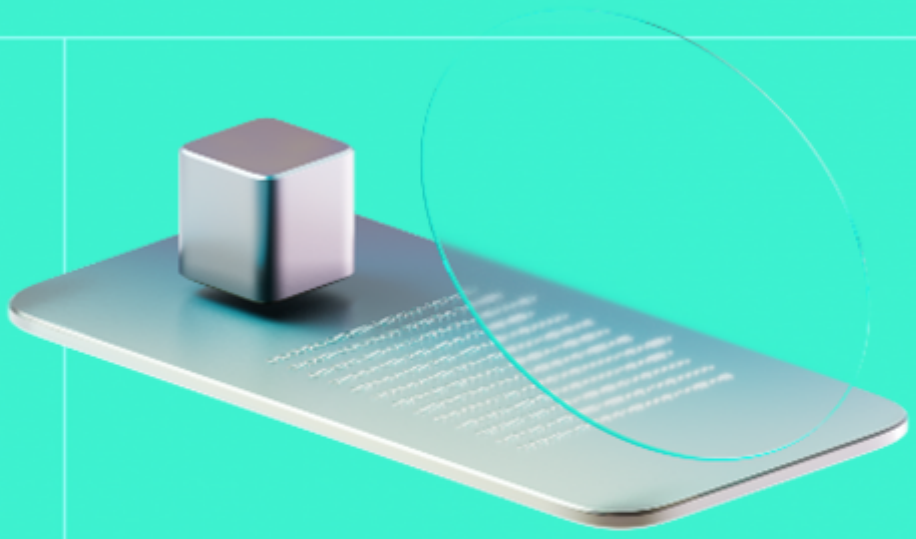
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: Paribus

Date: 08/01/2024

We thank Paribus for allowing us to conduct a Smart Contract Security Assessment. This document outlines our methodology, limitations, and results of the security assessment.

The Paribus Protocol is an Ethereum smart contract for supplying or borrowing assets. Through the pToken contracts, accounts on the blockchain supply capital (Ether or ERC-20 tokens) to receive pTokens or borrow assets from the protocol (holding other assets as collateral). The Paribus pToken contracts track these balances and algorithmically set interest rates for borrowers.

Platform: EVM

Language: Solidity

Tags: Governance, Timelock

Timeline: 28.11.2023 - 05.01.2024

Methodology: https://hackenio.cc/sc_methodology

Last Review Scope

Repository	https://github.com/Paribus/paribus-protocol
Commit	a45f4e9acab6879e90af4655a093f708e5418028
Remediation	1be7f09814f68bf6ef34f995eb3aa04eaa023426

Audit Summary

10/10	8/10	83.33%	10/10
Security Score	Code quality score	Test coverage	Documentation quality score

Total 9/10

The system users should acknowledge all the risks summed up in the risks section of the report

6	5	1	0
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	0
High	0
Medium	3
Low	2

Vulnerability	Status
F-2023-0058 - Redundant Function	Accepted
F-2023-0050 - Unsafe Use of DelegateCall	Fixed
F-2023-0053 - Redundant Fallback function	Fixed
F-2023-0055 - Missing Zero Address Validation	Fixed
F-2023-0057 - Missing Validation Of msg.value In executeTransaction()	Fixed
F-2023-0060 - Funds Lock	Fixed

This report 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 can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for Paribus
Audited By	Turgay Arda Usman, Kornel Światłowski
Approved By	Grzegorz Trawinski
Website	https://paribus.io
Changelog	06/12/2023 - Preliminary Report; 05/01/2024 - Final Report

Table to Contents

System Overview

Paribus is a Cardano-based lending/borrowing platform that aims to support conventional and unconventional crypto assets to allow for its users to unlock liquidity and interact with the markets without having to liquidate assets. It has the following contracts:

- GovernorDelegate — Used to initialize the contract during delegator constructor.
- GovernorDelegator — delegator contract for the GovernorDelegate.
- GovernorModerator - handles moderator assignments.

Privileged roles

- The Timelock contract incorporates a custom implementation of admin and pendingAdmin roles. The admin is empowered to set a new delay, accept a new admin, queue, cancel, and execute new transactions.
- The GovernorModerator contract implements a custom moderator role, allowing the address assigned to that role to invoke a queue.
- The GovernorDelegator contract employs a custom implementation of an admin role, allowing the address assigned to that role to establish a new implementation contract address, initialize implementation contract, launch PBX Rewards Campaign, set voting delay and period, set new moderator address, set the proposal threshold, add given address to whitelist and assinge whitelistGuardian role and set pedning admin role. This contract also implements whitelistGuardian role that can cancel given proposal and set the whitelist expiration as a timestamp for an account.

Executive Summary

This report presents an in-depth analysis and scoring of the customer's smart contract project. Detailed scoring criteria can be referenced in the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are detailed.
- Technical description is provided.

Code quality

The total Code Quality score is **8** out of **10**.

- The code contains style guide and best practice violations
  - See informational issues
- The development environment is configured.

Test coverage

Code coverage of the project is **83.33%** (branch coverage) .

- Deployment and basic user interactions are covered with tests.
- Interactions by several users are tested thoroughly.

Security score

Upon auditing, the code was found to contain **0** critical, **0** high, **3** medium, and **2** low severity issues, leading to a security score of **10** out of **10**.

All identified issues are detailed in the “Findings” section of this report.

Summary

The comprehensive audit of the customer's smart contract yields an overall score of **9**. This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

- No risks were identified.

Findings

Vulnerability Details

F-2023-0050 - Unsafe Use of DelegateCall - Medium

Description:

The GovernorDelegateor.sol benefits from delegatecalls while initializing the GovernorDelegate.sol contract. To do this, the contract calls the internal function delegateTo(). This function executes a delegatecall and then checks if the call is successful or not via the following if block:

```
function delegateTo(address callee, bytes memory data) internal {
    (bool success, bytes memory returnData) = callee.delegatecall(data);
    assembly {
        if eq(success, 0) {
            revert(add(returnData, 0x20), returndatasize())
        }
    }
}
```

The success flag, in the check, returned by delegatecall does not indicate whether the callee reverted or encountered an exception. It only indicates whether the call was executed successfully within the context of the callee. This means that even if the callee encounters a revert, the success flag might still be true.

This can lead the execution to continue even if the delegatecall fails.

Assets:

- GovernorDelegator.sol [https://github.com/Paribus/paribus-protocol]

Status: Fixed

Classification

Severity: Medium

Impact: 2/5

Likelihood: 2/5

Recommendations

Recommendation: To handle reverts or exceptions in the callee, the return data should be checked. If the first 32 bytes of the return data are non-zero, it typically

indicates an error or revert. A change in the code like the following would fix the issue:

```
function delegateTo(address callee, bytes memory data) internal {
    (bool success, bytes memory returnData) = callee.delegatecall(data);
    assembly {
        if eq(success, 0) {
            revert(add(returnData, 0x20), returndatasize())
        }
    }
    require(returnData.length == 0 || (returnData.length >= 32 && uint256(abi.de
}
```

This version checks if the delegatecall was unsuccessful based on the success flag, and then checks the return data to see if it contains a revert reason. If the return data length is greater than or equal to 32 bytes and the first 32 bytes are non-zero, it indicates a revert.

Remediation (revised commit: 1be7f09814f68bf6ef34f995eb3aa04eaa023426): The function is updated as suggested in the recommendation.

F-2023-0057 - Missing Validation Of msg.value In executeTransaction() - Medium

Description:

The `executeTransaction()` function is designed to execute a queued transaction after a specified period, and it is marked as `payable`. However, there is a lack of validation to ensure that the `msg.value` used in `executeTransaction()` matches the required value for the queued transaction. In cases where `msg.value` differs, it may result in a temporary funds lock.

```
function executeTransaction(address target, uint value, string memory signatu
    require(msg.sender == admin, "Timelock::executeTransaction: Call must cor

    bytes32 txHash = keccak256(abi.encode(target, value, signature, data, eta
    require(queuedTransactions[txHash], "Timelock::executeTransaction: Transa
    require(getBlockTimestamp() >= eta, "Timelock::executeTransaction: Transa
    require(getBlockTimestamp() <= eta + GRACE_PERIOD, "Timelock::executeTran

    queuedTransactions[txHash] = false;

    bytes memory callData;

    if (bytes(signature).length == 0) {
        callData = data;
    } else {
        callData = abi.encodePacked(bytes4(keccak256(bytes(signature))), data
    }

    // solium-disable-next-line security/no-call-value
    (bool success, bytes memory returnData) = target.call{value: value}(callD
    require(success, "Timelock::executeTransaction: Transaction execution rev

    emit ExecuteTransaction(txHash, target, value, signature, data, eta);

    return returnData;
}
```

Assets:

- Timelock.sol [https://github.com/Paribus/paribus-protocol]

Status: Fixed

Classification

Severity: Medium

Impact: 3/5

Likelihood: 2/5

Recommendations

Recommendation: It is recommended to include validation to verify whether the `msg.value` utilized in `executeTransaction()` aligns with the required value for executing the queued instruction.

Remediation (revised commit: 1be7f09814f68bf6ef34f995eb3aa04eaa023426): The following check is added: `require(value == msg.value, "Timelock::executeTransaction: Transaction ETH value mismatch");`

F-2023-0060 - Funds Lock - Medium

Description: The GovernorDelegator.sol contract delegates function calls coming to it to the deployed GovernorDelegate.sol contract, in other words, to the implementation contract. This forwarding is done through a fallback function. As the implemented fallback function has the payable modifier, it accepts native coins. However, neither the implementation contract nor this contract can handle native coins by design. This means that it is possible to send native funds to the contract and since the contract cannot handle native tokens they will be stuck.

```
fallback() external payable {
    // delegate all other functions to current implementation
    (bool success, ) = implementation.delegatecall(msg.data);


    assembly {
        let free_mem_ptr := mload(0x40)
        returndatacopy(free_mem_ptr, 0, returndatasize())


        switch success
        case 0 { revert(free_mem_ptr, returndatasize()) }
        default { return (free_mem_ptr, returndatasize()) }
    }
}
```

This can lead to a funds lock situation.

Assets:

- GovernorDelegator.sol [https://github.com/Paribus/paribus-protocol]

Status: Fixed

Classification

Severity: Medium

Impact: 3/5

Likelihood: 1/5

Recommendations

Recommendation: Implement a refund mechanism for the native tokens into the fallback function.

Remediation (revised commit: 1be7f09814f68bf6ef34f995eb3aa04eaa023426): Withdraw mechanism for native tokens is added.

F-2023-0053 - Redundant Fallback function - Low

Description: The `Timelock` contracts accept native tokens through `fallback` function. The contract's purpose is not to store native tokens, and this introduces unnecessary complexity, raises deployment costs, and may result in temporary fund lock.

Assets:

- Timelock.sol [https://github.com/Paribus/paribus-protocol]

Status: Fixed

Classification

Severity: Low

Impact: 2/5

Likelihood: 1/5

Recommendations

Recommendation: It is recommended to remove the unnecessary `fallback` function.

Remediation (revised commit: 1be7f09814f68bf6ef34f995eb3aa04eaa023426): Empty `fallback` function is removed.

F-2023-0055 - Missing Zero Address Validation - Low

Description: Address input parameters are being used without checking against the possibility of 0×0 value.

The following functions are missing such input validation: _setModerator(), _setWhitelistGuardian(), _setPendingAdmin(), constructor(),

This can lead to unwanted external calls to 0×0.

Assets:

- Timelock.sol [https://github.com/Paribus/paribus-protocol]
- GovernorModerator.sol [https://github.com/Paribus/paribus-protocol]
- GovernorDelegate.sol [https://github.com/Paribus/paribus-protocol]

Status: Fixed

Classification

Severity: Low

Impact: 1/5

Likelihood: 1/5

Recommendations

Recommendation: Implement zero address checks.

Remediation (revised commit: 1be7f09814f68bf6ef34f995eb3aa04eaa023426): Check against zero address is add in mentioned functions.

F-2023-0058 - Redundant Function - Info

Description: The `getBlockTimestamp()` and the `getBlockNumber()` functions are redundant, as they only return the global variables `block.timestamp` and `block.number`.

```
function getBlockTimestamp() internal view returns (uint) {
  // solium-disable-next-line security/no-block-members
  return block.timestamp;
}
```

```
function getBlockTimestamp() public virtual view returns (uint) {
    return block.timestamp;
}
```

This will lead to unnecessary gas and storage consumption.

Assets:

- Timelock.sol [https://github.com/Paribus/paribus-protocol]
- GovernorDelegate.sol [https://github.com/Paribus/paribus-protocol]

Status: Accepted

Classification

Severity: Info

Recommendations

Recommendation: Remove the redundant functions

Remediation-Mitigated: The client stated that they are aware of these functions, and they are keeping them for testing purposes.

Observation Details

F-2023-0016 - Floating Pragma - Info

Description: The project uses floating pragmas ^0.8.10.

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version, which may include bugs that affect the system negatively.

Assets:

- GovernorDelegate.sol [https://github.com/Paribus/paribus-protocol]
- GovernorDelegator.sol [https://github.com/Paribus/paribus-protocol]
- GovernorInterfaces.sol [https://github.com/Paribus/paribus-protocol]
- GovernorModerator.sol [https://github.com/Paribus/paribus-protocol]
- Timelock.sol [https://github.com/Paribus/paribus-protocol]

Status: Fixed

Recommendations

Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment. Consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Remediation (revised commit: 1be7f09814f68bf6ef34f995eb3aa04eaa023426): Floating pragma is set to 0.8.10.

F-2023-0047 - Use Custom Errors Instead Of Error Strings To Save Gas - Info

Description:

Custom errors were introduced in Solidity version 0.8.4, and they offer several advantages over traditional error handling mechanisms:

- Gas Efficiency: Custom errors can save approximately 50 Gas each time they are hit because they avoid the need to allocate and store revert strings. This efficiency can result in cost savings, especially when working with complex contracts and transactions.
- Deployment Gas Savings: By not defining revert strings, deploying contracts becomes more Gas-efficient. This can be particularly beneficial when deploying contracts to reduce deployment costs.
- Versatility: Custom errors can be used both inside and outside of contracts, including interfaces and libraries. This flexibility allows for consistent error handling across different parts of the codebase, promoting code clarity and maintainability.

Assets:

- Timelock.sol [https://github.com/Paribus/paribus-protocol]
- GovernorModerator.sol [https://github.com/Paribus/paribus-protocol]
- GovernorDelegator.sol [https://github.com/Paribus/paribus-protocol]
- GovernorDelegate.sol [https://github.com/Paribus/paribus-protocol]

Status: Accepted

Recommendations

Recommendation: It is recommended to use custom errors to save some Gas,

Remediation: The client stated that they will leave the issue as it is.

F-2023-0057 - Interface Tagged as a Contract - Info

Description: The GovernorDelegatorInterface interface in the GovernorInterfaces.sol contract is tagged as a contract.

```
contract GovernorDelegatorInterface is GovernorDelegationStorage {
    /// @notice Emitted when implementation is changed
    event NewImplementation(address oldImplementation, address newImplementation
}
```

Assets:

- GovernorInterfaces.sol [https://github.com/Paribus/paribus-protocol]

Status: Accepted

Recommendations

Recommendation: Change the “contract” statement with an “ interface statement.

Remediation: The client stated that they will leave the issue as it is.

F-2023-0062 - Public Functions That Should Be External - Info

Description: Functions that are meant to be exclusively invoked from external sources should be designated as "external" rather than "public." This is essential to enhance the overall security of the contract.

Assets:

- GovernorDelegate.sol [https://github.com/Paribus/paribus-protocol]

Status: Fixed

Recommendations

Recommendation: Transition the `initialize()` and `propose()` functions, which are exclusively utilized by external entities, from their current "public" visibility setting to the "external" visibility setting.

Remediation (revised commit: 1be7f09814f68bf6ef34f995eb3aa04eaa023426): The functions are updated as suggested in the recommendation.

F-2023-0063 - Code Duplication Of Owner Restricted Access Check - Info

Description: The recurrent utilization of the following check in multiple functions is evident: `require(msg.sender == admin);`. Enhancing Gas efficiency can be achieved by crafting a modifier and employing it consistently.

Assets:

- GovernorDelegate.sol [https://github.com/Paribus/paribus-protocol]

Status: Accepted

Recommendations

Recommendation: It is recommended to create an `onlyAdmin` modifier and apply it consistently for improved code structure.

Remediation: The client stated that they will leave the issue as it is.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with 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 report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

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 the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

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

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details

Repository	https://github.com/Paribus/paribus-protocol
Commit	a45f4e9acab6879e90af4655a093f708e5418028
Whitepaper	https://paribus.io/documents/PARIBUS-Litepaper-V1.0.pdf
Requirements	https://paribus.io/documents/PARIBUS-Litepaper-V1.0.pdf
Technical Requirements	Natspec

Scope Details

Repository	https://github.com/Paribus/paribus-protocol
Commit	1be7f09814f68bf6ef34f995eb3aa04eaa023426
Whitepaper	https://paribus.io/documents/PARIBUS-Litepaper-V1.0.pdf
Requirements	https://docs.paribus.io/docs/for-developers/governance
Technical Requirements	https://docs.paribus.io/docs/for-developers/governance

Contracts in Scope

contracts/Governance/GovernorDelegate.sol

contracts/Governance/GovernorDelegator.sol

contracts/Governance/GovernorInterfaces.sol

contracts/Governance/GovernorModerator.sol

contracts/Governance/Timelock.sol