0x Guard

Smart contracts security assessment

Final report

Tariff: Standard

Arbius Protocol

January 2024

0xguard.com

hello@0xguard.com

Contents

Introduction

The report has been prepared for **Arbius Protocol**.

The project consists of:

- upgradable project's BaseToken: an ERC-20 token with Permit and Votes extensions, it can be minted by a single privileged `l2Gateway` account;

- L1Token: an ERC-20 standard token contract without minting, upgradability, and modifications in transfer functions;

- TimelockV1: a direct import of OpenZeppelin's TimelockController, can't be upgraded;- GovernorV1: a governance contract inheriting OpenZeppelin's Governor, GovernorSettings, GovernorVotes, GovernorCompatibilityBravo, GovernorVotesQuorumFraction, and GovernorTimelockControl, can't be upgraded;

- upgradable EngineV1: the main contract to store models, tasks, and solutions. It also administers rewards, fees, and solution's contestations.

The code is available at the GitHub repository and was audited after the commit eba18c2b437a0408c67bba1a003de4c6cc3055e8.

The audit scope excludes the following contracts:

all contracts in the `Example` folder.

Report Update.

The contract's code was updated according to this report and rechecked after the commit e17188236e37488154484c7167616d525c915c07.

Name	Arbius Protocol
Audit date	2024-01-01 - 2024-01-29
Language	Solidity
Platform	Arbitrum Nova

Contracts checked

Name	Address
GovernorV1	
EngineV1	
BaseTokenV1	
L1Token	
TimelockV1	
Libraries: IPFS.sol, PoolAddress.sol	

Procedure

We perform our audit according to the following procedure:

Automated analysis

- Scanning the project's smart contracts with several publicly available automated Solidity analysis tools
- Manual verification (reject or confirm) all the issues found by the tools

Manual audit

- Manually analyze smart contracts for security vulnerabilities
- Smart contracts' logic check

Known vulnerabilities checked

Title	Check result
Unencrypted Private Data On-Chain	passed
Code With No Effects	passed
Message call with hardcoded gas amount	passed
Typographical Error	passed
DoS With Block Gas Limit	passed
Presence of unused variables	passed
Incorrect Inheritance Order	passed
Requirement Violation	passed
Weak Sources of Randomness from Chain Attributes	passed
Shadowing State Variables	passed
Incorrect Constructor Name	passed
Block values as a proxy for time	passed
Authorization through tx.origin	passed
DoS with Failed Call	passed
Delegatecall to Untrusted Callee	passed
Use of Deprecated Solidity Functions	passed
Assert Violation	passed
State Variable Default Visibility	passed
Reentrancy	passed
Unprotected SELFDESTRUCT Instruction	passed
Unprotected Ether Withdrawal	passed
Unchecked Call Return Value	passed

Floating Pragma	passed
Outdated Compiler Version	passed
Integer Overflow and Underflow	passed
Function Default Visibility	passed

Classification of issue severity

High severity	High severity issues can cause a significant or full loss of funds, change of contract ownership, major interference with contract logic. Such issues require immediate attention.
Medium severity	Medium severity issues do not pose an immediate risk, but can be detrimental to the client's reputation if exploited. Medium severity issues may lead to a contract failure and can be fixed by modifying the contract state or redeployment. Such issues require attention.
Low severity	Low severity issues do not cause significant destruction to the contract's functionality. Such issues are recommended to be taken into consideration.

Issues

High severity issues

1. Slashing amount can be manipulated (EngineV1)
Status: Fixed

Stake slashing occurs when a submitted solution is contested and both parties for and against contested solution vote by `getSlashAmount` amount of their stakes. The initial account who has submitted contested solution is forced to participate in voting if his stake is enough. The problem is `getSlashAmount` function calculates slashing amount as percentage of total staked token. This means that during the voting each voter reduces his stake by slashing amount calculated at the moment of voting, but actual slashing and refunding during the `contestationVoteFinish` execution will use different slashing amount.

```
    /// @notice Get slash amount
    /// @return Slash amount
    function getSlashAmount() public view returns (uint256) {
        uint256 ts = getPsuedoTotalSupply();
        if (ts < MIN_SUPPLY_FOR_SLASHING) {
            return 0;
        }
        return ts - ((ts * (1e18 - slashAmountPercentage)) / 1e18);
    }
```

Recommendation: Fix the logic, increase test coverage.

2. Voting can be manipulated (EngineV1)

Status: Fixed

The contestation of a solution can be finished at any moment after the `minContestationVotePeriodTime` time passed since contestation (and voting) has started. During the voting period, any account can become a validator and participate in voting. Thus, as soon as contestation is ready for finalization, any malicious actor can increase his voting power and won contestation, slashing all opponents, including honest validators.

Recommendation: Fix the voting logic. For example, there can be introduced a preparation window for voting, after which all new validators can't vote for that particular contestation.

Medium severity issues

1. Outdated imports (GovernorV1)

Status: Fixed

The inherited OpenZeppelin contracts are outdated (v4.3 release), they don't include several bug fixes and improvements for Governor contracts, check v4.4.0, v4.4.2, v4.5.0, v4.6.0, v4.7.2, v4.8.0, v4.8.3, v4.9.0, v4.9.1, v5.0.0 releases.

Recommendation: Use newer release of OpenZeppelin's contracts.

2. Possible math underflow (EngineV1)

Status: Fixed

The function `getPsuedoTotalSupply` subtract contract's BaseToken current balance from STARTING_ENGINE_TOKEN_AMOUNT constant, which is 60% of total supply. There's no guarantee, that this operation succeeds. Thus, all functions containing `getPsuedoTotalSupply` call (`getReward`, `getSlashAmount`, `getValidatorMinimum`, `submitContestation`, `voteOnContestation`, `contestationVoteFinish`, `signalSupport`, `submitSolution`, `claimSolution`) can fail too.

```
    /// @notice Because we are using a token which is fully minted upfront we must
calculate total supply based on the amount remaining in Engine
    /// @return Total supply of Engine tokens
    function getPsuedoTotalSupply() public view returns (uint256) {
        return STARTING_ENGINE_TOKEN_AMOUNT - baseToken.balanceOf(address(this));
    }
```

Recommendation: Consider case of `baseToken.balanceOf(address(this)) > STARTING_ENGINE_TOKEN_AMOUNT`.

Low severity issues

1. Not following guidelines (EngineV1)

Status: Fixed

The EngineV1 contract is designed to be upgradable with use of OpenZeppellin's library and Upgrades Plugin. However, EngineV1 contract doesn't follow the OpenZeppellin guideline for writing upgradable contracts: it neither have storage gaps, nor it initializes the implementation (`_disableInitializers` in the constructor).

Recommendation: Follow the guideline to ease future upgrades.

2. Inconsistent comment (EngineV1)

Status: Fixed

L23 comment contains wrong link to external documentation.

```
// https://github.com/OffchainLabs/arbitrum/blob/master/docs/sol_contract_docs/md_docs/
arb-os/arbos/builtin/ArbSys.md
```

Recommendation: Correct link is https://github.com/OffchainLabs/arbitrum-classic/blob/master/docs/sol_contract_docs/md_docs/arb-os/arbos/builtin/ArbSys.md.

Conclusion

Arbius Protocol GovernorV1, EngineV1, BaseTokenV1, L1Token, TimelockV1, Libraries: IPFS.sol, PoolAddress.sol contracts were audited. 2 high, 2 medium, 2 low severity issues were found.
2 high, 2 medium, 2 low severity issues have been fixed in the update.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability)set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without 0xGuard prior written consent.

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts 0xGuard to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

0x Guard