CERTIK

Security Assessment

# Juicebox Contracts V2

Mar 29th, 2022

# Table of Contents

# Summary

This report has been prepared for Juicebox Contracts V2 to discover issues and vulnerabilities in the source code of the Juicebox Contracts V2 project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

# Overview

## Project Summary

| | |
|---|---|
| Project Name | Juicebox Contracts V2 |
| Platform | Ethereum |
| Language | Solidity |
| Codebase | https://github.com/jbx-protocol/juice-contracts-v2 |
| Commit | 2d846c510df9fd3e6eb844a08db0ea5cf6d3f095 |

## Audit Summary

| | |
|---|---|
| Delivery Date | Mar 29, 2022 UTC |
| Audit Methodology | Static Analysis, Manual Review |

## Vulnerability Summary

| Vulnerability Level | Total | Pending | Declined | Acknowledged | Mitigated | Partially Resolved | Resolved |
|---|---|---|---|---|---|---|---|
| ● Critical | 2 | 0 | 0 | 2 | 0 | 0 | 0 |
| ● Major | 12 | 0 | 0 | 12 | 0 | 0 | 0 |
| ● Medium | 3 | 0 | 0 | 2 | 0 | 0 | 1 |
| ● Minor | 4 | 0 | 0 | 4 | 0 | 0 | 0 |
| ● Informational | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| ● Discussion | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## Audit Scope

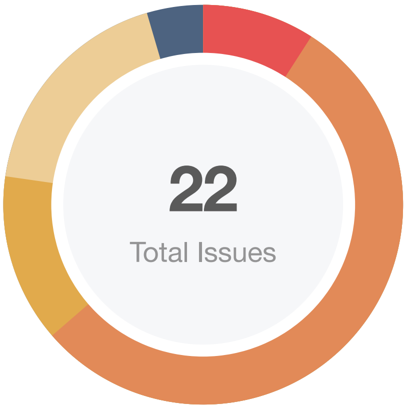
| ID | File | SHA256 Checksum |
|---|---|---|
| JBC | abstract/JBControllerUtility.sol | f75a67bf73e33511d1bf95387142400a3b1722b343e65d48fc64a10907a188c1 |
| JBO | abstract/JBOperatable.sol | 5eece505fa18abef81219f20dadea3f138b3c4c2be9b0dcf9e2c48a8fe0881d0 |
| JBP | abstract/JBProject.sol | df052296dfcc532d903a01e8ef11b180d02cc42ab2a43074043835f849cdd334 |
| JBB | enums/JBBallotState.sol | 2cca68ba8359303baffdd3d2c0f40ec7ff90574e2a3e2c5b05dca21bc995bca8 |
| IJB | interfaces/IJBController.sol | 71bb74bafbcfaa86a108dd4e0d6bdc60eda177ca5a58254f6f133e64ed5b45de |
| IJC | interfaces/IJBControllerUtility.sol | 12a6273523cefa2c517b8813e165187a34a5365be3acbc2464d8b303789b0782 |
| IJD | interfaces/IJBDirectory.sol | 286cfd3dcf313cc42da10a7e474762fabfcb93d88a4ea73feb9e33b346e6b5a5 |
| IJE | interfaces/IJBETHPaymentTerminal.sol | a722d6b3bdf67d71139c5a1c4bd950e8b47e1cff0a545042e98a8fd66557fab6 |
| IJF | interfaces/IJBFeeGauge.sol | 1ca6ef69a31ce45999c3aaa9b916ad334b9e9f773192d4dea069a0d89086f880 |
| IJK | interfaces/IJBFundingCycleBallot.sol | 6ff022c5d5d0f8c540905ca912036be119f50326b298a112f690732764fa4680 |
| IJS | interfaces/IJBFundingCycleDataSource.sol | f12d70cc76767cc64079a88709ea3eb1b1de73bf6be82fd6e2bd33c12cb96ade |
| IJP | interfaces/IJBFundingCycleStore.sol | 53c0a9e1b8eb99e481fcbe32fd5782fce6950b29a38e9f94075a09b20d348699 |
| IJO | interfaces/IJBOperatable.sol | 4d262fa25df12c656dabad4e9c205af48c69dbdad7841d414feba6253efce24f |
| IBO | interfaces/IJBOperatorStore.sol | f885ac9b3f349bb1822146ca81d56382ba05d1465ff5da9f7e05c5867638f729 |

| ID | File | SHA256 Checksum |
|---|---|---|
| IBP | interfaces/IJBPayDelegate.sol | 79b9fd8378a9977d153079e2ffdca7ddf7102588598c37fe70bc419ed80564c1 |
| IBC | interfaces/IJBPrices.sol | 9c89ff63e59168c89cef60982a6ef204d45a7bd77611639336eedfa43fd058fc |
| IBK | interfaces/IJBProjects.sol | fd02f49ff9355a99995f5b221b0149b813b83de8272b54bb60b699ad9e2d19a9 |
| IJR | interfaces/IJBRedemptionDelegate.sol | c24375efa81265aebfd6d77328c011200fd8be070655287df6467d747d6ed802 |
| IJA | interfaces/IJBSplitAllocator.sol | 93d5a9b00a57c53f7e2903fa1752ccddeda0eed49d39c00d5a2ab00f74738715 |
| IBS | interfaces/IJBSplitsStore.sol | ae65fe7ad157d66abab08c87c3572706482a335a1c186e5e3d32757cfe55abd8 |
| IJT | interfaces/IJBTerminal.sol | ab6d1f4d102136759a88af86d11a0b347ad8694fac0611e05cdd2320a19a919e |
| IJU | interfaces/IJBTerminalUtility.sol | a216a60bc4bd4b4c30613417aeb733290c4e8d3c554daadb843214367f6e3d0c |
| IBT | interfaces/IJBToken.sol | 9b4bcd58e98d6499d7db12f62ea4af3dd07fcceee735dcf2cfa41354d25c6a4a |
| ITS | interfaces/IJBTokenStore.sol | c009cd2db847147d4f5c98cf259cce468bce03a4b9477088b74f93962d136dc3 |
| IBU | interfaces/IJBTokenUriResolver.sol | ea6337a80244b2000d9948f627d01f7ca5cd105f89012fc0fa040f769fef99be |
| JBK | libraries/JBConstants.sol | 7f8ad371cc2e037125b131ed15e7c52ba60a1ac56678eb1ccb06432f3ea543a9 |
| JCC | libraries/JBCurrencies.sol | 507d3e929ceb8e354702023bd85f64ab00db4f8f4634bd1cb02610296efd4944 |
| JBF | libraries/JBFundingCycleMetadataResolver.sol | 5a85a2634e57396086dea8567b6e08e58f860f0fe743b7ac0051c1b364554907 |
| JOC | libraries/JBOperations.sol | ad7a60290e1c8deca0a25502ea61ea94ef6552244af005b031029b1259810930 |

| ID | File | SHA256 Checksum |
|---|---|---|
| JBS | libraries/JBSplitsGroups.sol | c5ade262956b060e82168ef3807d554a97d6f720bd971fd110839d4dd2f0a3d8 |
| JBT | libraries/JBTokens.sol | cee85338870941dbceb69cdf03a62bce7352e2786c0e163e99bf6de416714e0a |
| JBD | structs/JBDidPayData.sol | 9b998eb39ce9a70ee4c0f55f7b7fa52daed60ce17c20d74f33c99d552e95ab8c |
| JBR | structs/JBDidRedeemData.sol | 29f144559a8871cdbbd6b434848ab6be6c3e89391732782a09ed9cbdf4eaf4d0 |
| JFC | structs/JBFee.sol | 49f90cdccc27309443eafd8388175c3415585628fd3f7b226d0552fc9ca61fdd |
| JBA | structs/JBFundAccessConstraints.sol | e507635c9f4d7e8faea13f4043d802ef72233ab72df6c41d31da89b588cee04c |
| JFK | structs/JBFundingCycle.sol | 5ed5d997117b1ef69978632576df5b2f0b0d0b39ceeee0200334e165b980c87a |
| JFD | structs/JBFundingCycleData.sol | cf71da35da7ff461a5252b49d2cfe0fcc478dd8eb4d3a50211c4e7840ee5c708 |
| JBM | structs/JBFundingCycleMetadata.sol | 9417e31dd783ac2ca11525ae582e0cb207f2ece80f771d6ac46250a955ae033b |
| JBG | structs/JBGroupedSplits.sol | 113f51f2c1deb689d09f3c1e54ff6901eaada99d61f32b68bbe9cf3b1773f864 |
| JOD | structs/JBOperatorData.sol | 265e05d7152162b1beb484a9782c5d07be61edf353929ade34d7cc1806ff18c1 |
| JPP | structs/JBPayParamsData.sol | 0b99667ae3331e6c9d7595e93f2d26fcec7f9ac141b81dd05000a52351cd2e52 |
| JPM | structs/JBProjectMetadata.sol | 4422d5a94d7cf61c5bccd69f0c0c860d9496793198eea6ac3b10abfbd80b6567 |
| JRP | structs/JBRedeemParamsData.sol | 00c35d9d26a5a32cc346b11c91ec41d785190093add78395d87752ebe210dbf1 |
| JSC | structs/JBSplit.sol | 0d0d66beb1f4ffb02afbb00c44934852cbf3c59ba169b01effa1bafccfebbde3 |

| ID | File | SHA256 Checksum |
|---|---|---|
| JFF | test/JBFakeFundingCycleMetadataResolver.sol | 83b00dee9fefb26018c3b1ac228b9b3d1161806079d4e1571dd092b587277d93 |
| JFP | test/JBFakeProject.sol | 3e246749e1b68f92ad13f1968c4336ca9661c9231607ff69f20c209ae3103599 |
| JDR | JB3DayReconfigurationBufferBallot.sol | 243069ed3efe0d18c5a97d39de6a283e5892ebe4a1bc377ae0a8b4a02c1c219b |
| JDB | JB7DayReconfigurationBufferBallot.sol | 5fed6c70711f61144686ae3d6f3f383ed66739373fecf126e8b049281ab79d96 |
| JCK | JBController.sol | 956cf815f20ab9941813c1cd75533886207e4f6f1aa37c2b6e2c772d11077e3b |
| JDC | JBDirectory.sol | 3fda232dcad8644527597a47159fccb71f1a7bf892d369766db26c98a39586e2 |
| JBE | JBETHPaymentTerminal.sol | f1ae346d5c827363e327d4e5d2cc6fbde8012a9062e43f0c8fc452768649968c |
| JBH | JBETHPaymentTerminalStore.sol | 528c3d799a0fa0a6bcda11c74ec470968765e2387cea4cb415b70f85b37074f9 |
| JFS | JBFundingCycleStore.sol | c2293335a08757fade1518edd4dee01b597dc54251dc87bf30df43d2077cbdb5 |
| JOS | JBOperatorStore.sol | 0cc20d0b9fa9174facd06bbfc369b0f92797821709718044b17034c6eafd87db |
| JPC | JBPrices.sol | d391951753aabac4aeccb6c6e15a5573ac592e4869370b9bb422d02e2f44f420 |
| JPK | JBProjects.sol | c2a94bb2369141c2f2597ccd7961f949443e761b1749ed84d389def2d14f5eb6 |
| JSS | JBSplitsStore.sol | 34d61a64d97c92a73486d1ca353a96ba70d9b5e062cf2806f6def9e09ddf7821 |
| JTC | JBToken.sol | 906efd8c5d07ab76705c403cf969e16ba51d44af0202c3792d936b3e21034254 |
| JTS | JBTokenStore.sol | 3b2783a320f4b852012de6624d2f66477494808b8e1fd510522870a3b7bde28a |

# Findings



| | | |
|---|---|---|
| 🟥 | **Critical** | **2** (9.09%) |
| 🟧 | **Major** | **12** (54.55%) |
| 🟨 | **Medium** | **3** (13.64%) |
| 🟨 | **Minor** | **4** (18.18%) |
| 🟦 | **Informational** | **1** (4.55%) |
| 🟩 | **Discussion** | **0** (0.00%) |

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| GLOBAL-01 | Unknown implementation of interfaces | Volatile Code | ● Minor | ⓘ Acknowledged |
| **JBE-01** | User funds could be arbitrarily transfer out | **Centralization / Privilege** | ● **Critical** | ⓘ Acknowledged |
| **JBE-02** | Centralization risk in JBETHPaymentTerminal.sol | **Centralization / Privilege** | ● **Major** | ⓘ Acknowledged |
| JBE-03 | Potential redeem issue for investors | Logical Issue | ● Major | ⓘ Acknowledged |
| **JBH-01** | Centralization risk in JBETHPaymentTerminalStore.sol | **Centralization / Privilege** | ● **Major** | ⓘ Acknowledged |
| **JCK-01** | Centralization risk in JBController.sol | **Centralization / Privilege** | ● **Major** | ⓘ Acknowledged |
| JCK-02 | Risk on the passed-in variable `_reservedRate` | Logical Issue | ● Medium | ✓ Resolved |
| JCK-03 | Logic issue on migration | Logical Issue | ● Medium | ⓘ Acknowledged |
| JCK-04 | Logic issue about `_processedTokenTrackerOf[_projectId]` | Logical Issue | ● Medium | ⓘ Acknowledged |
| JCK-05 | Logic issue in `_reservedTokenAmountFrom()` | Logical Issue | ● Minor | ⓘ Acknowledged |
| JCK-06 | Lack of restriction on function `launchProjectFor()` | Volatile Code | ● Informational | ⓘ Acknowledged |

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| **JDC-01** | Centralization risk in JBDirectory.sol | **Centralization / Privilege** | ● **Major** | ⓘ Acknowledged |
| **JFS-01** | Centralization risk in JBFundingCycleStore.sol | **Centralization / Privilege** | ● **Major** | ⓘ Acknowledged |
| **JPC-01** | Centralization risk in JBPrices.sol | **Centralization / Privilege** | ● **Major** | ⓘ Acknowledged |
| JPC-02 | Third party dependencies of `AggregatorV3Interface` | Logical Issue | ● Minor | ⓘ Acknowledged |
| **JPK-01** | Centralization risk in JBProjects.sol | **Centralization / Privilege** | ● **Major** | ⓘ Acknowledged |
| **JSS-01** | Centralization risk in JBSplitsStore.sol | **Centralization / Privilege** | ● **Major** | ⓘ Acknowledged |
| **JTC-01** | Centralization risk in JBToken.sol | **Centralization / Privilege** | ● **Major** | ⓘ Acknowledged |
| **JTS-01** | Centralization risk in JBTokenStore.sol | **Centralization / Privilege** | ● **Major** | ⓘ Acknowledged |
| **JUI-01** | Project contract implementations and parameter settings can be arbitrarily set and modified | **Centralization / Privilege** | ● **Critical** | ⓘ Acknowledged |
| JUI-02 | Investor assets are diluted by the reserved token | Logical Issue | ● Major | ⓘ Acknowledged |
| JUI-03 | Calculation issues by wrong divisors | Logical Issue | ● Minor | ⓘ Acknowledged |

# GLOBAL-01 | Unknown Implementation Of Interfaces

| Category | Severity | Location | Status |
|---|---|---|---|
| Volatile Code | ● Minor | Global | ⓘ Acknowledged |

## Description

There is no contract implementation present for the interfaces `IJBFeeGauge`, `IJBSplitAllocator`, `IJBFundingCycleDataSource`, `IJBPayDelegate` and `IJBRedemptionDelegate` in the codebase. The scope of the audit treats 3rd party entities as black boxes and assumes their functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of 3rd parties can possibly create severe impacts, such as increasing fees of 3rd parties, migrating to new LP pools, etc.

## Recommendation

We recommend ensuring the external addresses are correct, the external contracts are credible, and the third-party implementations and the way these functions are called can meet the requirements. We also encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

## Alleviation

The team acknowledged this issue and they stated the following:

"As you've mentioned, anyone can roll out a terminal for people to use. These new terminals have arbitrary functional differences from the ones written by the community and should require separate audits. It is the responsibility of projects to determine the efficacy and legitimacy of terminals they accept funds through.

In the scope of this audit are the JBETHPaymentTerminal and JBERC20PaymentTerminal."

# JBE-01 | User Funds Could Be Arbitrarily Transfer Out

| Category | Severity | Location | Status |
|---|---|---|---|
| **Centralization / Privilege** | ● **Critical** | JBETHPaymentTerminal.sol | ⓘ Acknowledged |

## Description

The function `distributePayoutsOf()` distribute the ETHs paid by normal users among the splits and transfer the `_leftoverDistributionAmount` ETHs directly to the project owner's address. These addresses are all EOAs(Externally Owned Account) set by project owner or `RECONFIGURE` operators.

Additionally, by calling the function `useAllowanceOf()`, the project owner and `USE_ALLOWANCE` operators can send the rest ETHs (overflow) to an arbitrary address `_beneficiary`, which is an EOA as well.

As a result, any compromise to the EOAs may allow the malicious owner to steal the ETHs.

## Recommendation

We strongly recommend that the EOA addresses in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term and long-term:

## Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

## Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

## Alleviation

The team acknowledged this issue and they stated the following:

"Being a treasury management tool, transfer of fund in and out of Juicebox is intended.

The `distributePayoutsOf()` function sends a project's treasury funds to configured splits, and sends any remaining funds to the project owner's address if the splits do not add up to 100% by design. Payouts can only be distributed from the treasury within the project's distribution limit.

The `useAllowanceOf()` function allows a project owner to withdraw discretionary funds from its project's overflow within the allowance that it pre-configures in the funding cycle. This is by design."
(reference: https://docs.juicebox.money/protocol/learn/glossary/overflow)

# JBE-02 | Centralization Risk In JBETHPaymentTerminal.sol

| Category | Severity | Location | Status |
|---|---|---|---|
| **Centralization / Privilege** | ● **Major** | JBETHPaymentTerminal.sol | ⓘ Acknowledged |

## Description

In the contract `JBETHPaymentTerminal`, the role `owner` has the authority over the following function:

- function `setFee()`: change the fee percentage and contractually capped at 5%,
- function `setFeeGauge()`: change the `feeGauge` address to affect the fee discount amount.

Also, the role project owner has the authority over the following function:

- function `useAllowanceOf()`: send ETH to arbitrary `_beneficiary` address with the `overflowAllowanceOf` as the limit,
- function `redeemTokensOf()`: claim the project's overflowed ETH,
- function `migrate()`: migrate project funds and operations to a new terminal,
- function `processFees()`: process the held fees.

Among the previous mentioned functions which can be called by the project owner, the specific operator roles have the authority over the following function:

- The operator with the `USE_ALLOWANCE` permission can call the function `useAllowanceOf()`.
- The operator with the `REDEEM` permission can call the function `redeemTokensOf()`.
- The operator with the `MIGRATE_TERMINAL` permission can call the function `migrate()`.
- The operator with the `PROCESS_FEES` permission can call the function `processFees()`.

The contract deployer has the authority over the following function:

- function `constructor()`: transfer the `owner` role to an arbitrary address, initialize important contract addresses to any contract addresses implementing the corresponding interfaces, for example: `operatorStore`, `projects`, `directory`, `splitsStore`.

Any compromise to the privileged accounts may allow the hacker to take advantage of this authority and users' assets may suffer loss.

## Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

## Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

## Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

## Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

## Alleviation

The team acknowledged this issue and they stated the following:

"For each project, the above-mentioned functions can only be accessed by either the address that owns the project's NFT or by operator addresses explicitly allowed by the address that owns the project's NFT." (reference: https://docs.juicebox.money/protocol/learn/glossary/operator#operatable-functionality)

# JBE-03 | Potential Redeem Issue For Investors

| Category | Severity | Location | Status |
|---|---|---|---|
| Logical Issue | ● Major | JBETHPaymentTerminal.sol | ⓘ Acknowledged |

## Description

Only when there exist `overflow` ETHs could investors redeem their funds, nevertheless, under the number of `distributionLimitOf()` and `overflowAllowanceOf()`, the project owner and corresponding operators could always revoke `distributePayoutsOf()` and `useAllowanceOf()` to take funds away. Besides, the ETHs redeemed by the investors would further shrink in terms of the `_redemptionRate` and the `reservedRate`, as a result, only a few ETHs or even none will be left when investors want to redeem their funds.

## Recommendation

We would like to confirm with the client if the current implementation aligns with the original project design.

## Alleviation

The team acknowledged this issue and they stated the following:

"Overflow is a function of a project's distribution limit, this is by design. If a project owner reconfigures its distribution limit, it can reshape was is reclaimable by token holders who redeem. This is by design." (reference: https://docs.juicebox.money/protocol/learn/glossary/overflow)

# JBH-01 | Centralization Risk In JBETHPaymentTerminalStore.sol

| Category | Severity | Location | Status |
|---|---|---|---|
| **Centralization / Privilege** | ● **Major** | JBETHPaymentTerminalStore.sol | ⓘ Acknowledged |

## Description

In the contract `JBETHPaymentTerminalStore`, the role `terminal` has the authority over the following function:

- function `recordPaymentFrom()`: record user payment data and mint project token to the user,
- function `recordDistributionFor()`: calculate and record the distribution amount,
- function `recordUsedAllowanceOf()`: calculate and record the withdrawnAmount amount,
- function `recordRedemptionFor()`: burn user's project token, calculate and record the redeem amount and transfer the corresponding amount of ETH to the user,
- function `recordAddedBalanceFor()`: add the ETH balance of a given project,
- function `recordMigration()`: set the current project ETH balance to 0 and return the original balance.

The contract deployer has the authority over the following function:

- function `constructor()`: initialize important contract addresses to any contract addresses implementing the corresponding interfaces, for example: `prices`, `projects`, `directory`, `fundingCycleStore` and `tokenStore`.

Any compromise to the privileged accounts may allow the hacker to take advantage of this authority and users' assets may suffer loss.

## Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

## Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

## Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

## Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

## Alleviation

The team acknowledged this issue and they stated the following:

"A store’s terminal is the only address that has access to recording data. This is by design. It would be a major flaw if this were not the case."

# JCK-01 | Centralization Risk In JBController.sol

| Category | Severity | Location | Status |
|---|---|---|---|
| **Centralization / Privilege** | ● **Major** | JBController.sol | ⓘ Acknowledged |

## Description

In the contract `JBController`, the role project owner has the authority over the following function:

- function `launchFundingCycleFor()`: initialize the funding cycle configurations for a given project,
- function `reconfigureFundingCyclesOf()`: change the funding cycle configurations for a given project,
- function `issueTokenFor()`: create a new ERC20 token and associated with a given project ,
- function `changeTokenOf()`: change the associated token of a give project,
- function `mintTokensOf()`: mint new tokens for a give project,
- function `burnTokensOf()`: burn tokens for a give project,
- function `migrate()`: move the project to another controller.

Also, the operator with the `RECONFIGURE` permission has the authority over the following function:

- function `launchFundingCycleFor()` : initialize the funding cycle configurations for a given project,
- function `reconfigureFundingCyclesOf()`: change the funding cycle configurations for a given project.

The operator with the `ISSUE` permission has the authority over the following function:

- function `issueTokenFor()`: create a new ERC20 token associated with a given project.

The operator with the `CHANGE_TOKEN` permission has the authority over the following function:

- function `changeTokenOf()`: change the associated token of a given project.

The operator with the `MINT` permission has the authority over the following function:

- function `mintTokensOf()`: mint new tokens for a give project.

The operator with the `BURN` permission has the authority over the following function:

- function `burnTokensOf()`: burn tokens for a give project.

The operator with the `MIGRATE_CONTROLLER` permission has the authority over the following function:

- function `migrate()`: move the project to another controller.

The contract deployer has the authority over the following function:

- function `constructor()`: initialize important contract addresses to any contract addresses implementing the corresponding interfaces, for example: `operatorStore`, `projects`, `directory`, `fundingCycleStore`, `tokenStore` and `splitsStore`.

Any compromise to the privileged accounts may allow the hacker to take advantage of this authority and users' assets may suffer loss.

## Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

## Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

## Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND

- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

## Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

## Alleviation

The team acknowledged this issue and they stated the following:

"For each project, the above-mentioned functions can only be accessed by either the address that owns the project's NFT or by operator addresses explicitly allowed by the address that owns the project's NFT." (reference: https://docs.juicebox.money/protocol/learn/glossary/operator#operatable-functionality)

# JCK-02 | Risk On The Passed-in Variable `_reservedRate`

| Category | Severity | Location | Status |
|---|---|---|---|
| Logical Issue | ● Medium | JBController.sol: 543 | ✓ Resolved |

## Description

In the function `mintTokensOf()`, the `_reservedRate` is a passed-in variable set by the caller. We understand the contract `JBETHPaymentTerminalStore` will call this function and pass the correct value `fundingCycle.reservedRate()`, however, the project owner and other `MINT` operators can also call this function externally with an arbitrary `_reservedRate` value.

## Recommendation

We would like to confirm with the client if the current implementation aligns with the original project design. It may be better to get the `reservedRate` of the current funding cycle by `fundingCycleStore.currentOf(_projectId).reservedRate()` rather than passing in an uncertain value.

## Alleviation

The team heeded our advice and resolved this issue in commit `f670d12b5947d3d3e2fe6d1b4e2b3ac1845b655a`.

# JCK-03 | Logic Issue On Migration

| Category | Severity | Location | Status |
|---|---|---|---|
| Logical Issue | ● Medium | JBController.sol: 681, 724 | ⓘ Acknowledged |

## Description

In the function `migrate()`, the old controller will call the function `prepForMigrationOf()` in the target controller to transfer the token total supply. The `_projectId` used in these two functions are the same, indicating that the two controllers will use the same `_projectId`. However, there may be already a created project in the position of `_projectId`. Thus, without proper management, the migration may override the currently active project in the target controller.

## Recommendation

We recommend carefully managing the project and perhaps give the migrating project a new project id in the target controller.

## Alleviation

The team acknowledged this issue and they will leave it as it is.

# JCK-04 | Logic Issue About `_processedTokenTrackerOf[_projectId]`

| Category | Severity | Location | Status |
|---|---|---|---|
| Logical Issue | ● Medium | JBController.sol: 537~602 | ⓘ Acknowledged |

## Description

The state variable **_processedTokenTrackerOf[_projectId]** is used to track the reserved tokens for a given project. When new tokens are minted, part of the tokens will be recorded with this variable instead of directly minting.

However, in the function `mintTokensOf()`, the variable **_processedTokenTrackerOf[_projectId]** is only updated when the passed-in `_reservedRate` equals to `MAX_RESERVED_RATE` or 0. When the value of `_reservedRate` is between `MAX_RESERVED_RATE` and 0, the function mints part of the tokens but does not record the other part in the variable **_processedTokenTrackerOf[_projectId]**.

Since the `beneficiaryTokenCount` is the actual minted token amount, the other portion is the reserved token amount which is `_tokenCount - beneficiaryTokenCount`. The new **_processedTokenTrackerOf[_projectId]** should be:

_processedTokenTrackerOf[_projectId] = _processedTokenTrackerOf[_projectId] +

beneficiaryTokenCount - (_tokenCount - beneficiaryTokenCount) = _processedTokenTrackerOf[_projectId] + 2 * beneficiaryTokenCount - _tokenCount

(The formula only shows the algebra calculation logic and does not consider the variable type.)

## Recommendation

The reserve amount calculation logic described in the team's response is only reasonable when the reserve rate is a constant value that does not change. However, the reserve rate of a given project can be changed when setting up a new funding cycle. Thus, if the project owner does not call the function distributeReservedTokensOf() to distribute the reserved token, the reserve rate can be updated and the calculation in the function _reservedTokenAmountFrom() will use the new reserve rate. Because the minted amount(beneficiaryTokenCount) is already calculated by the old reserve rate, inconsistency occurs. This is why we recommend recording the _processedTokenTrackerOf[_projectId] (reserve amount) for each mint/burn operations in the functions mintTokensOf() and burnTokensOf().

## Alleviation

The team acknowledged this issue and they stated the following:

"This issue is well known and by design — a tradeoff of making the mint/pay transaction as cheap as possible."

# JCK-05 | Logic Issue In `_reservedTokenAmountFrom()`

| Category | Severity | Location | Status |
|---|---|---|---|
| Logical Issue | ● Minor | JBController.sol: 868~890 | ⓘ Acknowledged |

## Description

The return value of the function `_reservedTokenAmountFrom()` may be incorrect.

According to the code, the `_processedTokenTracker` is

---

the minted token amount(total supply) - _unprocessedTokenBalanceOf

---

Thus, L874 will calculate the `_unprocessedTokenBalanceOf` correctly by the result of "total supply - _processedTokenTracker". As we mentioned in the issue `JCK-03`, the reserved token amount is the un-minted token recorded with `_processedTokenTracker`. So the `_unprocessedTokenBalanceOf` calculated in L874 is exactly the reserved token amount and the `return` statement (L884-L889) in the function `_reservedTokenAmountFrom()` can just return the value of `_unprocessedTokenBalanceOf`.

## Recommendation

The reserve amount calculation logic described in the team's response is only reasonable when the reserve rate is a constant value that does not change. However, the reserve rate of a given project can be changed when setting up a new funding cycle. Thus, if the project owner does not call the function distributeReservedTokensOf() to distribute the reserved token, the reserve rate can be updated and the calculation in the function _reservedTokenAmountFrom() will use the new reserve rate. Because the minted amount(beneficiaryTokenCount) is already calculated by the old reserve rate, inconsistency occurs. This is why we recommend recording the _processedTokenTrackerOf[_projectId] (reserve amount) for each mint/burn operations in the functions mintTokensOf() and burnTokensOf().

## Alleviation

The team acknowledged this issue and they stated the following:

"This issue is well known and by design — a tradeoff of making the mint/pay transaction as cheap as possible."

# JCK-06 | Lack Of Restriction On Function `launchProjectFor()`

| Category | Severity | Location | Status |
|---|---|---|---|
| Volatile Code | ● Informational | JBController.sol: 341 | ⓘ Acknowledged |

## Description

The function `launchProjectFor()` in the contract `JBController` does not have a permission restriction, so anyone can call this function to create a project. This may allow the malicious users to take advantage of this. For example:

- Front-running: since project 1 is the platform project which receives the charged fees, the hackers can create a project right after the contract deployment so that the hacker's project will be the platform project.
- The malicious user can call the function constantly to create many meaningless projects to contaminate the project pool.

## Recommendation

We recommend using whitelist for the function `launchProjectFor()` to only allow whitelisted users calling this function.

## Alleviation

The team acknowledged this issue and they stated the following:

"`launchProjectFor()` is accessible to the public without restriction by design. Anyone can launch a project on the Juicebox protocol. This is an open protocol."

## JDC-01 | Centralization Risk In JBDirectory.sol

| Category | Severity | Location | Status |
|---|---|---|---|
| **Centralization / Privilege** | ● **Major** | JBDirectory.sol | ⓘ Acknowledged |

### Description

In the contract `JBDirectory`, the role `owner` has the authority over the following function:

- function `addToSetControllerAllowlist()/removeFromSetControllerAllowlist()`: add/remove a controller to/from the trusted controller list to allow the controller to set the controller of the current directory to be another controller in the trusted list including itself.

Also, the role project owner has the authority over the following function:

- function `setControllerOf()`: update the controller that manages how terminals interact with the ecosystem,
- function `addTerminalsOf()`: add terminals to the terminal list of a specific project.

Among the previous mentioned functions which can be called by the project owner, the specific operator roles have the authority over the following function:

- The operator with the `SET_CONTROLLER` permission and the controller in the trusted list can call the function `setControllerOf()` to update the controller that manages how terminals interact with the ecosystem.
- The operator with the `addTerminalsOf` permission can call the function `addTerminalsOf()` to add terminals to the project's list of terminals.

The contract deployer has the authority over the following function:

- function `constructor()`: initialize important contract addresses to any contract addresses implementing the corresponding interfaces, for example: `operatorStore`, `projects`.

Any compromise to the privileged accounts may allow the hacker to take advantage of this authority and users' assets may suffer loss.

### Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

## Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

## Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

## Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

## Alleviation

The team acknowledged this issue and they stated the following:

"Adding and removing controllers from the allow list can be done by JuiceboxDAO members — only trusted contracts should be added.

For each project, the above-mentioned functions can only be accessed by either the address that owns the project's NFT or by operator addresses explicitly allowed by the address that owns the project's NFT." (reference: https://docs.juicebox.money/protocol/learn/glossary/operator#operatable-functionality)

# JFS-01 | Centralization Risk In JBFundingCycleStore.sol

| Category | Severity | Location | Status |
|---|---|---|---|
| **Centralization / Privilege** | ● **Major** | JBFundingCycleStore.sol | ⓘ Acknowledged |

## Description

In the contract `JBFundingCycleStore`, the role `directory.controllerOf(_projectId)` has the authority over the following function:

- function `configureFor()`: configures the next eligible funding cycle for a specified project.

The contract deployer has the authority over the following function:

- function `constructor()`: initialize the contract address `directory` to any arbitrary address.

Any compromise to the privileged accounts may allow the hacker to take advantage of this authority and users' assets may suffer loss.

## Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

## Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND

- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

## Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

## Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

## Alleviation

The team acknowledged this issue and stated the following:

"A project’s controller is the only address that has access to `configureFor()` to configure a project’s funding cycle. This is by design. It would be a major flaw if this were not the case."

# JPC-01 | Centralization Risk In JBPrices.sol

| Category | Severity | Location | Status |
|---|---|---|---|
| **Centralization / Privilege** | ● **Major** | JBPrices.sol: 109 | ⓘ Acknowledged |

## Description

In the contract `JBPrices`, the role `owner` has the authority over the following function:

- function `addFeedFor()`: add a price feed for a currency in terms of the provided base currency.

Any compromise to the privileged accounts may allow the hacker to take advantage of this authority and users' assets may suffer loss.

## Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

## Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

## Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

## Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

## Alleviation

The team acknowledged this issue and they stated the following:

"JuiceboxDAO members have the ability to add new price feeds to JBPrices through addFeedFor(). This is by design."

# JPC-02 | Third Party Dependencies Of `AggregatorV3Interface`

| Category | Severity | Location | Status |
|---|---|---|---|
| Logical Issue | ● Minor | JBPrices.sol: 39 | ⓘ Acknowledged |

## Description

The contract is serving as the underlying entity to interact with third-party AggregatorV3Interface. The scope of the audit treats 3rd party entities as black boxes and assumes their functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of 3rd parties can possibly create severe impacts, such as increasing fees of 3rd parties, migrating to new LP pools, etc.

## Recommendation

We understand that the business logic of JBPrices requires interaction with AggregatorV3Interface. We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

## Alleviation

The team acknowledged this issue and they stated the following:

"Dependence of Chainlink price feeds is by design."

# JPK-01 | Centralization Risk In JBProjects.sol

| Category | Severity | Location | Status |
|---|---|---|---|
| **Centralization / Privilege** | ● **Major** | JBProjects.sol | ⓘ Acknowledged |

## Description

In the contract `JBProjects`, the role `owner` has the authority over the following function:

- function `setTokenUriResolver()`: change the funding cycle configurations for a given project.

Also, the project owner and the operator with the `SET_METADATA` permission have the authority over the following function:

- function `setMetadataOf()`: initialize the funding cycle configurations for a given project.

Any compromise to the privileged accounts may allow the hacker to take advantage of this authority and users' assets may suffer loss.

## Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

## Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND

- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

## Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

## Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

## Alleviation

The team acknowledged this issue and they stated the following:

"A project’s owner or operators are explicitly given permission by the project’s owner can set metadata of the project. This is by design."

# JSS-01 | Centralization Risk In JBSplitsStore.sol

| Category | Severity | Location | Status |
|---|---|---|---|
| **Centralization / Privilege** | ● **Major** | JBSplitsStore.sol | ⓘ Acknowledged |

## Description

In the contract `JBSplitsStore`, the role project owner has the authority over the following function:

- function `set()`: to add a project's splits to the original ones.

Also, the operator with the `SET_SPLITS` permission has the authority over the following function:

- function `set()` to add a project's splits to the original ones.

Any compromise to the privileged accounts may allow the hacker to take advantage of this authority and users' assets may suffer loss.

## Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

## Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND

- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

## Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

## Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

## Alleviation

The team acknowledged this issue and they stated the following:

"For each project, the above-mentioned functions can only be accessed by either the address that owns the project's NFT or by operator addresses explicitly allowed by the address that owns the project's NFT." (reference: https://docs.juicebox.money/protocol/learn/glossary/operator#operatable-functionality)

# JTC-01 | Centralization Risk In JBToken.sol

| Category | Severity | Location | Status |
|---|---|---|---|
| **Centralization / Privilege** | ● **Major** | JBToken.sol | ⓘ Acknowledged |

## Description

In the contract `JBToken`, the role `owner` has the authority over the following function:

- function `mint()`: to mint arbitrary amount of new tokens,
- function `burn()`: to burn some tokens,
- function `transferOwnership()`: to transfer the `owner` privilege to the new owner.

Any compromise to the privileged accounts may allow the hacker to take advantage of this authority and users' assets may suffer loss.

## Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

## Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

## Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

## Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

## Alleviation

The team acknowledged this issue and they stated the following:

"The owner of JBToken will be the contract JBTokenStore, which should be the only address able to mint(), burn(), and transferOwnership(). This is by design."

# JTS-01 | Centralization Risk In JBTokenStore.sol

| Category | Severity | Location | Status |
|---|---|---|---|
| **Centralization / Privilege** | ● **Major** | JBTokenStore.sol | ⓘ Acknowledged |

## Description

In the contract `JBTokenStore`, the role project owner (Of `_projectId`) has the authority over the following function:

- function `shouldRequireClaimingFor()`: to allow a project to force all future mints to be claimed into the holder's wallet.

The role `holder` has the authority over the following function:

- function `transferTo()`: to transfer unclaimed tokens to another account.

Also, the operator with the `REQUIRE_CLAIM` permission has the authority over the following function:

- function `shouldRequireClaimingFor()`: to allow a project to force all future mints to be claimed into the holder's wallet.

The operator with the `TRANSFER` permission has the authority over the following function:

- function `transferTo()`: to transfer unclaimed tokens to another account.

The role `Controller` has the authority over the following function:

- function `burnFrom()`: to burn tokens.
- function `mintFor()`: to mint new tokens.
- function `changeFor()`: to swap the current project's token that is minted and burned for another, and transfer ownership of the current token to another address if needed.
- function `issueFor()`:to issues a ERC-20 token.

Any compromise to the privileged accounts may allow the hacker to take advantage of this authority and users' assets may suffer loss.

## Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

## Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

## Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

## Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

## Alleviation

The team acknowledged this issue and they stated the following:

"For each project, the above-mentioned functions can only be accessed by either the address that owns the project's NFT or by operator addresses explicitly allowed by the address that owns the project's NFT." (reference: https://docs.juicebox.money/protocol/learn/glossary/operator#operatable-functionality)

# JUI-01 | Project Contract Implementations And Parameter Settings Can Be Arbitrarily Set And Modified

| Category | Severity | Location | Status |
|---|---|---|---|
| **Centralization / Privilege** | ● **Critical** | JBController.sol<br>JBDirectory.sol | ⓘ Acknowledged |

## Description

The protocol provides a platform where everyone can create a project and become the owner of this project. The contracts composed of a given complete project can be initialized in the constructors via passing the contract addresses as parameters, however, they can be initialized to malicious contracts that implement the protocol-defined interfaces, or modified by the project owner and corresponding operators after the launch.

For example,

- JBController.constructor(), to set the `IJBOperatorStore`, `IJBProjects`, `IJBDirectory`, `IJBFundingCycleStore`, `IJBTokenStore` and `IJBSplitsStore` address
- JBDirectory.constructor(), to set the `IJBProjects` address
- JBDirectory.setControllerOf(), to modify the `IJBController` address
- JBDirectory.addTerminalsOf()/removeTerminalOf(), to modify the `IJBTerminal` addresses

Although this provides great extensibility to each project, the protocol will have no control over the created projects, and the project users' assets may suffer loss.

Even when the project owner adopts the default implementation of the project contracts, the project owner has the privilege to set all the parameters of a funding cycle, terminals, tokens, splits, and beneficiaries without any limitations. As a result, the project users may not get as many ETHs as expected when redeeming, or even worse, may not be able to redeem any ETHs. The project owner, splits and beneficiaries are able to get ETHs by the functions `distributePayoutsOf()` and `useAllowanceOf()`.

For example, if the `_percentTotal` values of all the splits are set quite low, the rest `_leftoverDistributionAmount` ETHs in the function `distributePayoutsOf()` will be transferred directly to the project owner's address, which may cause a huge loss of the project users.

## Recommendation

We would like to confirm with the client if the current implementation aligns with the original project design.

## Alleviation

The team acknowledged this issue and they stated the following:

"JuiceboxDAO has no control over each project’s behavior, and each project can roll its own extensions that can add arbitrary amounts of risk and cost alongside powerful functionality to the default protocol behavior. This is by design. Use at your own risk, and feel free to fork to offer more restrictions."

## JUI-02 | Investor Assets Are Diluted By The Reserved Token

| Category | Severity | Location | Status |
|---|---|---|---|
| Logical Issue | ● Major | JBController.sol: 537<br>JBETHPaymentTerminalStore.sol: 301~308, 722~737<br>JBETHPaymentTerminal.sol | ⓘ Acknowledged |

## Description

In the contract `JBETHPaymentTerminal` and `JBETHPaymentTerminalStore`, neither do the investors get an equivalent amount of minted project token when depositing ETHs and nor do they get an equivalent amount of ETHs when burning the project token.

The investors deposit ETH and get the minted project token by calling the function `pay()`. The mint amount is the deposited ETH amount multiplied by the weight set in the funding cycle configurations. However, the function `mintTokensOf()` in the contract `JBController` only mint a portion of the mint amount. The other portion is distributed to the reserved token `splits` by the function `distributeReservedTokensOf()` and the leftover amount (`_leftoverTokenCount`) of the tokens are minted directly to the project owner.

```
  beneficiaryTokenCount = PRBMath.mulDiv(
    _tokenCount,
    JBConstants.MAX_RESERVED_RATE - _reservedRate,
    JBConstants.MAX_RESERVED_RATE
  );

  // Mint the tokens.
  tokenStore.mintFor(_beneficiary, _projectId, beneficiaryTokenCount,
_preferClaimedTokens);
```

Also, in the function `redeemTokensOf()`, the investors only can get a portion of the overflow ETHs after the project owner distributes the ETHs to the `splits` by the function `distributePayoutsOf()`. However, the function `recordRedemptionFor()` still burns out all the `_tokenCount`.

```
uint256 _base = PRBMath.mulDiv(_currentOverflow, _tokenCount, _totalSupply);
```

```
return
  PRBMath.mulDiv(
    _base,
    _redemptionRate +
      PRBMath.mulDiv(
        _tokenCount,
        JBConstants.MAX_REDEMPTION_RATE - _redemptionRate,
        _totalSupply
      ),
    JBConstants.MAX_REDEMPTION_RATE
  );
```

directory.controllerOf(_projectId).burnTokensOf(_holder, _projectId, _tokenCount, '', false);

## Recommendation

We would like to confirm with the client if the current implementation aligns with the original project design.

## Alleviation

The team acknowledged this issue and they stated the following:

"These are by design. Contributors to projects should understand and approve of how a project is configured and controlled before making a decision to commit funds."

# JUI-03 | Calculation Issues By Wrong Divisors

| Category | Severity | Location | Status |
|---|---|---|---|
| Logical Issue | ● Minor | JBController.sol: 885~889<br>JBETHPaymentTerminal.sol: 729~731 | ⓘ Acknowledged |

## Description

The divisors used in the below calculations are confusing.

```
  PRBMath.mulDiv(
    _unprocessedTokenBalanceOf,
    JBConstants.MAX_RESERVED_RATE,
    JBConstants.MAX_RESERVED_RATE - _reservedRate
  ) - _unprocessedTokenBalanceOf;
```

```
feeAmount =
  _amount -
  PRBMath.mulDiv(_amount, JBConstants.MAX_FEE, _discountedFee + JBConstants.MAX_FEE);
```

Normally the below formula would be used:

- _unprocessedTokenBalanceOf * _reservedRate / JBConstants.MAX_RESERVED_RATE,
- _amount * (1 - _discountedFee / JBConstants.MAX_FEE).

## Recommendation

We would like to confirm with the client if the current implementation aligns with the original project design.

Financial models of blockchain protocols need to be resilient to attacks. They need to pass simulations and verifications to guarantee the security of the overall protocol.

The financial model of this protocol is not in the scope of this audit.

## Alleviation

The team acknowledged the issue and explained their design in the following doc:

`https://docs.juicebox.money/protocol/api/contracts/or-abstract/jbpayoutredemptionpaymentterminal/read/_feeamount`

# Appendix

## Finding Categories

### Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

### Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

### Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

## Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

# 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 you (“Customer” or 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, nor may copies be delivered to any other person other than the Company, without CertiK’s prior written consent in each instance.

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 CertiK 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.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED “AS IS” AND “AS

AVAILABLE” AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER’S OR ANY OTHER PERSON’S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER’S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK’S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER’S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED “AS IS” AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK’S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING

MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

# About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.

CERTIK