SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: BarnBridge
Date: May 21st, 2021

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 BarnBridge - Initial Audit
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Yield
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Git Repository	https://github.com/BarnBridge/BarnBridge-SmartYieldBonds
Timeline	19 MAY 2021 - 21 MAY 2021
Changelog	21 MAY 2021 - INITIAL AUDIT

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

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by BarnBridge (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 on May 21st, 2021.

Scope

The scope of the project is the list of smart contracts of next Git Repository:

```
https://github.com/BarnBridge/BarnBridge-SmartYieldBonds
BarnBridge-SmartYieldBonds-master/contracts/providers/IAaveCumulator.sol
BarnBridge-SmartYieldBonds-master/contracts/providers/ICreamCumulator.sol
BarnBridge-SmartYieldBonds-master/contracts/model/BondModelV2Compounded.sol
BarnBridge-SmartYieldBonds-master/contracts/model/BondModelV2Linear.sol
BarnBridge-SmartYieldBonds-master/contracts/model/ABondModelV2.sol
BarnBridge-SmartYieldBonds-master/contracts/providers/AaveController.sol
BarnBridge-SmartYieldBonds-master/contracts/providers/CreamController.sol
BarnBridge-SmartYieldBonds-master/contracts/providers/AaveProvider.sol
BarnBridge-SmartYieldBonds-master/contracts/providers/CreamProvider.sol
```

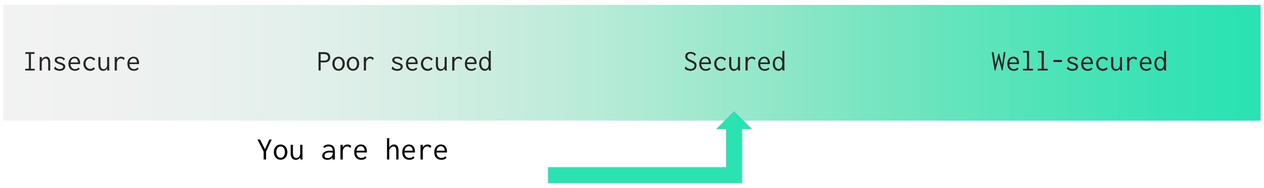
We have scanned these smart contracts for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

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 ▪ Asset's integrity ▪ User Balances manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Executive Summary

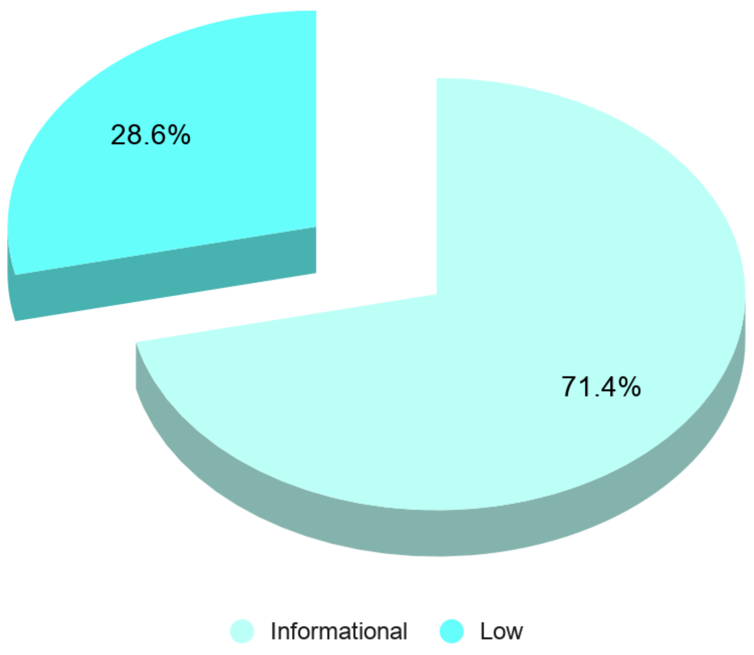
According to the assessment, the Customer's smart contracts are secured but could be improved



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. All found issues can be found in the Audit overview section.

Security engineers found 2 low and 5 informational issues during the first review.

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
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

Audit overview

■ ■ ■ ■ Critical

No Critical severity issues were found.

■ ■ ■ High

No High severity issues were found.

■ ■ Medium

No Medium severity issues were found.

■ Low

1. Vulnerability: No event on access control change

methods setDao(address), setGuardian(address), setController(address) changes important addresses, but have no events, so it is difficult to track changes off-chain

Recommendation: Please emit an event like DaoTransferred(address _newDAO), GuardianTransferred(address _newGuardian), etc.

Lines: Governed.sol#32-44

```
function setDao(address dao_)
 external
 onlyDao
{
 dao = dao_;
}

function setGuardian(address guardian_)
 external
 onlyDao
{
 guardian = guardian_;
}
```

Lines: providers/CreamProvider.sol#106-111

```
function setController(address newController_)
 external override
 onlyControllerOrDao
{
 controller = newController_;
}
```

2. Vulnerability: No event on financials/arithmetic change

   methods setHarvestCost(uint256), setBondMaxRatePerDay(uint256) changes important data, like cost, rate, but have no events, so it is difficult to track changes off-chain

   Recommendation: Please emit an event like HarvestCostUpdated(uint256 _newCost), BondMaxRatePerDayUpdated(uint256 _newRate), etc.

   Lines: IController.sol#44-53

```
function setHarvestCost(uint256 newValue_)
 public
 onlyDao
{
   require(
     HARVEST_COST < EXP_SCALE,
     "IController: HARVEST_COST too large"
   );
   HARVEST_COST = newValue_;
}
```

   Lines: IController.sol#55-60

```
function setBondMaxRatePerDay(uint256 newVal_)
 public
 onlyDao
{
 BOND_MAX_RATE_PER_DAY = newVal_;
}
```

Lowest / Code style / Best Practice

1. Vulnerability: Conformance to solidity naming convention

   State variable names like MAX_POOL_RATIO are not mixedCase, which is recommended for state variables. UPPER_CASE_WITH_UNDERSCORES is recommended for using with constants.

   Recommendation: Please follow solidity naming convention.

   Lines: model/ABondModelV2.sol#16

```
uint256 public MAX_POOL_RATIO = 750 * 1e15; // 75%
```

   Lines: IController.sol#26-42

```
uint256 public HARVEST_COST = 40 * 1e15; // 4%
```

```
// fee for buying jTokens
uint256 public FEE_BUY_JUNIOR_TOKEN = 3 * 1e15; // 0.3%

// fee for redeeming a sBond
uint256 public FEE_REDEEM_SENIOR_BOND = 100 * 1e15; // 10%

// max rate per day for sBonds
uint256 public BOND_MAX_RATE_PER_DAY = 719065000000000; // APY 30% /
year

// max duration of a purchased sBond
uint16 public BOND_LIFE_MAX = 90; // in days

bool public PAUSED_BUY_JUNIOR_TOKEN = false;

bool public PAUSED_BUY_SENIOR_BOND = false;
```

Lines: model/ABondModelV2.sol#16

```
uint256 public MAX_POOL_RATIO = 750 * 1e15; // 75%
```

Lines: model/ABondModelV2.sol#16

```
uint256 public MAX_POOL_RATIO = 750 * 1e15; // 75%
```

Lines: model/ABondModelV2.sol#16

```
uint256 public MAX_POOL_RATIO = 750 * 1e15; // 75%
```

2. Vulnerability: Boolean equality

   Boolean constants can be used directly and do not need to be compared to true or false.

   Lines: providers/CreamProvider.sol#93-96

```
require(
 false == _setup,
 "CrP: already setup"
);
```

3. Vulnerability: Too many digits

   Literals with many digits are difficult to read and review.

   Recommendation: Please consider replacing zeros by ether units and/or scientific notation and/or add dash separators for better readability

Lines: IController.sol#35

```
uint256 public BOND_MAX_RATE_PER_DAY = 719065000000000; // APY 30% /
year
```

4. Vulnerability: View function that could be declared pure

   View functions that are never access state variables should be declared pure

   Recommendation: It’s okay to define a constant instead of creating pure function for this. Public constants are callable externally like functions.

   Example:

```
uint256 public constant spotDailyDistributionRateProvider = 0;
```

Lines: providers/AaveController.sol#162-167

```
function spotDailyDistributionRateProvider()
 public view returns (uint256)
{
 // kept for backwards compat
 return 0;
}
```

5. Vulnerability: Public function that could be declared external

   public functions that are never called by the contract should be declared external to save gas.

Lines: model/ABondModelV2.sol#20-23

```
function setMaxPoolRatio(uint256 newMaxPoolRatio_)
 public
 onlyDao
{
```

Lines: model/ABondModelV2.sol#27-32

```
function maxDailyRate(
 uint256 total_,
 uint256 loanable_,
 uint256 dailyRate_
)
 public view
```

```
returns (uint256)
{
```

Lines: model/BondModelV2Compounded.sol#14-23

```
function gain(
   uint256 total_,
   uint256 loanable_,
   uint256 dailyRate_,
   uint256 principal_,
   uint16 forDays_
)
 public view override
returns (uint256)
{
```

Lines: model/BondModelV2Linear.sol#14-23

```
function gain(
   uint256 total_,
   uint256 loanable_,
   uint256 dailyRate_,
   uint256 principal_,
   uint16 forDays_
)
 public view override
returns (uint256)
{
```

Lines: IController.sol#44-47

```
function setHarvestCost(uint256 newValue_)
 public
 onlyDao
{
```

Lines: IController.sol#55-58

```
function setBondMaxRatePerDay(uint256 newVal_)
 public
 onlyDao
{
```

Lines: IController.sol#62-65

```
function setBondLifeMax(uint16 newVal_)
 public
 onlyDao
{
```

Lines: IController.sol#69-72

```
function setFeeBuyJuniorToken(uint256 newVal_)
 public
 onlyDao
{
```

Lines: IController.sol#76-79

```
function setFeeRedeemSeniorBond(uint256 newVal_)
 public
 onlyDao
{
```

Lines: IController.sol#83-86

```
function setPaused(bool buyJToken_, bool buySBond_)
 public
 onlyDaoOrGuardian
{
```

Lines: IController.sol#91-94

```
function setOracle(address newVal_)
 public
 onlyDao
{
```

Lines: IController.sol#98-101

```
function setBondModel(address newVal_)
 public
 onlyDao
{
```

Lines: IController.sol#105-108

```
function setFeesOwner(address newVal_)
 public
 onlyDao
{
```

Lines: IController.sol#112-115

```
function yieldControllTo(address newController_)
 public
 onlyDao
{
```

Lines: providers/CreamController.sol#72-75

```
function harvest(uint256)
 public
```

```
returns (uint256 rewardAmountGot, uint256 underlyingHarvestReward)
{
```

Lines: providers/CreamController.sol#96-99

```
function providerRatePerDay()
 public override virtual
returns (uint256)
{
```

Lines: providers/CreamController.sol#163-165

```
function spotDailyDistributionRateProvider()
 public pure returns (uint256)
{
```

Lines: providers/AaveController.sol#74-77

```
function harvest(uint256)
 public
returns (uint256 rewardAmountGot, uint256 underlyingHarvestReward)
{
```

Lines: providers/AaveController.sol#102-105

```
function providerRatePerDay()
 public override virtual
returns (uint256)
{
```

Lines: providers/AaveController.sol#162-164

```
function spotDailyDistributionRateProvider()
 public view returns (uint256)
{
```

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools.

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

Security engineers found 2 low and 5 informational issues during the first review.

Category	Check Items	Comments
➔ Code Review	➔ Style guide violation	➔ public function that could be declared external ➔ view function that could be declared pure ➔ boolean equality ➔ too many digits ➔ solidity naming convention
➔ Functional review	➔ Operation Trails & Event Generation	➔ No event on access control change ➔ No event on financials/arithmetic change

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 the 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.