HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Kalmar

Date: April 3rd, 2021

This document may contain confidential information about IT systems and the intellectual property of the Customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities fixed - upon a decision of the Customer.

Document

Name	Smart Contract Code Review and Security Analysis Report for Kalmar.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Sale
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/kalmar-io/leverage-yield-contracts
Commit	ad08aef5a2281639a3226f31d4d8d5aaba73967e
Timeline	23 MAR 2021 – 3 APR 2021
Changelog	3 APR 2021 – initial audit

Table of contents

Introduction 4
Scope 4
Executive Summary 5
Severity Definitions 7
AS-IS overview 8
Conclusion 18
Disclaimers 19

Introduction

Hacken OÜ (Consultant) was contracted by Kalmar (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of Customer's smart contract and its code review conducted between March 23rd, 2021 – April 3rd, 2021.

Scope

The scope of the project is smart contracts in the repository:
Repository: https://github.com/kalmar-io/leverage-yield-contracts

Files:

Bank.sol
BankConfig.sol
ConfigurableInterestBankConfig.sol
Goblin.sol
GoblinConfig.sol
MasterChefGoblin.sol
MasterChefGoblinConfig.sol
MasterChefPoolRewardPairGoblin.sol
SafeToken.sol
SimpleBankConfig.sol
SimplePriceOracle.sol
StrategyAllETHOnly.sol
StrategyAddTwoSidesOptimal.sol
StrategyLiquidate.sol
StrategyWithdrawMinimizeTrading.sol

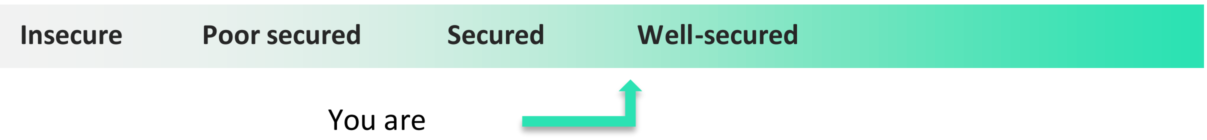
We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

Category	Check Item
Code review	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops ▪ DoS with (Unexpected) Throw ▪ DoS with Block Gas Limit ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ Costly Loop ▪ ERC20 API violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level

	■ Deployment Consistency ■ Repository Consistency ■ Data Consistency
Functional review	■ Business Logics Review ■ Functionality Checks ■ Access Control & Authorization ■ Escrow manipulation ■ Token Supply manipulation ■ Assets integrity ■ User Balances manipulation ■ Kill-Switch Mechanism ■ Operation Trails & Event Generation

Executive Summary

According to the assessment, the Customer's smart contracts are secure.



Insecure Poor secured Secured Well-secured

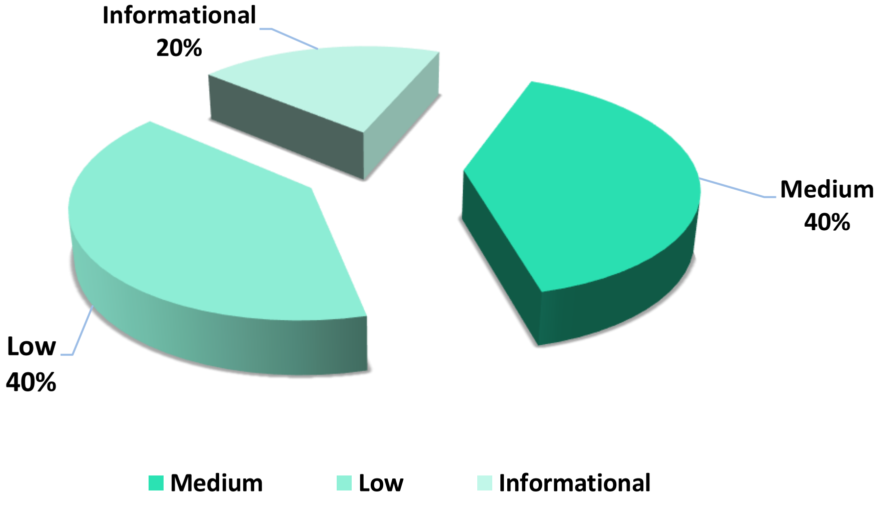
You are

Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. A general overview is presented in AS-IS section, and all found issues can be found in the Audit overview section.

Security engineers found **2** medium, **2** low and **1** informational issue during the audit.

Notice: the code is provided without tests and build configs. This complicates review process.

Graph 1. The distribution of vulnerabilities after the first review.



Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that can't have a significant impact on execution
Informational / Code Style / Best Practice	Informational vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

AS-IS overview

Bank.sol

Description

Bank is the ERC-20 token that also acts as coin storage and main gateway to goblins.

Imports

Bank has following imports:

- import "openzeppelin-solidity-2.3.0/contracts/ownership/Ownable.sol"
- import "openzeppelin-solidity-2.3.0/contracts/token/ERC20/ERC20.sol"
- import "openzeppelin-solidity-2.3.0/contracts/math/SafeMath.sol"
- import "openzeppelin-solidity-2.3.0/contracts/math/Math.sol"
- import "openzeppelin-solidity-2.3.0/contracts/utils/ReentrancyGuard.sol"
- import "./BankConfig.sol"
- import "./Goblin.sol"
- import "./SafeToken.sol"

Inheritance

Bank is ERC20, ReentrancyGuard, Ownable.

Usages

Bank contract has following usages:

- SafeMath for uint256.
- SafeToken for address.

Structs

Bank contract has following data structures:

- Position

Enums

Bank contract has no custom enums.

Events

Bank contract has following events:

- event AddDebt(uint256 indexed id, uint256 debtShare)
- event RemoveDebt(uint256 indexed id, uint256 debtShare)
- event Work(uint256 indexed id, uint256 loan)
- event Kill(uint256 indexed id, address indexed killer, uint256 prize, uint256 left)

Modifiers

Bank has following modifiers:

- onlyEOA – require that the caller must be an EOA account to avoid flash loans.
- Accrue – add more debt to the global debt pool.

Fields

Bank contract has following fields and constants:

- string public name = "Interest Bearing BNB"
- string public symbol = "iBNB"
- uint8 public decimals = 18
- BankConfig public config
- mapping (uint256 => Position) public positions
- uint256 public nextPositionID = 1
- uint256 public glbDebtShare
- uint256 public glbDebtVal
- uint256 public lastAccrueTime
- uint256 public reservePool

Functions

Bank has following public and external functions:

- ***constructor***
  Description
  Initializes the contract.
  Input parameters
  - BankConfig _config

  Constraints
  None
  Events emit

None

Output

None

- **pendingInterest, debtShareToVal, debtValToShare, positionInfo, totalETH**

  Description

  View functions

- ***deposit***

  Description

  Deposits ETH. Mints tokens instead.

  Input parameters

  None

  Constraints

  None

  Events emit

  None

  Output

  None

- ***withdraw***

  Description

  Withdraws ETH. Burns tokens instead.

  Input parameters

  - uint256 share

  Constraints

  None

  Events emit

  None

  Output

  None

- ***work***

  Description

  Create a new farming position.

  Input parameters

  - uint256 id
  - address goblin
  - uint256 loan
  - uint256 maxReturn
  - bytes calldata data

  Constraints

  - onlyEOA modifier.

  - id should be valid.
  - goblin address should be authorized goblin.
  - loan should be 0 or debt should be allowed by config.

  Events emit

  Emits Work event

  Output

  None

- ***kill***

  Description

  Kill a position if requirements are met.

  Input parameters

  - uint256 id

  Constraints

  - debtShare should be more than 0.
  - Kill factor conditions should be met.

  Events emit

  Emits Killevent

  Output

  None

- ***updateConfig, withdrawReserve, reduceReserve, recover***

  Description

  Protected owner functions.

ConfigurableInterestBankConfig.sol, MasterChefGoblinConfig.sol, SimpleBankConfig.sol

Description

ConfigurableInterestBankConfig, MasterChefGoblinConfig and SimpleBankConfig are contracts for storing parameters. All setter functions are protected and available only for owner.

MasterChefGoblin.sol, MasterChefPoolRewardPairGoblin.sol

Description

“Goblins” that works with masterchef.

Imports

Contracts has following imports:

- import "openzeppelin-solidity-2.3.0/contracts/ownership/Ownable.sol"

- import "openzeppelin-solidity-2.3.0/contracts/math/SafeMath.sol"
- import "openzeppelin-solidity-2.3.0/contracts/utils/ReentrancyGuard.sol"
- import "openzeppelin-solidity-2.3.0/contracts/token/ERC20/IERC20.sol"
- import "@uniswap/v2-core/contracts/interfaces/IUniswapV2Factory.sol"
- import "@uniswap/v2-core/contracts/interfaces/IUniswapV2Pair.sol"
- import "@uniswap/v2-core/contracts/libraries/Math.sol"
- import "./uniswap/IUniswapV2Router02.sol"
- import "./Strategy.sol"
- import "./SafeToken.sol"
- import "./Goblin.sol"
- import "./interfaces/IMasterChef.sol"

Inheritance

Contracts are Ownable, ReentrancyGuard, Goblin.

Usages

Contracts has following usages:

- SafeMath for uint256.
- SafeToken for address.

Structs

Contracts has following data structures:

- Position

Enums

Contracts has no custom enums.

Events

Contracts has following events:

- event Reinvest(address indexed caller, uint256 reward, uint256 bounty)
- event AddShare(uint256 indexed id, uint256 share)
- event RemoveShare(uint256 indexed id, uint256 share)
- event Liquidate(uint256 indexed id, uint256 wad)

Modifiers

Contracts has following modifiers:

- onlyEOA
- onlyOperator

Fields

MasterChefGoblin contract has following fields and constants:

- IMasterChef public masterChef
- IUniswapV2Factory public factory
- IUniswapV2Router02 public router
- IUniswapV2Pair public lpToken
- address public weth
- address public fToken
- address public rewardToken
- address public operator
- uint256 public pid
- mapping(uint256 => uint256) public shares
- mapping(address => bool) public okStrats
- uint256 public totalShare
- Strategy public addStra
- Strategy public liqStrat
- uint256 public reinvestBountyBps

MasterChefPoolRewardPairGoblin contract has following fields and constants:

- IMasterChef public masterChef
- IUniswapV2Factory public factory
- IUniswapV2Router02 public router
- IUniswapV2Pair public lpToken
- address public weth
- address public rewardToken
- address public operator
- uint256 public constant pid = 12
- mapping(uint256 => uint256) public shares
- mapping(address => bool) public okStrats
- uint256 public totalShare
- Strategy public addStrat
- Strategy public liqStrat

- uint256 public reinvestBountyBps

Functions

MasterChefGoblin has following public and external functions:

- ***constructor***
  Description
  Initializes the contract.
  Input parameters
  - address _operator
  - IMasterChef _masterChef
  - IUniswapV2Router02 _router
  - uint256 _pid
  - Strategy _addStrat
  - Strategy _liqStrat
  - uint256 _reinvestBountyBps

  Constraints
  None
  Events emit
  None
  Output
  None
- ***shareToBalance, balanceToShare, getMktSellAmount, health***
  Description
  View and pure functions.
- ***reinvest***
  Description
  Re-invest whatever this worker has earned back to staked LP tokens.
  Input parameters
  None
  Constraints
  None
  Events emit
  Emits Reinvest event.
  Output
  None
- ***work***
  Description
  Work on the given position
  Input parameters

- uint256 id
- address user
- uint256 debt
- bytes calldata data

Constraints

- onlyOperator modifier.
- Strategy should be approved.

Events emit

None

Output

None

- ***liquidate***

Description

Liquidate the given position by converting it to ETH and return back to caller.

Input parameters

- uint256 id

Constraints

- onlyOperator modifier.

Events emit

Emits Liquidate event.

Output

None

- ***recover, setReinvestBountyBps, setStrategyOk, setCriticalStrategies***

Description

Protected owner functions.

SimplePriceOracle.sol

Description

SimplePriceOracle is contract for storing token prices. Setter is protected for owner only.

StrategyAllETHOnly.sol, StrategyAllETHOnly.sol, StrategyLiquidate.sol, StrategyWithdrawMinimizeTrading.sol

Description

Strategy contracts that are used by golblins.

Audit overview

🟥🟥🟥🟥 Critical

No critical issues were found.

🟨🟨🟨 High

No high severity issues were found.

🟩🟩 Medium

1. The code is not tested.

   Contract: All

   Recommendation: implement unit test for all contracts.

2. Build tools and configs are not provided. Contracts may not be compiled in a current state.

   Contract: All

   Recommendation: configure build tools.

🟦 Low

1. safeApprove(address, uint256(-1)) function may fail for some specific implementation of underlying token. For example, COMP token volatiles the ERC-20 standard and reverts in if max uint256 is passed.

   Contracts: StrategyWithdrawMinimizeTrading.sol, StrategyLiquidate.sol, StrategyAllETHOnly.sol,

   Functions: execute

   Recommendation: ensure that lp pairs with such tokens are not added to the system.

2. Custom uniswap routers should not be used.

   Contracts: UniswapV2Router02.sol

   Recommendation: ensure that the UniswapV2Router02 version that is in the repository is used only for testing purposes.

■ Informational / Code style / Best Practice

1. Some code style issues were found by static code analyzers.

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. For the contract, high-level description of functionality was presented in As-Is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security engineers found **2** medium, **2** low and **1** informational issue during the audit.

Notice: the code is provided without tests and build configs. This complicates review process.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on blockchain platform. The platform, its programming language, and other software related to the smart contract can have its vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.