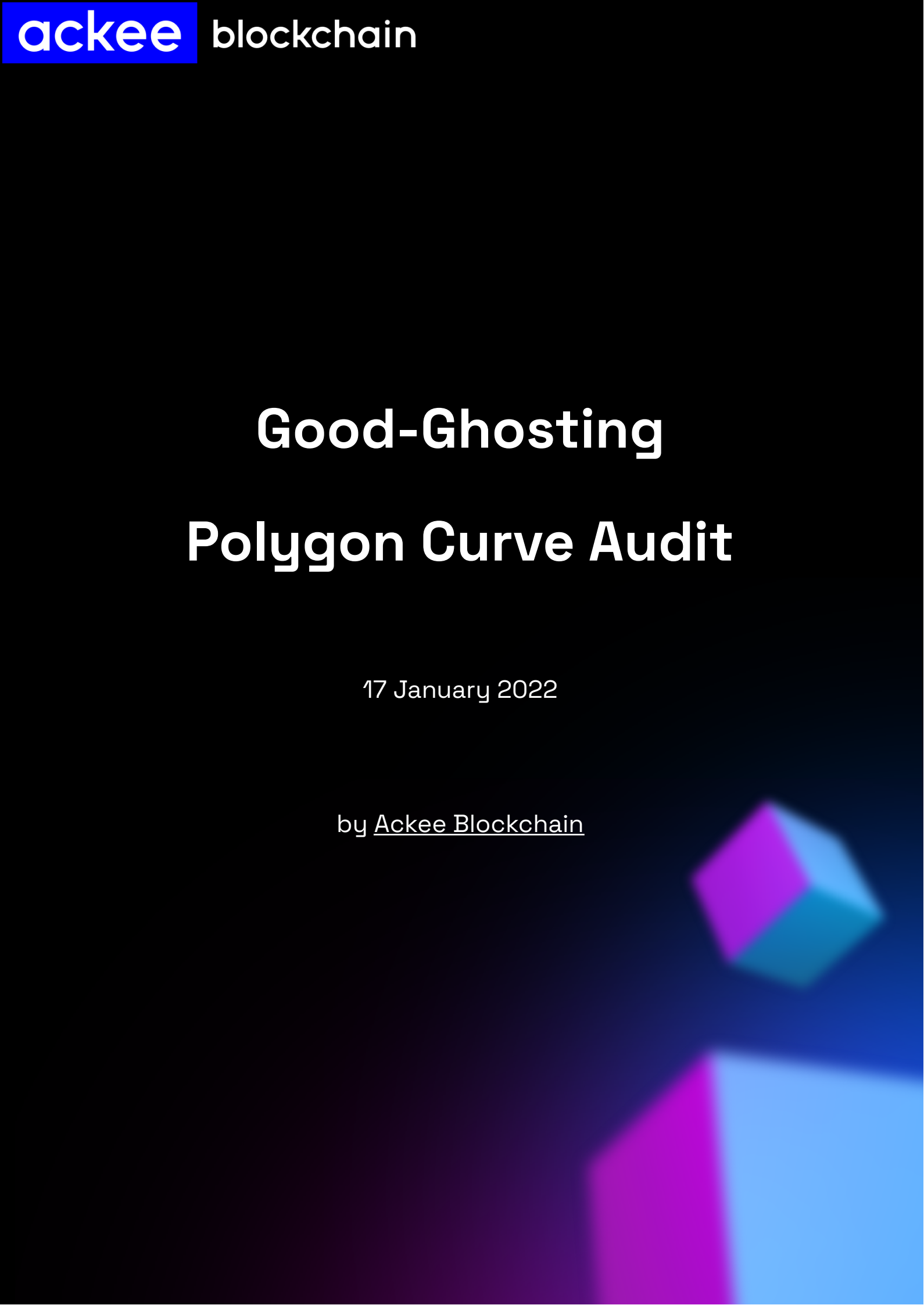

Table of Contents

Document Revisions

Revision	Date	Description
1	10 Dec 2021	Initial revision
1.1	17 Jan 2022	Updated branch, commit and issues' status

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 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.
5. **Unit testing** - run unit tests to ensure that the system works as expected, potentially write missing unit tests.

1.3 Review team

The audit has been performed with a total time donation of 4 engineering days. The complete work was done by the Lead Auditor. The whole process was supervised by the Audit Supervisor.

Member's Name	Position
Štěpán Šonský	Lead Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

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:

- GoodGhostingPolygonCurve.sol
- GoodGhostingPolygonCurveWhitelisted.sol
- MerkleDistributor.sol

Sources revision used during the whole auditing process:

- Repository: https://github.com/Good-Ghosting/goodghosting-smart-contracts
- Branch: master
- Commit: 0d7caecd3ecfad2e21fe9e8527a34d679c2c8e9c

2.2 Supporting Documentation

We've used the official documentation, which was enough to understand the whole project and the game logic.

- https://docs.goodghosting.com/goodghosting/

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.

3. System Overview

This chapter describes the audited system from our understanding. We'll focus on `GoodGhostingPolygonCurve.sol`, `GoodGhostingPolygonCurveWhitelisted.sol` and `MerkleDistributor.sol`.

3.1 GoodGhostingPolygonCurve.sol

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

`GoodGhostingPolygonCurve.sol` is designed to run on the Polygon network and interacts with lending pools AAVE and Atri on Curve.

Dependencies

The contract uses OpenZeppelin libraries and interfaces to interact with Curve protocol. The contract inherits from `Ownable` and `Pausable` and uses `SafeMath` for uint256.

Structs

`Player`

- Structure for player data, contains address, flags whether the player has already withdrawn funds, if can rejoin the game and if is the winner.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

Functions

```
constructor(IERC20 _inboundCurrency, ICurvePool _pool, int128
_inboundTokenIndex, uint64 _poolType, ICurveGauge _gauge, uint256
_segmentCount, uint256 _segmentLength, uint256 _segmentPayment,
uint128 _earlyWithdrawalFee, uint128 _customFee, uint256
_maxPlayersCount, IERC20 _curve, IERC20 _matic, IERC20
_incentiveToken) public
```

- Initializes a new instance of the game with a set of parameters containing segment settings, max players, fees, token addresses, Curve token address, wrapped MATIC token address...
- All inputs are validated using `require` statements.

`pause() external onlyOwner whenNotPaused`

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

`unpause() external onlyOwner whenPaused`

- Resumes the game.
- Only the owner can resume the game.

`adminFeeWithdraw() external onlyOwner whenGameIsCompleted`

- Allows admin to withdraw fees and additional incentives if there is no winner.
- Fees can be withdrawn only by the owner and only after the game ends.

`joinGame(uint256 _minAmount) external virtual whenNotPaused`

- Allows the player to join the game, if the game is not paused.

`earlyWithdraw(uint256 _minAmount) external whenNotPaused whenGameIsNotCompleted`

- Allows the player to withdraw funds before the game ends for a fee set during the deployment.
- Can be called only when the game is not paused and not ended.

`withdraw(uint256 _minAmount) external`

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

`makeDeposit(uint256 _minAmount) external whenNotPaused`

- Function for players to deposit tokens into the game every game segment.
- Funds can be deposited only when the game is not paused.

`unlockRenounceOwnership() external onlyOwner`

- Unlocks ability to renounce ownership.

`getNumberOfPlayers() external view returns (uint256)`

- Returns the number of players in the game.

`renounceOwnership() public override onlyOwner`

- Renounces the contract ownership if it's enabled.

`redeemFromExternalPool(uint256 _minAmount) public 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(uint256 _minAmount) internal`

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

`_joinGame(uint256 _minAmount) internal`

- Internal method to join players into the game.

3.2 GoodGhostingPolygonCurveWhitelisted.sol

The contract inherits from the `GoodGhostingPolygonCurve` and `MerkleDistributor` to allow only whitelisted players to join the game. Value `merkleRoot` is passed into the constructor and a player has to provide `merkleProof`, `index` and `minAmount` to join the game.

`constructor(IERC20 _inboundCurrency, ICurvePool _pool, int128 _inboundTokenIndex, uint64 _poolType, ICurveGauge _gauge, uint256 _segmentCount, uint256 _segmentLength, uint256 _segmentPayment, uint128 _earlyWithdrawalFee, uint128 _customFee, uint256 _maxPlayersCount, IERC20 _curve, IERC20 _matic, IERC20 _incentiveToken, bytes32 _merkleRoot) public`

- Passes all the parameters to the `GoodGhostingPolygonCurve` contructor and `merkleRoot` into the constructor of the `MerkleDistributor`.

`joinGame(uint256 _minAmount) external override whenNotPaused`

- Reverts all attempts to join the game without the proof.

`joinWhitelistedGame(uint256 index, bytes32[] calldata merkleProof, uint256 _minAmount) external whenNotPaused`

- Joins the player into the game if its whitelisted (provides valid `merkleProof`).

3.3 MerkleDistributor.sol

`MerkleDistributor` is used for validating Merkle proof in `GoodGhostingPolygonCurveWhitelisted`. It uses the `MerkleProof.sol` from the OpenZeppelin library. The contract hasn't been changed since our previous audit.

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 Pools

`GoodGhostingPolygonCurve` uses external lending pools on Curve on Polygon Network (AAVE and Atricrypto) to farm deposits. These actors are a black boxes 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 can be the winner and the owner can harvest all pool's rewards. Another part of trust is put into lending pools, but GoodGhosting is not responsible for security of these external systems.

5. Findings

This chapter shows detailed output of our analysis and testing.

5.1 General Comments

The code is very well documented and every function has understandable comments. The code quality is also solid and follows good practices. The game logic stays the same as in other previously audited contracts. Unit test coverage of Curve contracts 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	Default value assignment	GoodGhosting PolygonCurve. sol	36 38 54 67	Acknowledged

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.

L2: There is no need to explicitly init variables to default values.

Medium

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

✓ We haven't found any medium severity issues.

High

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

✓ We haven't found any high severity issues.

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 and have 100% coverage. The resulting table can be viewed in Appendix A.

6. Conclusion

We started the assessment with the static analysis of contracts. Then we performed a manual code review and a local smart contract deployment. We've checked test coverage and the correct contract behavior.

Most of the issues from our previous audits of other GoodGhosting contracts have been fixed. So we haven't found any new issues in GoodGhosting Curve contracts.

Appendix A

Coverage results

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	99.76	85.37	100	99.76	
GoodGhosting.sol	99.25	92.31	100	99.26	194
GoodGhostingPolygon.sol	100	85.71	100	100	
GoodGhostingPolygonCurve.sol	100	80.72	100	100	
GoodGhostingPolygonCurveWhitelisted.sol	100	100	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	
All files	99.76	85.37	100	99.76	

ackee blockchain

Thank You

Ackee Blockchain a.s.


Prague, Czech Republic


hello@ackeeblockchain.com


https://discord.gg/z4KDUbuPxq