HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: BarnBridge
Date: September 29th, 2020

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Barn Bridge
Type	Yield Farming, Staking
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/BarnBridge/BarnBridge-YieldFarming
Commit	cac9d607eea390c1844fb97700c1321d42988b15
Timeline	24 SEP 2020 - 25 SEP 2020
Changelog	25 SEP 2020 - Initial Audit 29 SEP 2020 - Secondary Audit

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by Barn Bridge (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 September 24th, 2020 - September 25th, 2020.

Secondary audit conducted at September 29th, 2020.

Scope

The scope of the project is smart contracts in the repository:

```
Repository  https://github.com/BarnBridge/BarnBridge-YieldFarming
Commit      f9abcace1dc30c4004caa5068b5bc262603b19d1
```

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 ▪ Data Consistency manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

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

Insecure	Poor secured	Secured	Well-secured

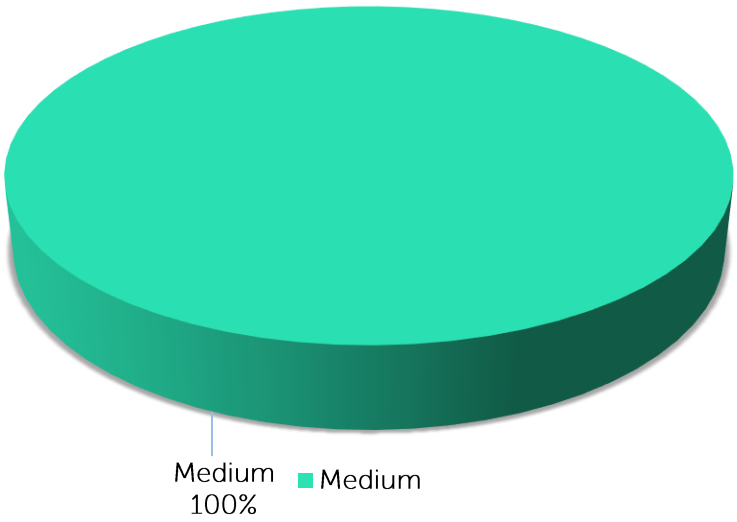
You are here

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 critical and 1 medium severity issues during the initial audit. The Customer did not provide any requirements or specifications so we cannot validate the correctness of all calculations. The code is well-tested and most of possible cases are covered.

All critical issues were fixed before the secondary audit.

Graph 1. The distribution of vulnerabilities.



Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Severity Definitions

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

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

AS-IS overview

CommunityVault.sol

Description

CommunityVault is a simple contract used for storage of an ERC-20 token.

Imports

CommunityVault contract has following imports:

- *Ownable* - from the OpenZeppelin.
- *IERC20* - from the OpenZeppelin.

Inheritance

CommunityVault is Ownable.

Usages

CommunityVault has no usages.

Structs

CommunityVault has no data structures.

Enums

CommunityVault contract has no custom enums.

Events

CommunityVault contract has following custom events:

- *event SetAllowance(address indexed caller, address indexed spender, uint256 amount);*

Modifiers

CommunityVault contract has no custom modifiers.

Fields

CommunityVault contract has following fields and constants:

- *IERC20 private _bond;* - ERC-20 token that is stored on the contract.

Functions

CommunityVault has following functions:

- *constructor*
  Description
  Sets token address.
  Visibility
  public
  Input parameters
  - *address bond* - address of a token that will be stored on the contract

  Constraints
  None
  Events emit
  None
  Output
  None
- *setAllowance*
  Description
  Allows to spend specified tokens sum for by a specified address.
  Visibility
  public
  Input parameters
  - *address spender* - a spender address.
  - *uint amount* - amount of tokens allowed for spending.

  Constraints
  - Can only be called by the contract owner.

  Events emit
  Emits *SetAllowance* event.
  Output
  None

YieldFarm.sol and YieldFarmLP.sol

Description

YieldFarm and *YieldFarmLP* are contracts used to collect rewards for staking.

Imports

YieldFarm and *YieldFarmLP* contracts have following imports:

- SafeMath - from the OpenZeppelin.
- IERC20 - from the OpenZeppelin.
- IStaking - from the project files.

Inheritance

YieldFarm and *YieldFarmLP* do not inherit anything.

Usages

YieldFarm and *YieldFarmLP* have following usages:

- using SafeMath for uint;
- using SafeMath for uint128;

Structs

YieldFarm and *YieldFarmLP* have no data structures.

Enums

YieldFarm and *YieldFarmLP* have no custom enums.

Events

YieldFarm and *YieldFarmLP* have following custom events:

- *event MassHarvest(address indexed user, uint256 epochsHarvested, uint256 totalValue);*
- *event Harvest(address indexed user, uint128 indexed epochId, uint256 amount);*

Modifiers

YieldFarm and *YieldFarmLP* have no custom modifiers.

Fields

YieldFarm contract has following fields and constants:

- *uint constant TOTAL_DISTRIBUTED_AMOUNT = 800000*; - bonus base for each epoch.
- *uint NR_OF_EPOCHS = 24;* - number of epochs.
- *address private _usdc;* - USDC contract address.
- *address private _susd;* - SUSD contract address.
- *address private _dai;* - DAI contract address.

YieldFarmLp contract has following fields and constants:

- *uint constant TOTAL_DISTRIBUTED_AMOUNT = 2000000;* - bonus base for each epoch.
- *uint NR_OF_EPOCHS = 100;* - number of epochs.
- *address private _uniLP;* - uniLP address.

YieldFarm and YieldFarmLp contracts have following common fields:

- *address private _communityVault;* - vault address.
- *IERC20 private _bond;* - reward token address.

- *IStaking private _staking;* - staking contract.
- *uint[] private epochs = new uint[](NR_OF_EPOCHS + 1);* - *epochs pool size.*
- *uint128 public lastInitializedEpoch;* - last initialized epoch.
- *mapping(address => uint128) lastEpochIdHarvested;* - last epochs harvested by an address.
- *uint epochDuration;* - epoch duration.
- *uint epochStart;* - first epoch start timestamp.

Functions

YieldFarm and YieldFarmLp have following public functions:

- ***constructor***
  Description
  Initialized contract. Sets required addresses.
  Visibility
  public
  Input parameters
  - address bondTokenAddress - bond token address.
  - address usdc - USDC address.
  - address susd - SUSD address.
  - address dai - DAI address.
  - address stakeContract - stake contract address.
  - address communityVault - community vault address.

  Constraints
  None
  Events emit
  None
  Output
  None
- ***massHarvest***
  Description
  Harvest rewards for all uncollected epochs.
  Visibility
  external
  Input parameters
  None
  Constraints
  None
  Events emit
  Emits *MassHarvest* event.
  Output
  - *uint* - total harvested sum.
- ***harvest***

Description
Harvest reward for a specified epoch.
Visibility
external
Input parameters
- *uint128 epochId* - epoch id.

Constraints
- *epochId* should be less or equal to *NR_OF_EPOCHS*.
- *epochId* should be less than current epoch.
- Epochs needs to be harvested in order.

Events emit
Emits *Harvest* event.
Output
- *uint* - harvested sum.

- *initEpoch*
  Description
  Init new epoch. Sets pool size value depending on the current sum of the pool on the Staking contract.
  Visibility
  external
  Input parameters
  - uint128 epochId - epoch to init.

  Constraints
  - Epochs can be init only in order.

  Events emit
  None
  Output
  None

- *getPoolSize*
  Description
  Get pool size of a specified epoch.
  Visibility
  external view
  Input parameters
  - *uint128 epochId* - epoch to get pool size for.

  Constraints
  None
  Events emit
  None
  Output
  - *uint* - the pool size.

- *getCurrentEpoch*
  Description
  Get current epoch depending on current time.
  Visibility
  external view

Input parameters
None
Constraints
None
Events emit
None
Output
- *uint* - current epoch.

- *getEpochStake*
  Description
  Get a stake sum of a provided address at a provided epoch.
  Visibility
  external view
  Input parameters
  - address userAddress - an address to get stake balance for.
  - uint128 epochId - an epoch to check.

  Constraints
  None
  Events emit
  None
  Output
  - *uint* - stake sum.
- *userLastEpochIdHarvested*
  Description
  Get last harvested epoch of a message sender.
  Visibility
  external view
  Input parameters
  None
  Constraints
  None
  Events emit
  None
  Output
  - *uint* - last harvested epoch.

Staking.sol

Description

Staking is a contract used to stake tokens. A reward pool increases a result of deposits and decreases as a result of withdrawals.

Imports

Staking contract has following imports:

- IERC20 - from the OpenZeppelin.
- SafeMath - from the OpenZeppelin.
- ReentrancyGuard - from the OpenZeppelin.

Inheritance

Staking is ReentrancyGuard.

Usages

Staking has following usages:

- SafeMath for uint256;

Structs

Staking has following data structures:

- *struct Pool* - stores a reward pool information.
  - *uint256 size;*
  - *bool set;*
- *struct Checkpoint* - stores deposits information.
  - *uint128 epochId;*
  - *uint128 multiplier;*
  - *uint256 startBalance;*
  - *uint256 newDeposits;*

Enums

Staking contract has no custom enums.

Events

Staking contract has following custom events:

- *event Deposit(address indexed user, address indexed tokenAddress, uint256 amount);*
- *event Withdraw(address indexed user, address indexed tokenAddress, uint256 amount);*
- *event ManualEpochInit(address indexed caller, uint128 indexed epochId, address[] tokens);*
- *event EmergencyWithdraw(address indexed user, address indexed tokenAddress, uint256 amount);*

Modifiers

Staking contract has no custom modifiers.

Fields

Staking contract has following fields and constants:

- *uint128 constant private BASE_MULTIPLIER = uint128(1 * 10 ** 18);*
- *uint256 public epoch1Start;* - timestamp of the first epoch.
- *uint256 public epochDuration;* - epoch duration.
- *mapping(address => mapping(address => uint256)) private balances;* - user balances.
- *mapping(address => mapping(uint256 => Pool)) private poolSize;* - pool sizes.
- *mapping(address => mapping(address => Checkpoint[])) private balanceCheckpoints;* - balance chackpoints.
- *mapping(address => uint128) private lastWithdrawEpochId;* - epoch id of a last withdraw of a token.

Functions

Staking has following public functions:

- ***constructor***
  Description
  Sets start timestamp of the first epoch and an epoch duration.
  Visibility
  public
  Input parameters
  - uint256 _epoch1Start - timestamp of the first epoch start.
  - uint256 _epochDuration - an epoch duration.

  Constraints
  None
  Events emit
  None
  Output
  None
- ***deposit***
  Description
  Deposits a specified amount of a specified token. Deposit operations increases pool size.
  Visibility
  public
  Input parameters
  - address tokenAddress - a token address.
  - uint256 amount - an amount to stake.

  Constraints

- nonReentrant modifier.
- Current or previous epoch should be initialized.
- Allowance for a specified amount should be set.

Events emit

Emits *Deposit* event.

Output

None

- ***withdraw***

Description

Withdraw a specified amount of a specified token. Withdraw operations decreases pool size.

Visibility

public

Input parameters

- address tokenAddress - a token address.
- uint256 amount - an amount to withdraw.

Constraints

- nonReentrant modifier.
- A message sender should have at least `*amount*` of tokens on his balance.
- Current or previous epoch should be initialized.

Events emit

Emits *Withdraw* event.

Output

None

- ***manualEpochInit***

Description

Manually init an epoch for a specified list of tokens.

Visibility

public

Input parameters

- address[] memory tokens - an array of tokens to init.
- uint128 epochId - epoch to init.

Constraints

- *epochId* should be less or equal to a current epoch id.
- Epoch should not be initialized for all *tokens*.
- Previous epoch should be initialized for all tokens.

Events emit

Emits *ManualEpochInit* event.

Output

None

- ***emergencyWithdraw***

Description

Withdraw all tokens of a message sender without changes of checkpoints and rewards pool.

Visibility

public

Input parameters

- address tokenAddress - a token to withdraw.

Constraints

- A token should not be withdrawn for at least 10 epochs.

Events emit

Emits *EmergencyWithdraw* event.

Output

None

- *emergencyWithdraw*

Description

Withdraw all tokens of a message sender without changes of checkpoints and rewards pool.

Visibility

public

Input parameters

- address tokenAddress - a token to withdraw.

Constraints

- The token should not be withdrawn for at least 10 epochs.

Events emit

Emits *EmergencyWithdraw* event.

Output

None

- *getEpochUserBalance*

Description

Returns the valid balance of a user that was taken into consideration in the total pool size for the epoch.

Visibility

public view

Input parameters

- address user
- address token
- uint128 epochId

Constraints

None

Events emit

None

Output

- uint256

- *balanceOf*

Description

Returns the amount of `token` that the `user` has currently staked.

Visibility

public view
Input parameters
- address user
- address token

Constraints
None
Events emit
None
Output
- uint256

- *getCurrentEpoch*
  Description
  Returns the id of the current epoch derived from block.timestamp.
  Visibility
  public view
  Input parameters
  None
  Constraints
  None
  Events emit
  None
  Output
  - uint128

- *getEpochPoolSize*
  Description
  Returns the total amount of `tokenAddress` that was locked from beginning to end of epoch identified by `epochId`.
  Visibility
  public view
  Input parameters
  - address tokenAddress
  - uint128 epochId

  Constraints
  None
  Events emit
  None
  Output
  - uint256

- *currentEpochMultiplier*
  Description
  Returns the percentage of time left in the current epoch.
  Visibility
  public view

Input parameters
None
Constraints
None
Events emit
None
Output
- uint128

- *computeNewMultiplier*
Description
Calculates new multiplier that is used in a checkpoint's calculations.
Visibility
public pure
Input parameters
- uint256 prevBalance - previous balance.
- uint128 prevMultiplier - previous multiplier.
- uint256 amount - balance change amount.
- uint128 currentMultiplier - current multiplier.

Constraints
None
Events emit
None
Output
- uint128 - new multiplier.

Vesting.sol

Description

Vesting is a simple contract used for tokens vesting. Total vesting sum is splitted among 100 weeks.

Imports

Vesting contract has following imports:

- *Ownable* - from the OpenZeppelin.
- *IERC20* - from the OpenZeppelin.
- *SafeMath* - from the OpenZeppelin.
- *ReentrancyGuard* - from the OpenZeppelin.

Inheritance

Vesting is Ownable and ReentrancyGuard.

Usages

Vesting has following usages:

- *SafeMath for uint;*

Structs

Vesting has no data structures.

Enums

Vesting contract has no custom enums.

Events

Vesting contract has no custom events.

Modifiers

Vesting contract has no custom modifiers.

Fields

Vesting contract has following fields and constants:

- *uint public constant NUMBER_OF_EPOCHS = 100;*
- *uint public constant EPOCH_DURATION = 604800; // 1 week duration*
- *IERC20 private _bond;*
- *uint public lastClaimedEpoch;*
- *uint private _startTime;*
- *uint public totalDistributedBalance;*

Functions

Vesting has following functions:

- *constructor*
  Description
  Sets owner and token addresses, start time and total distribution sum.
  Visibility
  public
  Input parameters
  - address newOwner
  - address bondTokenAddress

- uint startTime
- uint totalBalance

Constraints
None
Events emit
None
Output
None

- *claim*
  Description
  Claim tokens for a previous period.
  Visibility
  public
  Input parameters
  No
  Constraints
  None
  Events emit
  None
  Output
  None
- *claim*
  Description
  Claim tokens for a previous period.
  Visibility
  public
  Input parameters
  None
  Constraints
  - onlyOwner modifier.

  Events emit
  None
  Output
  None
- *balance*
  Description
  Get current token balance.
  Visibility
  public view
  Input parameters
  None
  Constraints
  None
  Events emit
  None
  Output

    - uint - current token balance.

- *getCurrentEpoch*
  Description
  Get current epoch.
  Visibility
  public view
  Input parameters
  None
  Constraints
  None
  Events emit
  None
  Output
    - uint - current epoch.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Audit overview

Critical

1. *initEpoch* function of the *YieldFarm* contract is not protected and can be called by anyone at any time. If epochs are not initialized on the staking contract, the *YieldFarm* contract state will not be consistent. It's recommended to remove or protect this function.

   Fixed before secondary audit.

2. *initEpoch* function of the *YieldFarmLP* contract is not protected and can be called by anyone at any time. If epochs are not initialized on the staking contract, the *YieldFarmLP* contract state will not be consistent. It's recommended to remove or protect this function.

   Fixed before secondary audit.

High

No high severity issues were found.

Medium

1. Users will not be able to *deposit* to or *withdraw* from the *Staking* contract without manual calling of the *manualEpochInit* function if the token has not been initialized yet. This may complicate user experience.

Low

No low severity issues were found.

Lowest / Code style / Best Practice

1. It's recommended to move some logic of the Staking contract to separate function to increase code readability.
2. Explicitly mark visibility of fields in the YieldFarm contracts.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Conclusion

Smart contracts within the scope was 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 critical and 1 medium severity issues during the audit. The Customer did not provide any requirements or specifications so we cannot validate the correctness of all calculations. The code is well-tested and most of possible cases are covered.

Note: All critical issues have been fixed before the secondary audit.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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 own vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee explicit security of the audited smart contracts.