SMART CONTRACT AUDIT

ZOKYO.

Jun 17, 2021 | v. 1.0

PASS

Zokyo's Security Team has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

TECHNICAL SUMMARY

This document outlines the overall security of the Bridge Mutual smart contracts, evaluated by Zokyo's Blockchain Security team.

The scope of this audit was to analyze and document the Bridge Mutual smart contract codebase for quality, security, and correctness.

Contract Status

There were no critical issues found during the audit.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Zokyo recommend that the Bridge Mutual team put in place a bug bounty program to encourage further and active analysis of the smart contract.

TABLE OF CONTENTS

AUDITING STRATEGY AND TECHNIQUES APPLIED

The Smart contract's source code was taken from the Bridge Mutual repository – https://github.com/Bridge-Mutual/bridgemutual-core/tree/.
Last commit – 6fa471de9330743512d55cac63d7613a06d13c1e.

Requirements:

- **Whitepaper:** https://uploads-ssl.webflow.com/5fac3e348dbd5932a7578690/60a7ffd8a8874fc955e580ac_Bridge%20Mutual%20WP%20v1.pdf
- **Pitch Deck:** https://uploads-ssl.webflow.com/5fac3e348dbd5932a7578690/6004108498ed9eacd55c3807_BridgeMutual%20BP%201.85.pdf
- **Token Economics:** https://uploads-ssl.webflow.com/5fac3e348dbd5932a7578690/600ecf0d85f4ea7616b1fe74_Bridge%20Economics%20-%20Final.pdf

Throughout the review process, care was taken to ensure that the token contract:

- Implements and adheres to existing Token standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of gas, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Whether the code meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of Bridge Mutual smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite using the Truffle testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

1	Due diligence in assessing the overall code quality of the codebase.	3	Testing contract logic against common and uncommon attack vectors.
2	Cross-comparison with other, similar smart contracts by industry leaders.	4	Thorough, manual review of the codebase, line-by-line.

EXECUTIVE SUMMARY

There were no critical issues found during the audit. All the mentioned findings may have an effect only in the case of specific conditions performed by the contract owner.

This audit certifies that all issues found by the Consensys Diligence team were successfully fixed. Zokyo auditing team has found a couple of additional issues that were not raised previously. All of them were resolved by Bridge Mutual developers so they won't bear any risk for the end-user.

The contracts are quite complex and sophisticated. Despite all known issues and vulnerabilities being fixes as of now, Zokyo auditing team can not guarantee that all issues are found as the contracts are still being developed by the team. Hence, we at Zokyo recommend to run additional audits at the final points of development.

Contracts are well written and structured. The findings during the audit have no impact on contract performance or security, so it is fully production-ready.

STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the ability of the contract to compile or operate in a significant way.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

MANUAL REVIEW

HIGH | RESOLVED

Contract PolicyBook is not letting to withdraw available liquidity by liquidity provider, even if that is available.

Snippet:

```
function getAvailableDAIXWithdrawalAmount(address _userAddr)
  external
  view
  override
  returns (uint256)
{
  uint256 allCover = totalCoverTokens.add(_getUserAvailableDAI(_userAddr));

  return totalLiquidity > allCover ? totalLiquidity - allCover : 0;
}
```

Recommendation:
Available liquidity should be calculated as totalLiquidity substracing totalCoverTokens.

HIGH | RESOLVED

PolicyBook is incorrectly plans premium distribution in case if distribution was not executed for more than 91 days.

Snippet:

```
function _distributePremiums() internal {
  uint256 currentEpoch = _getDistributionEpoch();
  uint256 lastEpoch = lastPremiumDistributionEpoch;

  if (currentEpoch > lastEpoch) {
    int256 currentDistribution = lastPremiumDistributionAmount;
    uint256 newTotalLiquidity = totalLiquidity;
    uint256 distributionEpoch =
      Math.min(currentEpoch, lastEpoch + MAX_PREMIUM_DISTRIBUTION_EPOCHS + 1);

    for (uint256 i = lastEpoch + 1; i <= distributionEpoch; i++) {
      currentDistribution += premiumDistributionDeltas[i];
      newTotalLiquidity = newTotalLiquidity.add(uint256(currentDistribution));
    }

    lastPremiumDistributionAmount = currentDistribution;
    lastPremiumDistributionEpoch = currentEpoch;
    totalLiquidity = newTotalLiquidity;
  }
}
```

Recommendation:

Set lastPremiumDistributionEpoch to last epoch of premiums distribution.

HIGH | RESOLVED

Policy holder is not able to make appeal claim if initial voting took time more than 2 weeks after ending of cover.

Snippet #1:

```
function _submitClaimAndInitializeVoting(address claimer, bool appeal) internal {
  require(policyRegistry.isPolicyActive(claimer, address(this)), "PB: Policy is not active");

  claimVoting.initializeVoting(
    claimer,
    address(this),
    policyHolders[claimer].coverTokens,
    policyHolders[claimer].payed.mul(PROTOCOL_PERCENTAGE).div(PERCENTAGE_100),
    appeal
  );
}
```

Snippet #2:

```
function isPolicyActive(address _userAddr, address _policyBookAddr)
  public
  view
  override
  returns (bool)
{
  uint256 endTime = policyInfos[_userAddr][_policyBookAddr].endTime;

  return endTime == 0 ? false : endTime.add(STILL_CLAIMABLE_FOR) > block.timestamp;
}
```

Recommendation:
Let Policy holder to make new appeal claims for specific period after declining previous claim.

MEDIUM | UNRESOLVED

Contract LiquidityMining accepts any EIP1155 token transfer, but knows how to work only with liquidity mining NFT contract.

Snippet:

```
function onERC1155Received(
  address operator,
  address from,
  uint256 id,
  uint256 value,
  bytes calldata data
) external pure override returns (bytes4) {
  return 0xf23a6e61;
}

function onERC1155BatchReceived(
  address operator,
  address from,
  uint256[] calldata ids,
  uint256[] calldata values,
  bytes calldata data
) external pure override returns (bytes4) {
  return 0xbc197c81;
}
```

Recommendation:
Add validation that token transfer is done only from liquidity mining NFT contract.

MEDIUM | RESOLVED

Method upgradePolicyBooks of PolicyBookAdmin contract is passing implementation address without validation of it.

Snippet:

```
function upgradePolicyBooks(
  address policyBookImpl,
  uint256 offset,
  uint256 limit
) external onlyOwner {
  _setPolicyBookImplementation(policyBookImpl);

  address[] memory _policies = policyBookRegistry.list(offset, limit);

  uint256 to = (offset.add(limit)).min(_policies.length).max(offset);

  for (uint256 i = offset; i < to; i++) {
    upgrader.upgrade(_policies[i], policyBookImpl);
  }
}
```

Recommendation:
Verify that passed policy book implementation address is valid contract address.

MEDIUM | RESOLVED

Method updateEpochsInfo of contract PolicyBook is vulnerable to `Out of gas` error in certain circumstances.

Snippet:

```
function updateEpochsInfo() public override {
  uint256 _countOfPassedEpoch = block.timestamp.sub(epochStartTime).div(EPOCH_DURATION);

  uint256 _newTotalCoverTokens = totalCoverTokens;
  uint256 _newEpochNumber = _countOfPassedEpoch + 1;

  for (uint256 i = currentEpochNumber; i < _newEpochNumber; i++) {
    _newTotalCoverTokens = _newTotalCoverTokens.sub(epochAmounts[i]);
    delete epochAmounts[i];
  }

  currentEpochNumber = _newEpochNumber;
  totalCoverTokens = _newTotalCoverTokens;
}
```

Recommendation:

Set hard-limit to number of iterations of the cycle.

MEDIUM | RESOLVED

By passing reward calculation amount equals to 0 contract lets to call that method before start of liquidity mining.

Snippet:

```
function getReward() external override returns (uint256) {
  require(
    startLiquidityMiningTime == 0 || block.timestamp > getEndLMTime(),
    "LM: 2 weeks after LME block"
  );

  address _teamAddr = usersTeamInfo[msg.sender].teamAddr;
  uint256 _userReward = checkAvailableBMIReward(msg.sender);

  if (_userReward == 0) {
    return 0;
  }

  bmiToken.transfer(msg.sender, _userReward);
  emit RewardSent(_teamAddr, msg.sender, _userReward);

  usersTeamInfo[msg.sender].countOfRewardedMonth += _getAvailableMonthForReward(msg.sender);

  return _userReward;
}
```

Recommendation:
Verify that startLiquidityMiningTime is greater than 0.

LOW | RESOLVED

State variable names policiesByInsuredAddress & _policies of contract PolicyBookRegistry are misleading.

Snippet:

```
mapping(address => address) public policiesByInsuredAddress;
EnumerableSet.AddressSet private _policies;
```

Recommendation:
Rename state variable policiesByInsuredAddress to policyBookssByInsuredAddress and _policies to _policyBooks respectively.

LOW | RESOLVED

Method getAvailableDAIXWithdrawalAmount of contract PolicyBook is returning DAI instead of DAIX.

Snippet:

```
function getAvailableDAIXWithdrawalAmount(address _userAddr)
  external
  view
  override
  returns (uint256)
{
  uint256 allCover = totalCoverTokens.add(_getUserAvailableDAI(_userAddr));

  return totalLiquidity > allCover ? totalLiquidity - allCover : 0;
}
```

Recommendation:
Rename method to getAvailableDAIWithdrawalAmount.

LOW | RESOLVED

Method stats of contract PolicyBook is returning static information instead of statistics.

Snippet:

```
function stats()
  external
  view
  override
  returns (
    string memory _symbol,
    address _insuredContract,
    IPolicyBookFabric.ContractType _contractType,
    bool _whitelisted
  )
{
  return (symbol(), insuranceContractAddress, contractType, whitelisted);
}
```

Recommendation:
Rename method to indicate that it returns static information instead of any statistic.

LOW | RESOLVED

Method uri from contract BMIDaiStaking is not returning token specific metadata url.

Snippet:

```
function uri(uint256) external view override returns (string memory) {
  return _uri;
}
```

Recommendation:
Return uri that includes token id.

We are grateful to have been given the opportunity to work with the Bridge Mutual team.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

Zokyo's Security Team recommends that the Bridge Mutual team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.