28th JANUARY 2021

BARNBRIDGE

SMART CONTRACT AUDIT REPORT

-

version v2.0

Smart Contract Security Audit and General Analysis

HAECHI AUDIT

Table of Contents

1 Issues (0 Critical, 0 Major, 1 Minor) Found

About HAECHI AUDIT

HAECHI AUDIT is a global leading smart contract security audit and development firm operated by HAECHI LABS. HAECHI AUDIT consists of professionals with years of experience in blockchain R&D and provides the most reliable smart contract security audit and development services.

So far, based on the HAECHI AUDIT's security audit report, our clients have been successfully listed on the global cryptocurrency exchanges such as Huobi, Upbit, OKEX, and others.

Our notable portfolios include SK Telecom, Ground X by Kakao, and Carry Protocol while HAECHI AUDIT has conducted security audits for the world's top projects and enterprises.

Trusted by the industry leaders, we have been incubated by Samsung Electronics and awarded the Ethereum Foundation Grants and Ethereum Community Fund.

Contact : audit@haechi.io
Website : audit.haechi.io

01. Introduction

This report was written to provide a security audit for the BarnBridge smart contract. HAECHI AUDIT conducted the audit focusing on whether BarnBridge smart contract is designed and implemented in accordance with publicly released information and whether it has any security vulnerabilities.

The issues found are classified as CRITICAL, MAJOR, MINOR or TIPS according to their severity.

CRITICAL	Critical issues are security vulnerabilities that MUST be addressed in order to prevent widespread and massive damage.
MAJOR	Major issues contain security vulnerabilities or have faulty implementation issues and need to be fixed.
MINOR	Minor issues are some potential risks that require some degree of modification.
TIPS	Tips could help improve the code's usability and efficiency

HAECHI AUDIT advises addressing all the issues found in this report.

02. Summary

The code used for the audit can be found at GitHub

- https://github.com/BarnBridge/BarnBridge-DAO
  - Commit Hash : e134311671d698359f9a2f8898bc96b6a84146d5
- https://github.com/BarnBridge/BarnBridge-Barn
  - Commit Hash : 3a0f8de8750d1642cfc5bd4cb319c50cb35f0bb5
  - v2.0 Hash: 0166325b2b70982ab901e20d958d2153ea65c458

Issues

HAECHI AUDIT has 0 Critical Issues, 0 Major Issues, and 1 Minor Issue; also, we included 0 Tip category that would improve the usability and/or efficiency of the code.

Severity	Issue	Status
MINOR	BarnFacet#votingPower() returns inconsistent values for locked bonds .	(Found - v1.0) (Intended - v2.0)
Notice	Even after voting for the proposal is completed, the proposer of the Proposal can block the execution by using Governance#cancelProposal().	(Found - v1.0) (Resolved - v2.0)
Notice	LibOwnership#onlyOwner modifier is implemented but not used.	(Found - v1.0) (Acknowledged - v2.0)

Update

[v2.0] - Barnbridge has confirmed that 1 issue and 1 notice intended, and 1 notice is resolved.

03. Overview

Contracts Subject to Audit

- Barn.sol
- Reward.sol
- Parameters.sol
- Bridge.sol
- Governance.sol
- facets
    - BarnFacet.sol
    - DiamondCutFacet.sol
    - DiamondLoupeFacet.sol
    - OwnershipFacet.sol
- libraries
    - LibBarnStorage.sol
    - LibDiamond.sol
    - LibDiamondStorage.sol
    - LibOwnership.sol

Roles

The BarnBridge Smart contract has the following authorizations:

- **ContractOwner**
- **DAO**

The features accessible by each level of authorization is as follows:

Role	Functions

ContractOwner	• BarnFacet
	  ○ initBarn()
	• DiamondCutFacet
	  ○ diamondCut()
	• OwnershipFacet
	  ○ transferOwnership()

DAO	• Parameters ○ setWarmUpDuration()

	○ setActiveDuration() ○ setQueueDuration() ○ setGracePeriodDuration() ○ setAcceptanceThreshold() ○ setMinQuorum()

Notice

- **Even after voting for the proposal is completed, the proposer of the Proposal can block the execution by using Governance#cancelProposal().**

  According to the implemented code, the proposer of the proposal can cancel it's execution even if it is determined by voting.

  When other users have already agreed to the execution of that proposal through their voting power, it is considered incorrect for a particular user to cancel the execution of the proposal.

  ### Update

  BarnBridge team has fixed this issue by not giving privilege to cancel transactions after agreement.

- **LibOwnership#onlyOwner modifier is implemented but not used.**

  The modifier and LibOwnership#enforceIsContractOwner() have the same role, so LibOwnership#onlyOwner is not used in the contract.

  Therefore, this implementation was excluded from audit coverage.

  ### Update

  BarnBridge team has confirmed that this issue is acknowledged.

04. Issues Found

MINOR : BarnFacet#votingPower() returns inconsistent values for locked bonds . (Found - v.1.0) (Intended - v2.0)

MINOR

```
181.    function votingPower(address user) public view returns (uint256) {
182.        return votingPowerAtTs(user, block.timestamp);
183.    }
184.
185.    // votingPowerAtTs returns the voting power (bonus included) + delegated voting power for a user
        at a point in time
186.    function votingPowerAtTs(address user, uint256 timestamp) public view returns (uint256) {
187.        LibBarnStorage.Stake memory stake = stakeAtTs(user, timestamp);
188.
189.        uint256 ownVotingPower;
190.
191.        // if the user delegated his voting power to another user, then he doesn't have any voting power
        left
192.        if (stake.delegatedTo != address(0)) {
193.            ownVotingPower = 0;
194.        } else {
195.            uint256 balance = stake.amount;
196.            uint256 multiplier = _stakeMultiplier(stake, timestamp);
197.            ownVotingPower = balance.mul(multiplier).div(BASE_MULTIPLIER);
198.        }
199.
200.        uint256 delegatedVotingPower = delegatedPowerAtTs(user, timestamp);
201.
202.        return ownVotingPower.add(delegatedVotingPower);
203.    }
204.
```

Problem Statement

BarnFacet#votingPower() returns votingPower owned by the user.

If the bond of the user is locked, voting power will be recalculated by line 197.

However, even if the bond is locked, if it is delegated to another address, the recalculation process mentioned above will not be performed.

So, when the user with locked bond delegate his voting power using BarnFacet#delegate() function, the voting power obtained through the bond is calculated differently from before being delegated.

Recommendation

If the delegated bond is locked, calculate the voting power in the same way as the non-delegated bond.

If the above is intended behavior during the development process, there is no need to modify it.

Update

[v2.0] - BarnBridge team has confirmed that this is intended behavior

05. Disclaimer

This report is not an advice on investment, nor does it guarantee adequacy of a business model and/or a bug-free code. This report should be used only to discuss known technical problems. The code may include problems on Ethereum that are not included in this report. It will be necessary to resolve addressed issues and conduct thorough tests to ensure the safety of the smart contract.

Appendix A. Test Results

The following are the results of a unit test that covers the major logics of the smart contract under audit. The parts in red contain issues and therefore have failed the test.

```
BarnFacet
 initBarn()
   ✓ should fail when already initialized (44ms)
   ✓ should fail when msg.sender is not diamondStroage.contractOwner
   ✓ should initialize barn
 deposit()
   ✓ should fail when deposit amount is 0
   ✓ should fail when deposit amount is smaller than allownance
  valid case
    ✓ should update user's history.timestamp
    ✓ should update user's history.value
    ✓ should decrease user's token balance
    ✓ should increase delegatePower
 lock()
   ✓ should fail when lock timestamp is already passed
   ✓ should fail when lock timestamp is over MAX LOCK
   ✓ should fail when sender do not have balance
  valid case
    ✓ should store appropriate lock expiryTimestamp
    ✓ should store appropriate lock amount
 withdraw()
   ✓ should fail when lock amount is zero
   ✓ should fail when not unlocked yet
   ✓ should fail when amount is larger than balance
  valid case
    ✓ should update user's history.timestamp
    ✓ should update user's history.value
    ✓ should increase user's token balance
    ✓ should decrease delegatePower if there is delegated
 depositAndLock()
   ✓ should update user's history.timestamp
   ✓ should update user's history.value
   ✓ should decrease user's token balance
   ✓ should store appropriate lock expiryTimestamp
   ✓ should store appropriate lock amount
 delegate()
```

```
  ✓ should fail when delegete itself
  ✓ should fail when there is not sender's balance
  valid case
    ✓ should increase delegatePower of to when there was no delegateTo
    ✓ should increase delegatePower of to when there was delegateTo (172ms)
stopDelegate()
  ✓ should reset delegatePower (47ms)
votingPower()
  ✓ should return voting power when user do not have delegatedTo
  ✓ should return 0 when user have delegatedTo (50ms)
  1) should return same voting power when it is delegated (132ms)
multiplierOf()
  ✓ should return BASE_MULTIPLIER when sufficient time passed
  ✓ should return appropriate multiplier when time not passed yet
userLockedUntil()
  ✓ should return unlock time
updataUserBalance()
  ✓ should change bondstake amount
updateDelegatedTo()
  ✓ should change bondstake amount
updateLockedBond()
  ✓ should change bondstake amount
stakeAtTs()
  ✓ should return checkpoints[max]
  ✓ should return checkpoints[mid] (46ms)
  ✓ should return checkpoints[min]
bondStakedAtTs()
  ✓ should return checkpoints[max]
  ✓ should return checkpoints[mid] (43ms)
  ✓ should return checkpoints[min]
delegatedPowerAtTs()
  ✓ should return checkpoints[max] (129ms)
  ✓ should return checkpoints[mid] (43ms)
  ✓ should return checkpoints[min]

DiamondCut
  _handleAddCut()
    ✓ should fail when function already added
    valid case
      ✓ should add cut.facetAddress to ds
      ✓ should add cut.facetSelector to ds
      ✓ should add selector info to ds
```

```
_handleRemoveCut()
  ✓ should fail when function is not added
  ✓ should fail when facet is address(this) (45ms)
  valid case
    ✓ should delete cut to ds when facet is last facet (91ms)
    ✓ should delete cut to ds when facet is not last facet (92ms)
_handleReplaceCut()
  ✓ should fail when facet is address(this) (46ms)
  ✓ should fail when facet address is same (42ms)
  ✓ should fail when function is not added
  valid case
    ✓ should replace cut.facetAddress to ds
    ✓ should replace cut.facetSelector to ds
    ✓ should replace selector info to ds
initializeDiamondCut()
  ✓ should fail if calldata is not empty when _init is ZERO_ADDRESS
  ✓ should fail if calldata is empty when _init is not ZERO_ADDRESS
  ✓ should fail if _init is empty contract
  ✓ should fail when inefficient calldata
  -
executeDiamondCut()
  ✓ should fail when functionSelectors is empty
  ✓ should fail when action is not 0, 1, 2
  CASE : action is 0 (add)
    ✓ should fail when facet address is zero
    ✓ should fail when facet address does not have code
    valid case
      ✓ should add cut.facetAddress to ds
      ✓ should add cut.facetSelector to ds
      ✓ should add selector info to ds
  CASE : action is 1 (replace)
    ✓ should fail when facet address is zero
    ✓ should fail when facet address does not have code
    valid case
      ✓ should replace cut.facetAddress to ds
      ✓ should replace cut.facetSelector to ds
      ✓ should replace selector info to ds
  CASE : action is 2 (remove)
    ✓ should fail when facet address is zero
    valid case
      ✓ should delete cut to ds when facet is last facet (88ms)
```

```
DiamondCutFacet
  facets()
    ✓ should return appropriate facetAddress (60ms)
    ✓ should return appropriate functionSelectors (118ms)
  facetFunctionSelectors()
    ✓ should return appropriate functionSelectors
  facetAddresses()
    ✓ should return appropriate facetAddress (43ms)
  facetAddresses()
    ✓ should return appropriate facetAddress
  facetAddresses()
    ✓ should return appropriate facetAddress

OwnershipFacet
  transferOwnership()
    ✓ should fail when msg.sender is not contractOwner
    ✓ should transfer ownership

Reward
  constructor()
    ✓ should transfer ownership
    ✓ should set rewardToken
    ✓ should set barn
  ackFunds()
    ✓ just return when contract do not have token balance
    valid case
      ✓ should change balanceBefore
      ✓ should change currentMultiplier
  setupPullToken()
    ✓ should fail when msg.sender is not owner
    valid case
      ✓ should set appropriate pull.source
      ✓ should set appropriate pull.startTs
      ✓ should set appropriate pull.endTs
      ✓ should set appropriate pull.totalDuration
      ✓ should set appropriate pull.totalAmount
      ✓ should set appropriate lastPullTs
  setBarn()
    ✓ should fail when msg.sender is not owner
    ✓ should set barn
  pullToken()
    ✓ just return when source is zeroAddress
```

```
    ✓ just return when startTime is later than now
    valid case
      ✓ should transfer pull amount

 Contract: Governance
   #initialize()
     ✓ Should fail if already initialized (121ms)
     Valid Case
       ✓ Should store barn
       ✓ Should store isInitialized (44ms)
   active()
     ✓ should fail when already activated (296ms)
     ✓ should fail when bondStake over threshold (161ms)
     valid case
       ✓ should activate (321ms)
   #propose()
     ✓ Should fail if not activated (589ms)
     ✓ Should fail if proposer's voting power is too low (960ms)
     ✓ Should fail if array's length is not same (3428ms)
     ✓ Should fail if target length is 0 (840ms)
     ✓ Should fail if target length is too large (2306ms)
     Valid Case
       ✓ Should store new proposal (3857ms)
       ✓ Should change lasProposalId (69ms)
       CASE : when msg.sender already has proposal
         ✓ Should fail if past proposal is in Active state (1135ms)
         ✓ Should fail if past proposal is in ReadyForActivation state (935ms)
         ✓ Should fail if past proposal is in WarmUp state (806ms)
   #state()
     ✓ Should fail if lastProposalId >= proposalId (99ms)
     ✓ Should fail if proposalId > 0 (74ms)
     Valid Case
       State Warm_UP
         ✓ Should be warm up when block.timestamp <= proposal.createTime + WARM_UP (264ms)
       State Active
         ✓ Should be activate up when block.timestamp <= proposal.createTime + WARM_UP+
ActiveDuration (197ms)
       State Failed
         ✓ Should be failed when forVote under 0.6 (868ms)
       State Accepted
         ✓ Should be Accepted when accept condition (988ms)
       State Queued
```

```
    ✓ Should be Queued when queue condition (1899ms)
  State Executed
    ✓ Should be Executed when execute condition (2648ms)
  State Grace
    ✓ Should be Grace when grace condition (1953ms)
  State Expired
    ✓ Should be Expired when expire condition (1885ms)
#castVote()
  ✓ Should fail if state is not Active (501ms)
  ✓ Should fail if already voted (767ms)
  CASE : startVote first when not started
    ✓ Should store vote result (386ms)
    ✓ Should store receipt data (600ms)
  Valid Case
    ✓ Should store vote result (272ms)
    ✓ Should store receipt data (746ms)
    ✓ Should emit Vote event
  Change the vote
    ✓ Shold change vote result (242ms)
    ✓ Should change receipt data (683ms)
#cancelVote()
  ✓ Should fail if state is not Active (629ms)
  ✓ Should fail if not voted yet (523ms)
  Valid Case
    ✓ Should store vote result (421ms)
    ✓ Should store receipt data (998ms)
    ✓ Should emit VoteCanceled event
#queue()
  ✓ Should fail if state is not Active (2500ms)
  ✓ Should fail if proposal canceled (1462ms)
  Valid Case
    ✓ Should store eta (227ms)
    ✓ Should emit ProposalQueued event
#execute()
  ✓ Should fail if state is not Queued or Grace (1444ms)
  ✓ Should fail if msg.sender is not guardian (1988ms)
  Valid Case
    ✓ Should emit ProposalExecuted event
#cancelProposal()
  ✓ Should fail if state is Executed (2417ms)
  ✓ Should fail if state is Failed (982ms)
  ✓ Should fail if state is Expired (1872ms)
```

```
    ✓ Should fail if msg.sender is not proposer (2627ms)
  Valid Case
    ✓ Should store proposal.canceled to true (251ms)
    ✓ Should emit ProposalCanceled event
getActions()
  ✓ Should return appropriate target (1287ms)
  ✓ Should return appropriate value (1196ms)
  ✓ Should return appropriate signatures (1211ms)
  ✓ Should return appropriate calldatas (1416ms)
#startCancellationProposal()
  ✓ Should fail if not Queued (318ms)
  ✓ Should fail if proposer's voting power is too low (2255ms)
  Valid Case
    ✓ Should store new cp (143ms)
#executeCancellationProposal()
  ✓ Should fail if state is canceled (574ms)
  Valid Case
    ✓ Should store proposal.canceled to true (226ms)
    ✓ Should emit CancellationProposalExecuted event
#voteCancellationProposal()
  ✓ Should fail if not startCacellation (355ms)
  ✓ Should fail if already voted (1086ms)
  CASE : startVote first when not started
    ✓ Should store vote result (161ms)
    ✓ Should store receipt data (512ms)
  Valid Case
    ✓ Should store vote result (111ms)
    ✓ Should store receipt data (439ms)
    ✓ Should emit CancellationProposalVote event
    Change the vote
      ✓ Shold change vote result (221ms)
      ✓ Should change receipt data (490ms)
#cancelVoteCancellationProposal()
  ✓ Should fail if invalid proposal ID (148ms)
  ✓ Should fail if not startCacellation (353ms)
  ✓ Should fail if not voted yet (730ms)
  Valid Case
    ✓ Should store vote result (247ms)
    ✓ Should store receipt data (448ms)
    ✓ Should emit CancellationProposalVoteCancelled event
#getProposalQuorum()
  ✓ Should fail when invalid proposal ID (169ms)
```

```
    ✓ should return appropriate propolsaQuorum (350ms)

  Contract: Paremeters
    #setWarmUpPeriod()
      ✓ Should set WARM_UP (106ms)
    #setActivePeriod()
      ✓ Should set ACTIVE (101ms)
    #setQueuePeriod()
      ✓ Should set QUEUE (124ms)
    #setGracePeriod()
      ✓ Should set GRACE_PERIOD (487ms)
    #setMinimumThreshold()
      ✓ Should fail when threshold over 100 (131ms)
      ✓ hould fail when threshold under 50 (464ms)
      ✓ Should set MINIMUM_FOR_VOTES_THRESHOLD (384ms)
    #setMinQuorum()
      ✓ Should fail when quorum over 100 (148ms)
      ✓ Should set MINIMUM_QUORUM (228ms)
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/					
Barn.sol	**100**	**100**	**100**	**100**	
Rewards.sol	**100**	**100**	**100**	**100**	
Parameters.sol	**100**	**100**	**100**	**100**	
Bridge.sol	**100**	**100**	**100**	**100**	
Governance.sol	**100**	**100**	**100**	**100**	
Parameters.sol	**100**	**100**	**100**	**100**	
contracts/facets/					
BarnFacet.sol	**100**	**100**	**100**	**100**	
DiamondCutFacet.sol	**100**	**100**	**100**	**100**	
DiamondLoupeFacet.sol	**100**	**100**	**100**	**100**	
OwnershipFacet.sol	**100**	**100**	**100**	**100**	
contracts/libraries/					
LibBarnStorage.sol	**100**	**100**	**100**	**100**	

LibDiamond.sol	**100**	**100**	**100**	**100**	
LibDiamondStorage.sol	**100**	**100**	**100**	**100**	
LibOwnership.sol	**85.71**	**50**	**75**	**75**	**30,31**

[Table 1] Test Case Coverage