Introducing Code4rena Blue: Dedicated defense. Competitive bounties. Independent judging.
Learn more →

Juicebox V2 contest
Findings & Analysis Report

2022-10-11

Table of contents

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 audit contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the audit contest outlined in this document, C4 conducted an analysis of the Juicebox V2 smart contract system written in Solidity. The audit contest took place between July 1—July 8 2022.

Following the C4 audit contest, warden berndartmueller reviewed the mitigations for all identified issues; the mitigation review report is appended below the audit contest report.

Wardens

111 Wardens contributed reports to the Juicebox V2 contest:

1. berndartmueller
2. cccz
3. zzzitron
4. IIIIIII
5. Lambda
6. philogy
7. 0x52
8. hake
9. DimitarDimitrov
10. 0x29A (0x4non and rotcivegaf)
11. hubble (ksk2345 and shri4net)
12. AlleyCat
13. ylv
14. dirk_y
15. oyc_109
16. horsefacts
17. pashov

18. hyh
19. Ruhum
20. codexploder
21. Meera
22. bardamu
23. Alex the Entreprenerd
24. rbserver
25. robee
26. _141345_
27. 0xA5DF
28. 0x1f8b
29. defsec
30. Picodes
31. joestakey
32. fatherOfBlocks
33. simon135
34. jonatascm
35. 0xDjango
36. 0xNazgul
37. 0xf15ers (remora and twojoy)
38. Chom
39. GimelSec (rayn and sces60107)
40. 0xdanial
41. TomJ
42. Funen
43. Sm4rty
44. Ch_301
45. delfin454000
46. durianSausage

47. m_Rassska
48. Kaiziron
49. rajatbeladiya
50. asutorufos
51. Hawkeye (0xwags and 0xmint)
52. sach1r0
53. ReyAdmirado
54. MiloTruck
55. Bnke0x0
56. brgltd
57. Waze
58. JC
59. 0v3rf10w
60. _Adam
61. Rohan16
62. Noah3o6
63. jayfromthe13th
64. djxploit
65. 0xNineDec
66. sahar
67. svskaushik
68. TerrierLover
69. samruna
70. Chandr
71. aysha
72. 0xKitsune
73. Cheeezzyyyy
74. kebabsec (okkothejawa and FlameHorizon)
75. Limbooo

76. Saintcode_
77. RedOneN
78. c3phas
79. apostle0x01
80. UnusualTurtle
81. sashik_eth
82. JohnSmith
83. cRat1st0s
84. ajtra
85. Tutturu
86. Tomio
87. rfa
88. Metatron
89. kaden
90. ignacio
91. Aymen0909
92. Randyyy
93. mrpathfindr
94. ElKu
95. mektigboy
96. exd0tpy
97. 0x09GTO
98. Franfran
99. hansfriese
100. Green
101. tabish
102. tintin
103. cloudjunky
104. cryptphi

105. peritoflores

This contest was judged by Jack the Pug.

Mitigations reviewed by berndartmueller.

Final report assembled by itsmetechjay.

Summary

The C4 analysis yielded an aggregated total of 17 unique vulnerabilities. Of these vulnerabilities, 2 received a risk rating in the category of HIGH severity and 15 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 60 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 74 reports recommending gas optimizations.

All of the issues presented here are linked back to their original finding.

Scope

The code under review can be found within the C4 Juicebox V2 contest repository, and is composed of 10 smart contracts written in the Solidity programming language and includes 2,088 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into three primary risk categories: high, medium, and low/non-critical.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges

- Arithmetic
- Gas use

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website.

High Risk Findings (2)

[H-01] Oracle data feed can be outdated yet used anyways which will impact payment logic

Submitted by 0xNineDec, also found by 0x1f8b, 0x29A, 0x52, 0xdanial, 0xDjango, 0xf15ers, bardamu, cccz, Cheeezzyyyy, Chom, codexploder, defsec, Franfran, Alex the Entreprenerd, Green, hake, hansfriese, horsefacts, hubble, hyh, lllllll, jonatascm, kebabsec, Meera, oyc_109, pashov, rbserver, Ruhum, simon135, tabish, tintin, and zzzitron

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBChainlinkV3PriceFeed.sol#L44

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBPrices.sol#L57

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBSingleTokenPaymentTerminalStore.sol#L387

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBSingleTokenPaymentTerminalStore.sol#L585

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBSingleTokenPaymentTerminalStore.sol#L661

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBSingleTokenPaymentTerminalStore.sol#L830

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBSingleTokenPaymentTerminalStore.sol#L868

Impact

The current implementation of `JBChainlinkV3PriceFeed` is used by the protocol to showcase how the feed will be retrieved via Chainlink Data Feeds. The feed is used to retrieve the `currentPrice`, which is also used afterwards by `JBPrices.priceFor()`, then by `JBSingleTokenPaymentTerminalStore.recordPaymentFrom()`, `JBSingleTokenPaymentTerminalStore.recordDistributionFor`, `JBSingleTokenPaymentTerminalStore.recordUsedAllowanceOf`, `JBSingleTokenPaymentTerminalStore._overflowDuring` and `JBSingleTokenPaymentTerminalStore._currentTotalOverflowOf`. Although the current feeds are calculated by a non implemented IJBPriceFeed, if the implementation of the price feed is the same as the showcased in `JBChainlinkV3PriceFeed`, the retrieved data can be outdated or out of bounds.

It is important to remember that the sponsor said on the dedicated Discord Channel that also oracle pricing and data retrieval is inside the scope.

Proof of Concept

Chainlink classifies their data feeds into four different groups regarding how reliable is each source thus, how risky they are. The groups are *Verified Feeds, Monitored Feeds, Custom Feeds and Specialized Feeds* (they can be seen here). The risk is the lowest on the first one and highest on the last one.

A strong reliance on the price feeds has to be also monitored as recommended on the Risk Mitigation section. There are several reasons why a data feed may fail such as unforeseen market events, volatile market conditions, degraded performance of infrastructure, chains, or networks, upstream data providers outage, malicious activities from third parties among others.

Chainlink recommends using their data feeds along with some controls to prevent mismatches with the retrieved data. Along some recommendations, the feed can include circuit breakers (for extreme price events), contract update delays (to ensure that the injected data into the protocol is fresh enough), manual kill-switches (to cease connection in case of found bug or vulnerability in an upstream contract), monitoring (control the deviation of the data) and soak testing (of the price feeds).

The `feed.lastRoundData()` interface parameters according to Chainlink are the following:

```
function latestRoundData() external view
    returns (
        uint80 roundId,              //  The round ID.
        int256 answer,               //  The price.
        uint256 startedAt,           //  Timestamp of when the rc
        uint256 updatedAt,           //  Timestamp of when the rc
        uint80 answeredInRound       //  The round ID of the roun
    )
```

Regarding Juicebox itself, only the `answer` is used on the `JBChainlinkV3PriceFeed.currentPrice()` implementation. The retrieved price of the `priceFeed` can be outdated and used anyways as a valid data because no timestamp tolerance of the update source time is checked while storing the return parameters of `feed.latestRoundData()` inside `JBChainlinkV3PriceFeed.currentPrice()` as recommended by Chainlink in here. The usage of outdated data can impact on how the Payment terminals work regarding pricing calculation and value measurement.

Precisely the following protocol logic within `JBSingleTokenPaymentTerminalStore` will work unexpectedly regarding value management.

- `recordPaymentFrom()`:

  This function handles the minting of a project tokens according to a data source if one is given. If the retrieved value of the oracle is outdated, the `_weightRatio` at Line 387 will return an incorrect value and then the

`tokenCount` calculated amount will suffer from this mismatch, impacting in the amount of tokens minted.

- `recordDistributionFor()`:

  Performs the recording of recently distributed funds for a project. On line 580 the `distributedAmount` is computed and if the boolean check is false, then the call will perform a call to `priceFor` at line 585. If the returned oracle value is not adjusted with current market prices, the `distributedAmount` will also drag that error computing an incorrect `distributedAmount`. Afterwards, because the `distributedAmount` is also used to update the token balances of the `msg.sender` (line 598) it means that the mismatch impacts on the modified balance.

- `recordUsedAllowanceOf()`:

  Keeps record of used allowances of a project. It returns are analogue to the ones shown at `recordDistributionFor` where the `usedAmount` resembles the `distributedAmount`. The `usedAmount` is also used to update the project's balance. If the data of the oracle is outdated, the `usedAmount` will be calculated dragging that error.

- `_overflowDuring()`:

  Used to get the amount that is overflowing relative to a specified cycle. The data retrieved from the oracle is used to calculate the value of `_distributionLimitRemaining` on line 827 which is used later to calculate the return value if the boolean check performed at line 834 is true. Because the return of this function is the current balance of a project minus the amount that can be still distributed, if the amount that can still be distributed is wrong so will be the subtraction thus the return value.

- `_currentTotalOverflowOf()`:

  Similar to the latter but used to get the overflow of all the terminals of a project. If the retrieved data has a mismatch with the market, the `_totalOverflow18Decimal` calculated on line 866 if the boolean check is false

will drag this mismatch which will also be dragged into the final return of the function.

The issues of those miscalculations impact on every project currently minted, which also affects subsequently on each user that has tokens of a project resulting in a high reach impact.

Recommended Mitigation Steps

As Chainlink recommends:

> Your application should track the `latestTimestamp` variable or use the `updatedAt` value from the `latestRoundData()` function to make sure that the latest answer is recent enough for your application to use it. If your application detects that the reported answer is not updated within the heartbeat or within time limits that you determine are acceptable for your application, pause operation or switch to an alternate operation mode while identifying the cause of the delay.

> During periods of low volatility, the heartbeat triggers updates to the latest answer. Some heartbeats are configured to last several hours, so your application should check the timestamp and verify that the latest answer is recent enough for your application.

It is recommended to add a tolerance that compares the `updatedAt` return timestamp from `latestRoundData()` with the current block timestamp and ensure that the `priceFeed` is being updated with the required frequency.

If the `ETH/USD` is the only one that is needed to retrieve, because it is the most popular and available pair. It can also be useful to add other oracle to get the price feed (such as Uniswap's). This can be used as a redundancy in the case of having one oracle that returns outdated values (what is outdated and what is up to date can be determined by a tolerance as mentioned).

mejango (Juicebox) confirmed, but disagreed with severity and commented:

> There is also a good description in this duplicate #78

mejango (Juicebox) resolved:

> PR with fix: PR #1

berndartmueller (warden) reviewed mitigation:

> Appropriate validations to prevent price staleness, round incompleteness and a negative price is put in place now.

[H-02] Token Change Can Be Frontrun, Blocking Token

Submitted by philogy, also found by berndartmueller and Lambda

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBTokenStore.sol#L246

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBTokenStore.sol#L266

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBController.sol#L605

Impact

This vulnerability allows malicious actors to block other users from changing tokens of their projects. Furthermore if ownership over the token contract is transferred to the `JBTokenStore` contract prior to the change, as suggested in the recourse section of Juicebox's 24.05.2022 post-mortem update, this vulnerability would allow an attacker to become the owner of tokens being transferred. For `JBToken` based tokens this would allow an attacker to begin issuing arbitrary amounts the token that was meant to be transferred.

Proof of Concept

Exploit scenario:

1. Wanting to assign their token to their JB project an unsuspecting owner / admin transfers ownership to a `JBTokenStore` contract, either directly by calling `transferOwnership` on the token or indirectly by calling the `changeFor` method on an older `JBTokenStore` contract with `_newOwner` set as the new `JBTokenStore` contract. (For the newer Juicebox contracts the `JBController` contract's `changeTokenOf` method would be called)
2. Seeing this change an attacker submits a `changeTokenFor` calling transaction to the new `JBController` contract, triggering the `JBTokenStore` contract's `changeFor` method, linking it to one of the attacker's projects (this could be created in advance or as part of the same transaction via an attack contract)
3. The attacker can then gain ownership over the token by calling `changeTokenFor` again with the `_newOwner` set to the attacker's address
4. Assuming the token has an owner restricted `mint` method like `JBToken` based tokens the attacker can now mint an arbitrary amount of the token

Recommended Mitigation Steps

Before allowing a caller to change to a specific token ensure that they have control over it. This can be achieved by storing a list of trusted older JB directories and projects which are then queried. Alternatively the contract could require the caller to actually be the `.owner()` address of the token to migrate, this would require admins to:

1. Call `changeTokenOf` with themselves as the new owner
2. Call the new change token method on the newer contract, since they are the owner they'd pass the check
3. Independently transfer the ownership to the new token store to ensure that it can issue tokens

Future migrations can be made more seamless by having older contracts directly call new contracts via a sub-call, removing a necessary transaction for the admin. The newer contracts needs to verify that the older contract is the owner address of the token that's being set and also has approval of the project owner which is being configured.

mejango (Juicebox) confirmed and commented:

> Nice. The project should first `changeToken` and then transfer ownership.

mejango (Juicebox) resolved:

> PR with fix: PR #1

berndartmueller (warden) reviewed mitigation:

> Changing an already set project token is not possible anymore.

Medium Risk Findings (15)

[M-01] Duplicated locked splits can be discarded

Submitted by zzzitron

The function of the protocol could be impacted. This proof of concept demonstrates the discarding of one of the duplicated locked splits. In the beginning it launches a project with two identical locked splits. As the owner of the project, it updates splits to only one of the two splits. Since all of original splits are locked both of them should still in the split after the update, but only one of them exists in the updated splits.

It happens because the check of the locked split is not suitable for duplicated cases.

Please see warden's original report for full details.

Recommended Mitigation Steps

Either prevent duplicates in the splits or track the matches while checking the locked splits.

mejango (Juicebox) acknowledged, but disagreed with severity

[M-02] Lack of check on `mustStartAtOrAfter`

Submitted by zzzitron, also found by lllllll

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBFundingCycleStore.sol#L306-L312

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBFundingCycleStore.sol#L518-L522

Impact

By setting huge `mustStartAtOrAfter`, the owner can set start time in the past. It might open up possibility to bypass the ballot waiting time depending on the ballot's implementation.

Proof of Concept

The proof of concept is almost the same as `TestReconfigure::testReconfigureProject`. In the original test, the owner of the project is reconfiguring funding cycle, but it is not in effect immediately because ballot is set. Only after 3 days the newly set funding cycle will be the current one.

In the above proof of concept, only one parameter of the funding cycle is modified: `mustStartAtOrAfter` is set to `type(uint56).max`. As the result, the newly set funding cycle is considered as the current one without waiting for the ballot.

The cause of this is missing check on `mustStartAtOrAfter` upon setting here. If the given `_mustStartAtOrAfter` is huge, it will be passed eventually to the `_initFor`, `_packAndStoreIntrinsicPropertiesOf`. Then it will 'overflow' by shifting and set to the funding cycle, which essentially can be set to any value including the past. Also, it seems like the number will be also effected because the bigger digit will carry over.

```
// in JBFundingCycleStore::_packAndStoreIntrinsicPropertiesOf
// where the `_start` is derived from `_mustStartAtOrAfter`
```

```
./JBFundingCycleStore.sol-518-    // start in bits 144-199.
./JBFundingCycleStore.sol:519:    packed |= _start << 144;
./JBFundingCycleStore.sol-520-
./JBFundingCycleStore.sol-521-    // number in bits 200-255.
./JBFundingCycleStore.sol-522-    packed |= _number << 200;
```

Tools Used

Foundry

Recommended Mitigation Steps

Add a check for the `_mustStartAtOrAfter`:

```
// example check for _mustSTartAtOrAfter
// in JBFundingCycleStore::configureFor

if (_mustStartAtOrAfter > type(uint56).max) revert INVALID_START
```

drgorillamd (Juicebox) confirmed and commented:

> We've seen the POC, now assessing how to best mitigate (at what level).

jack-the-pug (judge) commented:

> Good catch!

drgorillamd (Juicebox) resolved:

> PR with fix: PR #1

berndartmueller (warden) reviewed mitigation:

> `mustStartAtOrAfter` and the start date of an upcoming funding cycle are now validated to fit in `uint56`.

[M-03] Use a safe transfer helper library for ERC20 transfers

Submitted by horsefacts, also found by 0x1f8b, 0x29A, 0x52, 0xf15ers, AlleyCat, apostle0x01, berndartmueller, cccz, Ch_301, Chom, cloudjunky, codexploder, cryptphi, delfin454000, durianSausage, fatherOfBlocks, Franfran, hake, hansfriese, hyh, IIIIIII, jonatascm, Kaiziron, Limbooo, m_Rassska, Meera, oyc_109, peritoflores, rajatbeladiya, rbserver, Ruhum, Sm4rty, svskaushik, and zzzitron

`JBERC20PaymentTerminal#_transferFrom` calls `IERC20#transfer` and `transferFrom` directly. There are two issues with using this interface directly:

1. `JBERC20PaymentTerminal#_transferFrom` function does not check the return value of these calls. Tokens that return `false` rather than revert to indicate failed transfers may silently fail rather than reverting as expected.
2. Since the IERC20 interface requires a boolean return value, attempting to transfer ERC20s with missing return values will revert. This means Juicebox payment terminals cannot support a number of popular ERC20s, including USDT and BNB.

`JBERC20PaymentTerminal#_transferFrom`:

```
    function _transferFrom(
      address _from,
      address payable _to,
      uint256 _amount
    ) internal override {
      _from == address(this)
        ? IERC20(token).transfer(_to, _amount)
        : IERC20(token).transferFrom(_from, _to, _amount);
    }
```

Impact

Juicebox payment terminals may issue project tokens to users even though their incoming token transfer failed. Juicebox payment terminals cannot support USDT, BNB, and other popular (but nonstandard) ERC20s.

Recommended Mitigation Steps

Use a safe transfer library like OpenZeppelin SafeERC20 to ensure consistent handling of ERC20 return values and abstract over inconsistent ERC20

implementations.

Additionally, since payment terminals are meant to support a variety of ERC20s, consider writing simulation tests that make token transfers using payment terminals for the most popular and most unusual ERC20s.

(Note also that the out of scope `JBETHERC20ProjectPayer` and `JBETHERC20SplitsPayer` contracts also call `IERC20#transfer` and `transferFrom` without a helper!)

See the following Forge test, which simulates an attempted USDT transfer. (Run this in fork mode using the `--fork-url` flag).

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.6;

import './helpers/TestBaseWorkflow.sol';
import '@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol'

address constant USDT_ADDRESS = address(0xdAC17F958D2ee523a22062

contract TestWeirdERC20 is TestBaseWorkflow {
  using SafeERC20 for IERC20Metadata;

  JBController controller;
  JBProjectMetadata _projectMetadata;
  JBFundingCycleData _data;
  JBFundingCycleMetadata _metadata;
  JBGroupedSplits[] _groupedSplits;
  JBFundAccessConstraints[] _fundAccessConstraints;
  IJBPaymentTerminal[] _terminals;
  JBTokenStore _tokenStore;
  JBERC20PaymentTerminal _tetherTerminal;

  IERC20Metadata usdt = IERC20Metadata(USDT_ADDRESS);
  address _projectOwner;

  uint256 WEIGHT = 1000 * 10**18;

  function setUp() public override {
    super.setUp();
```

```
_projectOwner = multisig();

_tokenStore = jbTokenStore();

controller = jbController();

_projectMetadata = JBProjectMetadata({content: 'myIPFSHash',

_data = JBFundingCycleData({
  duration: 14,
  weight: WEIGHT,
  discountRate: 450000000,
  ballot: IJBFundingCycleBallot(address(0))
});

_metadata = JBFundingCycleMetadata({
  global: JBGlobalFundingCycleMetadata({allowSetTerminals: f
  reservedRate: 5000, //50%
  redemptionRate: 5000, //50%
  ballotRedemptionRate: 0,
  pausePay: false,
  pauseDistributions: false,
  pauseRedeem: false,
  pauseBurn: false,
  allowMinting: false,
  allowChangeToken: false,
  allowTerminalMigration: false,
  allowControllerMigration: false,
  holdFees: false,
  useTotalOverflowForRedemptions: false,
  useDataSourceForPay: false,
  useDataSourceForRedeem: false,
  dataSource: address(0)
});

_tetherTerminal = new JBERC20PaymentTerminal(
  usdt,
  jbLibraries().ETH(), // currency
  jbLibraries().ETH(), // base weight currency
  1, // JBSplitsGroupe
  jbOperatorStore(),
  jbProjects(),
  jbDirectory(),
  jbSplitsStore(),
  jbPrices(),
  jbPaymentTerminalStore(),
```

```
      multisig()
    );
    evm.label(address(_tetherTerminal), 'TetherTerminal');

    _terminals.push(_tetherTerminal);
  }

  function testTetherPaymentsRevert() public {
    JBERC20PaymentTerminal terminal = _tetherTerminal;

    _fundAccessConstraints.push(
      JBFundAccessConstraints({
        terminal: terminal,
        token: address(USDT_ADDRESS),
        distributionLimit: 10 * 10**18,
        overflowAllowance: 5 * 10**18,
        distributionLimitCurrency: jbLibraries().ETH(),
        overflowAllowanceCurrency: jbLibraries().ETH()
      })
    );

    uint256 projectId = controller.launchProjectFor(
      _projectOwner,
      _projectMetadata,
      _data,
      _metadata,
      block.timestamp,
      _groupedSplits,
      _fundAccessConstraints,
      _terminals,
      ''
    );

    address caller = msg.sender;
    evm.label(caller, 'caller');
    deal(address(usdt), caller, 20 * 10**18);

    evm.prank(caller);
    usdt.safeApprove(address(terminal), 20 * 10**18);
    evm.prank(caller);
    terminal.pay(
      projectId,
      20 * 10**18,
      address(usdt),
      msg.sender,
      0,
```

```
            false,
            'Forge test',
            new bytes(0)
        );
    }
}
```

[mejango (Juicebox) confirmed]

mejango (Juicebox) resolved:

> PR with fix: [PR #1]

berndartmueller (warden) reviewed mitigation:

> OpenZeppelins' `SafeERC20` library is now used to ensure consistent handling of ERC20 token transfers.

[M-04] Juicebox project owner can create a honeypot to cause grief

Submitted by dirk_y, also found by lllllll, and ylv

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBController.sol#L760

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBSplitsStore.sol#L147

Impact

In a Juicebox project the project owner (or anyone that they approve) can set splits. These splits are details of the token distributions to other addresses in response to contributions to the project.

At the moment the `SPLITS_TOTAL_PERCENT = 1_000_000_000`. This means that the project owner could theoretically add 1 billion different splits, each with a percent value of 1. Of course, this would require too much gas, but the idea stands. A project owner could honeypot users by creating a project with the `MAX_RESERVED_RATE` reserved rate, and setting a large percentage split for the `msg.sender` who calls `distributeReservedTokensOf` in `JBController.sol`. The project owner could then fund the project with a series of large payments to ensure that the reserved amount was sufficiently large to entice a user to call `distributeReservedTokensOf` in the belief that they will be obtaining a large percentage of the reserve.

However, when a user calls this method they will hit the block gas limit and will have spent a large amount of ETH on gas, without receiving any of their expected split.

I consider this to be of high severity since user assets (in the form of gas) can be permanently lost without any loss to the project owner/griefer.

Proof of Concept

The key behaviour we need to prove is that it's possible to set more splits before hitting the block gas limit than it is to distribute reward tokens over the same number of splits. If this is true, the project owner will be able to set a number of splits that will always make the `distributeReservedTokensOf` hit the block gas limit, and hence grief the caller.

This can be demonstrated by modifying the existing test cases. From some basic testing I have found that calling `distributeReservedTokensOf` hits the block gas limit when there are at least 389 splits, but for the same split count the project owner can successfully call `set` without hitting the block gas limit.

```
diff --git a/test/jb_controller/distribute_reserved_token_of.tes
_token_of.test.js
index 2f964d8..6cfd645 100644
--- a/test/jb_controller/distribute_reserved_token_of.test.js
+++ b/test/jb_controller/distribute_reserved_token_of.test.js
@@ -119,10 +119,15 @@ describe('JBController::distributeReservec
     const { addrs, projectOwner, jbController, mockJbTokenStore
       await setup();
     const caller = addrs[0];
-    const splitsBeneficiariesAddresses = [addrs[1], addrs[2]].m
```

```
+    let addressList = [addrs[1], addrs[2]];
+    for (let i = 1; i < 389; i++) {
+      addressList.push(addrs[1]);
+    }
+
+    const splitsBeneficiariesAddresses = addressList.map((signe

     const splits = makeSplits({
-      count: 2,
+      count: 389,
       beneficiary: splitsBeneficiariesAddresses,
       preferClaimed: true,
     });
diff --git a/test/jb_splits_store/set.test.js b/test/jb_splits_s
index 3dd0331..5992957 100644
--- a/test/jb_splits_store/set.test.js
+++ b/test/jb_splits_store/set.test.js
@@ -54,7 +54,7 @@ describe('JBSplitsStore::set(...)', function (
     };
   }

-  function makeSplits(beneficiaryAddress, count = 4) {
+  function makeSplits(beneficiaryAddress, count = 389) {
     let splits = [];
     for (let i = 0; i < count; i++) {
       splits.push({
```

Tools Used

VSCode & Hardhat

Recommended Mitigation Steps

For `JBSplit` objects there should be a minimum percentage for each split when calling `set`. Furthermore, it would probably be wise to prevent duplicate beneficiaries, but I have omitted that in the below recommendation for clarity. Below is a suggested diff. I've arbitrarily set a minimum percentage of 10,000 but given the PoC the min percentage should be conservatively set to ensure no more than 389 splits can be created (I would probably suggest a cap of max 100 splits per group).

```
diff --git a/contracts/JBSplitsStore.sol b/contracts/JBSplitsSto
index d61cca2..429d78a 100644
```

```
--- a/contracts/JBSplitsStore.sol
+++ b/contracts/JBSplitsStore.sol
@@ -227,8 +227,8 @@ contract JBSplitsStore is IJBSplitsStore, JB
     uint256 _percentTotal = 0;

     for (uint256 _i = 0; _i < _splits.length; _i++) {
-      // The percent should be greater than 0.
-      if (_splits[_i].percent == 0) revert INVALID_SPLIT_PERCEN
+      // The percent should be greater than or equal to 10000.
+      if (_splits[_i].percent < JBConstants.MIN_SPLIT_PERCENT)

       // ProjectId should be within a uint56
       if (_splits[_i].projectId > type(uint56).max) revert INVA
diff --git a/contracts/libraries/JBConstants.sol b/contracts/lib
index 9a418f2..afb5f23 100644
--- a/contracts/libraries/JBConstants.sol
+++ b/contracts/libraries/JBConstants.sol
@@ -10,6 +10,7 @@ library JBConstants {
   uint256 public constant MAX_REDEMPTION_RATE = 10000;
   uint256 public constant MAX_DISCOUNT_RATE = 1000000000;
   uint256 public constant SPLITS_TOTAL_PERCENT = 1000000000;
+  uint256 public constant MIN_SPLIT_PERCENT = 10000;
   uint256 public constant MAX_FEE = 1000000000;
   uint256 public constant MAX_FEE_DISCOUNT = 1000000000;
 }
```

An alternative to setting a minimum percentage would be to have a check on the length of the splits array and capping that at a sensible value. In this instance a project owner could still set low percentages per split, however I don't personally see the value in being able to set a value of 1 (to receive 1 billionth of the reserve).

mejango (Juicebox) acknowledged, but disagreed with severity and commented:

> Damn. Word. This is deep. Thank you.

> Not sure about "high" severity. But surely should be noted among the protocol's risks.

jack-the-pug (judge) decreased severity to Medium and commented:

> Not bad, but also not a High. This is similar to the unbounded loop and other out-of-gas issues, the honeypot probably wont work if the wallet UI is better (alerts

> about the out-of-gas error).

> Will downgrade to Medium.

[M-05] Discounted fee calculation is imprecise and calculates less fees than anticipated

Submitted by berndartmueller, also found by 0x52, hyh, and Ruhum

The `JBPayoutRedemptionPaymentTerminal._feeAmount` function is used to calculate the fee based on a given `_amount`, a fee rate `_fee` and an optional discount `_feeDiscount`.

However, the current implementation calculates the fee in a way that leads to inaccuracy and to fewer fees being paid than anticipated by the protocol.

Proof of Concept

[JBPayoutRedemptionPaymentTerminal._feeAmount](#)

```
    function _feeAmount(
        uint256 _amount,
        uint256 _fee,
        uint256 _feeDiscount
      ) internal pure returns (uint256) {
        // Calculate the discounted fee.
        uint256 _discountedFee = _fee -
          PRBMath.mulDiv(_fee, _feeDiscount, JBConstants.MAX_FEE_DIS

        // The amount of tokens from the `_amount` to pay as a fee.
        return
          _amount - PRBMath.mulDiv(_amount, JBConstants.MAX_FEE, _di
      }
```

Example:

Given the following (don't mind the floating point arithmetic, this is only for simplicity. The issues still applies with integer arithmetic and higher decimal precision):

- `amount` - 1000
- `fee` - 5 (5%)
- `feeDiscount` - 10 (10%)
- `MAX_FEE_DISCOUNT` - 100
- `MAX_FEE` - 100

$discountedFee = fee - {{fee \ast feeDiscount} \over MAX_FEE_DISCOUNT}$
$discountedFee = 5 - {{5 \ast 10} \over 100}$
$discountedFee = 4.5$

Calculating the fee amount based on the discounted fee of 4.5:

$fee\_{Amount} = amount - {{amount \ast MAX_FEE} \over {discountedFee + MAX_FEE}}$
$fee\_{Amount} = 1000 - {{1000 \ast 100} \over {4.5 + 100}}$
$fee\_{Amount} = 1000 - 956.93779904$
$fee\_{Amount} = 43.06220096$

The calculated and wrong fee amount is `~43`, instead, it should be `45`. The issue comes from dividing by `_discountedFee + JBConstants.MAX_FEE`.

Now the correct way:

I omitted the `discountedFee` calculation as this formula is correct.

$fee\_{Amount} = {{amount \ast discountedFee} \over {MAX_FEE}}$
$fee\_{Amount} = {{1000 \ast 4.5} \over {100}}$
$fee\_{Amount} = 45$

Recommended Mitigation Steps

Fix the discounted fee calculation by adjusting the formula to:

$fee\_{Amount} = amount \ast {fee - fee \ast {discount \over MAX\_{FEE_DISCOUNT}} \over MAX\_{FEE}}$

In Solidity:

```
    function _feeAmount(
        uint256 _amount,
        uint256 _fee,
        uint256 _feeDiscount
    ) internal pure returns (uint256) {
        // Calculate the discounted fee.
        uint256 _discountedFee = _fee -
          PRBMath.mulDiv(_fee, _feeDiscount, JBConstants.MAX_FEE_DIS

        // The amount of tokens from the `_amount` to pay as a fee.
        return PRBMath.mulDiv(_amount, _discountedFee, JBConstants.M
    }
```

mejango (Juicebox) acknowledged

jack-the-pug (judge) commented:

> Great job! One of the best write-ups I have ever seen, simple and clean.

> Here is a trophy for you: 🏆

[M-06] Code credits fee-on-transfer tokens for amount stated, not amount transferred

Submitted by lllllll, also found by cccz, hake, Meera, rbserver, and robee

Some ERC20 tokens, such as USDT, allow for charging a fee any time `transfer()` or `transferFrom()` is called. If a contract does not allow for amounts to change after transfers, subsequent transfer operations based on the original amount will `revert()` due to the contract having an insufficient balance.

Impact

If there is only one user that has use a payment terminal with a fee-on-transfer token to pay a project for its token, that project will be unable to withdraw their funds, because the amount available will be less than the amount stated during deposit, and therefore the token's `transfer()` call will revert during withdrawal. For more users, consider what happens if the token has a 10% fee-on-transfer fee - deposits

will be underfunded by 10%, and the projects trying to withdraw the last 10% of deposits/rewards will have their calls revert due to the contract not holding enough tokens. If a whale does a large withdrawal, the extra 10% that that whale gets will mean that *many* projects will not be able to withdraw anything at all.

Proof of Concept

Because the terminals rely on terminal stores, which only store the initial value provided during the payment, and provide it during distributions, the terminals are unable to use the decreased value when they later are told to distribute funds to a project.

`JBSingleTokenPaymentTerminalStore.recordPaymentFrom()` stores the value passed in:

```
File: contracts/JBSingleTokenPaymentTerminalStore.sol   #1

372       // Add the amount to the token balance of the project.
373       balanceOf[IJBSingleTokenPaymentTerminal(msg.sender)][_
374         balanceOf[IJBSingleTokenPaymentTerminal(msg.sender)]
375         _amount.value;
```

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBSingleTokenPaymentTerminalStore.sol#L372-L375

And provide that same value when recording a dispersion:

```
File: contracts/JBSingleTokenPaymentTerminalStore.sol   #2

597       // Removed the distributed funds from the project's to
598       balanceOf[IJBSingleTokenPaymentTerminal(msg.sender)][_
599         balanceOf[IJBSingleTokenPaymentTerminal(msg.sender)]
600         distributedAmount;
```

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBSingleTokenPaymentTerminalStore.sol#L597-L600

The terminals themselves use the values directly, and don't consult their balances to look for changes (lines 817 and 850 below):

File: contracts/abstract/JBPayoutRedemptionPaymentTerminal.sol

```
817       (JBFundingCycle memory _fundingCycle, uint256 _distrib
818         _projectId,
819         _amount,
820         _currency
821       );
822
823       // The amount being distributed must be at least as mu
824       if (_distributedAmount < _minReturnedTokens) revert IN
825
826       // Get a reference to the project owner, which will re
827       // and receive any extra distributable funds not alloc
828       address payable _projectOwner = payable(projects.owner
829
830       // Define variables that will be needed outside the sc
831       // Keep a reference to the fee amount that was paid.
832       uint256 _fee;
833
834       // Scoped section prevents stack too deep. `_feeDiscou
835       {
836         // Get the amount of discount that should be applied
837         // If the fee is zero or if the fee is being used by
838         uint256 _feeDiscount = fee == 0 || isFeelessAddress[
839           ? JBConstants.MAX_FEE_DISCOUNT
840           : _currentFeeDiscount(_projectId);
841
842         // The amount distributed that is eligible for incur
843         uint256 _feeEligibleDistributionAmount;
844
845         // The amount leftover after distributing to the spl
846         uint256 _leftoverDistributionAmount;
847
848         // Payout to splits and get a reference to the leftc
849         // Also get a reference to the amount that was distr
850         (_leftoverDistributionAmount, _feeEligibleDistributi
851           _projectId,
852           _fundingCycle.configuration,
853           payoutSplitsGroup,
854           _distributedAmount,
855           _feeDiscount
```

```
856         );
```

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/abstract/JBPayoutRedemptionPaymentTerminal.sol#L817-L856

The terminals used the amounts stated, rather than transferred in (lines 349 and 356):

File: contracts/abstract/JBPayoutRedemptionPaymentTerminal.sol

```
332     function pay(
333       uint256 _projectId,
334       uint256 _amount,
335       address _token,
336       address _beneficiary,
337       uint256 _minReturnedTokens,
338       bool _preferClaimedTokens,
339       string calldata _memo,
340       bytes calldata _metadata
341     ) external payable virtual override isTerminalOf(_projec
342       _token; // Prevents unused var compiler and natspec cc
343 
344       // ETH shouldn't be sent if this terminal's token isn'
345       if (token != JBTokens.ETH) {
346         if (msg.value > 0) revert NO_MSG_VALUE_ALLOWED();
347 
348         // Transfer tokens to this terminal from the msg ser
349         _transferFrom(msg.sender, payable(address(this)), _a
350       }
351       // If this terminal's token is ETH, override _amount w
352       else _amount = msg.value;
353 
354       return
355         _pay(
356           _amount,
357           msg.sender,
358           _projectId,
359           _beneficiary,
360           _minReturnedTokens,
361           _preferClaimedTokens,
362           _memo,
363           _metadata
```

```
364         );
365     }
```

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/abstract/JBPayoutRedemptionPaymentTerminal.sol#L332-L365

Same here (lines 555 and 561):

```
File: contracts/abstract/JBPayoutRedemptionPaymentTerminal.sol

540     function addToBalanceOf(
541       uint256 _projectId,
542       uint256 _amount,
543       address _token,
544       string calldata _memo,
545       bytes calldata _metadata
546     ) external payable virtual override isTerminalOf(_projec
547       _token; // Prevents unused var compiler and natspec cc
548 
549       // If this terminal's token isn't ETH, make sure no ms
550       if (token != JBTokens.ETH) {
551         // Amount must be greater than 0.
552         if (msg.value > 0) revert NO_MSG_VALUE_ALLOWED();
553 
554         // Transfer tokens to this terminal from the msg ser
555         _transferFrom(msg.sender, payable(address(this)), _a
556       }
557       // If the terminal's token is ETH, override `_amount`
558       else _amount = msg.value;
559 
560       // Add to balance while only refunding held fees if th
561       _addToBalanceOf(_projectId, _amount, !isFeelessAddress
562     }
```

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/abstract/JBPayoutRedemptionPaymentTerminal.sol#L540-L562

The transfer of fees and reserves have the same issue.

Recommended Mitigation Steps

Measure the contract balance before and after the call to `transfer()` / `transferFrom()` in `JBERC20PaymentTerminal._transferFrom()`, and use the difference between the two as the amount, rather than the amount stated

drgorillamd (Juicebox) acknowledged

drgorillamd (Juicebox) resolved:

> PR with fix: PR #1

berndartmueller (warden) reviewed mitigation:

> The delta of the token balance before and after a transfer is used instead of the amount stated to handle fee-on-transfer tokens appropriately.

[M-07] processFees() may fail due to exceed gas limit

Submitted by oyc_109, also found by 0x52, lllllll, and pashov

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/abstract/JBPayoutRedemptionPaymentTerminal.sol#L594

Impact

The function `processFees()` in `JBPayoutRedemptionPaymentTerminal.sol` may fail due to unbounded loop over `_heldFeesOf[_projectId]`

`_heldFeesOf[_projectId]` can get very large due to the function `_takeFeeFrom()` where it pushes fees that should be paid to a specific beneficiary onto the array

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/abstract/JBPayoutRedemptionPaymentTerminal.sol#L1199

`_heldFeesOf[_projectId]` could get large and cause a DOS condition where no fees can be distributed due to exceed of gas limit

Proof of Concept

```
        for (uint256 _i = 0; _i < _heldFeeLength; ) {
          // Get the fee amount.
          uint256 _amount = _feeAmount(
            _heldFees[_i].amount,
            _heldFees[_i].fee,
            _heldFees[_i].feeDiscount
          );
```

mejango (Juicebox) acknowledged

[M-08] Reentrancy issues on function `distributePayoutsOf`

Submitted by 0x29A, also found by AlleyCat and hubble

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/abstract/JBPayoutRedemptionPaymentTerminal.sol#L415-L448

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/abstract/JBPayoutRedemptionPaymentTerminal.sol#L788-L900

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/abstract/JBPayoutRedemptionPaymentTerminal.sol#L981-L1174

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBETHPaymentTerminal.sol#L63-L79

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBER

C20PaymentTerminal.sol#L73-L89

Impact

In the contract `JBPayoutRedemptionPaymentTerminal`, the function `distributePayoutsOf` calls the internal function `_distributePayoutsOf` and this internal function perfoms a loop where is using the function `_distributeToPayoutSplitsOf` In these functions there are a `_transferFrom` what:

- `JBETHPaymentTerminal` using a `Address.sendValue(_to, _amount)`
- `JBERC20PaymentTerminal` using a `IERC20(token).transfer(_to, _amount)` with a `ERC777` as token

Both give back the control to the `msg.sender` (`_to` variable) creating a reentrancy attack vector.

Also could end with a lot of bad calculation because is using uncheckeds statements and function `_distributePayoutsOf` its no respecting the `checks, effects, interactions` pattern.

Proof of Concept

Craft a contract to call function `distributePayoutsOf`, on receive ether reentrant to function `distributePayoutsOf` or use a `ERC777` callback.

Recommended Mitigation Steps

Add a reentrancyGuard as you do on `JBSingleTokenPaymentTerminalStore.sol`;
You have already imported the ReentrancyGuard on JBPayoutRedemptionPaymentTerminal.sol#L5 but you are not using it.

My recommendation is to add `nonReentrant` modifier on function `distributePayoutsOf`.

drgorillamd (Juicebox) acknowledged

jack-the-pug (judge) commented:

> Lack of clear path to exploit it, but it does seem like `_distributeToPayoutSplitsOf` can be used to reenter `distributePayoutsOf`; it requires the attacker to be one of the project's splits beneficiaries, though.

> https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/abstract/JBPayoutRedemptionPaymentTerminal.sol#L1148-L1153

```
_transferFrom(
  address(this),
  _split.beneficiary != address(0) ? _split.beneficiar
  _netPayoutAmount
);
```

[M-09] Unhandled chainlink revert would lock all price oracle access

Submitted by bardamu, also found by berndartmueller, codexploder, Alex the Entreprenerd, and horsefacts

Call to `latestRoundData` could potentially revert and make it impossible to query any prices. Feeds cannot be changed after they are configured (https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBPrices.sol#L115) so this would result in a permanent denial of service.

Proof of Concept

Chainlink's multisigs can immediately block access to price feeds at will. Therefore, to prevent denial of service scenarios, it is recommended to query Chainlink price feeds using a defensive approach with Solidity's try/catch structure. In this way, if the call to the price feed fails, the caller contract is still in control and can handle any errors safely and explicitly.

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBPrices.sol#L69

```
if (_feed != IJBPriceFeed(address(0))) return _feed.currentPrice
```

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBChainlinkV3PriceFeed.sol#L42-L44

```
function currentPrice(uint256 _decimals) external view override
  // Get the latest round information. Only need the price is ne
  (, int256 _price, , , ) = feed.latestRoundData();
```

Refer to https://blog.openzeppelin.com/secure-smart-contract-guidelines-the-dangers-of-price-oracles/ for more information regarding potential risks to account for when relying on external price feed providers.

Tools Used

VIM

Recommended Mitigation Steps

Surround the call to `latestRoundData()` with `try/catch` instead of calling it directly. In a scenario where the call reverts, the catch block can be used to call a fallback oracle or handle the error in any other suitable way.

mejango (Juicebox) acknowledged

jack-the-pug (judge) commented:

> Good catch! Seems like we should update this function to allow changing the feed contract:

> https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBPrices.sol#L109-L121

[M-10] Grieffer beneficiary can cause DOS

Submitted by hake, also found by cccz

Payouts won't be able to be distributed if one of multiple beneficiaries decides to revert the transaction on receival.

Proof of Concept

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/abstract/JBPayoutRedemptionPaymentTerminal.sol#L1147-L1152

```
    // If there's a beneficiary, send the funds directly to the bene
            _transferFrom(
              address(this),
              _split.beneficiary != address(0) ? _split.beneficiar
              _netPayoutAmount
            );
```

If token used is native ETH or ERC777 a beneficiary can revert the transaction on the callback and DOS `_distributeToPayoutSplitsOf()` for all the other beneficiaries.

Recommended Mitigation Steps

Have beneficiaries withdraw their benefit instead of sending it to them.

mejango (Juicebox) acknowledged and commented:

> By design. Project owners bring their own risks and opportunities when setting payout splits. Made clear here.

hake (warden) commented:

> A malicious or compromised beneficiary is not exactly under a project owner's control. Implementing the recommended mitigation step would prevent the possibility of DOS while maintaining all privileges of project owner. No risks outlined in link below would be mitigated by the recommended mitigation, thus project owner would still have access to same range of functionalities. https://info.juicebox.money/dev/learn/risks/#setting-a-distribution-limit-and-payout-splits

[M-11] addFeedFor should check if inverse feed already exists

Submitted by 0x52, also found by DimitarDimitrov

Potentially inconsistent currency conversions.

Proof of Concept

`addFeedFor` requires that a price feed for the _currency _base doesn't exist when adding a new price feed but doesn't check if the inverse already exists. This means that two different oracles (potentially with different prices) could be used for _currency -> _base vs. _base -> _currency. Different prices would lead to inconsistent between conversion ratios depending on the direction of the conversion.

Recommended Mitigation Steps

Change L115 to: `if (feedFor\[\_currency]\[\_base] != IJBPriceFeed(address(0)) || feedFor\[\_base]\[\_currency] != IJBPriceFeed(address(0))) revert PRICE_FEED_ALREADY_EXISTS()`

mejango (Juicebox) confirmed

mejango (Juicebox) resolved:

> PR with fix: PR #1

berndartmueller (warden) reviewed mitigation:

> An additional check has been added to prevent adding a price feed for the inverse pair.

[M-12] changeTokenOf makes it impossible for holders of oldToken to redeem the overflowed assets.

Submitted by cccz

When the owner calls the changeTokenOf function of the JBController contract, the token corresponding to the current project will be changed, which will make the oldToken holder unable to redeem the overflowing assets.

Proof of Concept

https://github.com/jbx-protocol/juice-contracts-v2-code4rena//blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBController.sol#L588-L606

https://github.com/jbx-protocol/juice-contracts-v2-code4rena//blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBTokenStore.sol#L236-L269

Recommended Mitigation Steps

Consider adding a delay to `changeTokenOf`, or adding a function to convert `oldToken` to `newToken`.

mejango (Juicebox) confirmed

mejango (Juicebox) resolved:

> PR with fix: PR #1

berndartmueller (warden) reviewed mitigation:

> Once a token is set for a project, it can not be changed anymore.

[M-13] JBToken: mint function could mint arbitrary amount of tokens

Submitted by cccz

The owner of the JBToken contract can mint arbitrary amount of tokens.

In general, the owner of the JBToken contract is the JBTokenStore contract, and the minting of the tokens is controlled by the JBController contract, but when the

changeTokenOf function of the JBController contract is called, the owner will be transferred to any address, which can mint arbitrary amount of tokens.

```
function mint(
  uint256 _projectId,
  address _account,
  uint256 _amount
) external override onlyOwner {
  _projectId; // Prevents unused var compiler and natspec comp

  return _mint(_account, _amount);
}
```

Proof of Concept

https://github.com/jbx-protocol/juice-contracts-v2-code4rena//blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBTokenStore.sol#L106-L114

Recommended Mitigation Steps

Consider setting minter as the JBTokenStore contract and adding the onlyminter modifier to the mint function.

mejango (Juicebox) acknowledged

[M-14] More outstanding reserved tokens are distributed than anticipated leading to less redeemable assets and therefore loss of user funds

Submitted by berndartmueller

The `JBController.distributeReservedTokensOf` function is used to distribute all outstanding reserved tokens for a project. Internally, the `JBController._distributeReservedTokensOf` function calculates the distributable amount of tokens `tokenCount` with the function `JBController._reservedTokenAmountFrom`.

However, the current implementation of `JBController._reservedTokenAmountFrom` calculates the amount of reserved tokens currently tracked in a way that leads to inaccuracy and to more tokens distributed than anticipated.

Impact

More tokens than publicly defined via the funding cycle `reservedRate` are distributed (minted) to the splits and the owner increasing the total supply and therefore reducing the amount of terminal assets redeemable by a user. The increased supply takes effect in `JBSingleTokenPaymentTerminStore.recordRedemptionFor` on L784. The higher the token supply, the less terminal assets redeemable.

Proof of Concept

JBController._reservedTokenAmountFrom

```
    function _reservedTokenAmountFrom(
        int256 _processedTokenTracker,
        uint256 _reservedRate,
        uint256 _totalEligibleTokens
      ) internal pure returns (uint256) {
        // Get a reference to the amount of tokens that are unproces
        uint256 _unprocessedTokenBalanceOf = _processedTokenTracker
          ? _totalEligibleTokens - uint256(_processedTokenTracker)
          : _totalEligibleTokens + uint256(-_processedTokenTracker);

        // If there are no unprocessed tokens, return.
        if (_unprocessedTokenBalanceOf == 0) return 0;

        // If all tokens are reserved, return the full unprocessed a
        if (_reservedRate == JBConstants.MAX_RESERVED_RATE) return _

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

Example:

Given the following (don't mind the floating point arithmetic, this is only for simplicity. The issues still applies with integer arithmetic and higher decimal precision):

- `processedTokenTracker` - `-1000`
- `reservedRate` - 10 (10%)
- `totalEligibleTokens` - 0
- `MAX_RESERVED_RATE` - 100

$unprocessedTokenBalanceOf = 0 + (--1000)$
$unprocessedTokenBalanceOf = 1000$

$reservedTokenAmount = {{unprocessedTokenBalanceOf \ast MAX_RESERVED_RATE} \over {MAX_RESERVED_RATE - reservedRate}} - unprocessedTokenBalanceOf$
$reservedTokenAmount = {{1000 \ast 100} \over {100 - 10}} - 1000$
$reservedTokenAmount = 1111.111 - 1000$
$reservedTokenAmount = 111,111$

The calculated and wrong amount is `~111`, instead it should be `100` (10% of 1000). The issue comes from dividing by `JBConstants.MAX_RESERVED_RATE - _reservedRate`.

Now the correct way:

$reservedTokenAmount = {{unprocessedTokenBalanceOf \ast reservedRate} \over MAX_RESERVED_RATE}$
$reservedTokenAmount = {{1000 \ast 10} \over 100}$
$reservedTokenAmount = 100$

Recommended Mitigation Steps

Fix the outstanding reserve token calculation by implementing the calculation as following:

```
  function _reservedTokenAmountFrom(
      int256 _processedTokenTracker,
      uint256 _reservedRate,
      uint256 _totalEligibleTokens
  ) internal pure returns (uint256) {
    // Get a reference to the amount of tokens that are unprocesse
    uint256 _unprocessedTokenBalanceOf = _processedTokenTracker >=
      ? _totalEligibleTokens - uint256(_processedTokenTracker)
      : _totalEligibleTokens + uint256(-_processedTokenTracker);

    // If there are no unprocessed tokens, return.
    if (_unprocessedTokenBalanceOf == 0) return 0;

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

[mejango (Juicebox) disputed and commented]():

> The only case where the tracker can be -1000 but the totalEligibleTokens is 0 is if reserved rate is 100%. https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBController.sol#L664
>
> ```
> if (_reservedRate == JBConstants.MAX_RESERVED_RATE)
>   // Subtract the total weighted amount from the tracker so the full reserved token amount can be printed later.
>   _processedTokenTrackerOf[_projectId] =
>     _processedTokenTrackerOf[_projectId] -
>     int256(_tokenCount);
> else {
>   // The unreserved token count that will be minted for the beneficiary.
>   beneficiaryTokenCount = PRBMath.mulDiv(
>     _tokenCount,
>     JBConstants.MAX_RESERVED_RATE - _reservedRate,
>     JBConstants.MAX_RESERVED_RATE
>   );
>
>   if (_reservedRate == 0)
>     // If there's no reserved rate, increment the tracker with the newly minted tokens.
>     _processedTokenTrackerOf[_projectId] =
>       _processedTokenTrackerOf[_projectId] +
>       int256(beneficiaryTokenCount);
>
>   // Mint the tokens.
>   tokenStore.mintFor(_beneficiary, _projectId, beneficiaryTokenCount, _preferClaimedTokens);
> }
> ```

> Furthermore, reserved rate changes per fc is noted in the protocol's known risks exposed by design:https://info.juicebox.money/dev/learn/risks#undistributed-reserved-rate-risk.

jack-the-pug (judge) decreased severity to Medium and commented:

> I find this issue to be a valid Medium issue as it introduced an unexpected behavior that can cause a leak of value in certain circumstances.

[M-15] Locked splits can be updated

Submitted by berndartmueller

The check if the newly provided project splits contain the currently locked splits does not check the `JBSplit` struct properties `preferClaimed` and `preferAddToBalance`.

According to the docs in `JBSplit.sol`, *"...if the split should be unchangeable until the specified time, with the exception of extending the locked period."*, locked sets are unchangeable.

However, locked sets with either `preferClaimed` or `preferAddToBalance` set to true can have their bool values overwritten by supplying the same split just with different bool values.

Proof of Concept

JBSplitsStore.sol#L213-L220

The check for sameness does not check the equality of the struct properties `preferClaimed` and `preferAddToBalance`.

Please see warden's original report for full PoC and Mitigation details.

Recommended Mitigation Steps

Add two additional sameness checks for `preferClaimed` and `preferAddToBalance`:

mejango (Juicebox) confirmed

mejango (Juicebox) resolved:

> PR with fix: PR #1

berndartmueller (warden) reviewed mitigation:

> Two additional sameness checks for the split properties `preferClaimed` and `preferAddToBalance` have been added.

Low Risk and Non-Critical Issues

For this contest, 60 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by IIIIIII received the top score from the judge.

The following wardens also submitted reports: Lambda, _141345_, zzzitron, berndartmueller, Meera, GimelSec, Picodes, horsefacts, 0xNazgul, 0x1f8b, simon135, cccz, sahar, robee, jonatascm, joestakey, hubble, Funen, codexploder, 0xDjango, Sm4rty, Hawkeye, delfin454000, Ch_301, asutorufos, hake, Waze, TomJ, TerrierLover, svskaushik, samruna, sach1r0, Rohan16, ReyAdmirado, rbserver, pashov, oyc_109, MiloTruck, m_Rassska, Kaiziron, JC, durianSausage, defsec, Chom, Chandr, Bnke0x0, aysha, 0xNineDec, 0xf15ers, 0xdanial, 0x29A, 0v3rf10w, _Adam, rajatbeladiya, Noah3o6, jayfromthe13th, fatherOfBlocks, djxploit, *and* brgltd.

Low Risk Issues

	Issue	Instances
L-01	Weight of one being used as zero not documented	1
L-02	Calls may run out of gas until arrays are reduced in size	2
L-03	Dust amounts not compensated, even if not using price oracle	1
L-04	Splits can't be locked once the timestamp passes `type(uint48).max`	1
L-05	Unsafe use of `transfer()` / `transferFrom()` with `IERC20`	2

Total: 7 instances over 5 issues

[L-01] Weight of one being used as zero not documented

The comments and code below say that a weight of one is being used as a weight of zero. If a project is mature, or eventually becomes mature, a weight of one may in fact be a useful weighting, and the project owners will become very confused when they are unable to receive funds with this weighting.

There is 1 instance of this issue:

```
File: contracts/JBFundingCycleStore.sol   #1

467         // A weight of 1 is treated as a weight of 0.
468         // This is to allow a weight of 0 (default) to repres
469         _weight = _weight > 0
470           ? (_weight == 1 ? 0 : _weight)
471:          : _deriveWeightFrom(_baseFundingCycle, _start);
```

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBFundingCycleStore.sol#L467-L471

[L-02] Calls may run out of gas until arrays are reduced in size

The examples below are of functions that may revert due to the size of the data they're processing, but no funds are at risk because the arrays can be changed.

There are 2 instances of this issue. (For in-depth details on this and all further gas optimizations with multiple instances, see the warden's full report.)

[L-03] Dust amounts not compensated, even if not using price oracle

If there's a fixed weighting between what the user provides, and what is minted for them, there should be code that tracks partial token amounts, so that later payments are compensated for their prior partial amounts.

There is 1 instance of this issue:

```
File: contracts/JBSingleTokenPaymentTerminalStore.sol   #1

385       uint256 _weightRatio = _amount.currency == _baseWeightC
386         ? 10**_decimals
387         : prices.priceFor(_amount.currency, _baseWeightCurrer
388
389       // Find the number of tokens to mint, as a fixed point
390:      tokenCount = PRBMath.mulDiv(_amount.value, _weight, _we
```

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBSingleTokenPaymentTerminalStore.sol#L385-L390

[L-04] Splits can't be locked once the timestamp passes `type(uint48).max`

This behavior isn't documented anywhere, and a project will be confused by this behavior when that time comes (the original developers will be unable to explain it because they'll be dead).

There is 1 instance of this issue:

```
File: contracts/JBSplitsStore.sol   #1

261:          if (_splits[_i].lockedUntil > type(uint48).max) rev
```

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBSplitsStore.sol#L261

[L-05] Unsafe use of `transfer()` / `transferFrom()` with `IERC20`

Some tokens do not implement the ERC20 standard properly but are still accepted by most code that accepts ERC20 tokens. For example Tether (USDT)'s `transfer()`

and `transferFrom()` functions on L1 do not return booleans as the specification requires, and instead have no return value. When these sorts of tokens are cast to `IERC20`, their function signatures do not match and therefore the calls made, revert (see this link for a test case). Use OpenZeppelin's `SafeERC20`'s `safeTransfer()` / `safeTransferFrom()` instead.

There are 2 instances of this issue.

Non-Critical Issues

	Issue	Instances
N-01	Confusing variable names	1
N-02	Return values of `approve()` not checked	1
N-03	Adding a `return` statement when the function defines a named return variable, is redundant	4
N-04	Non-assembly method available	1
N-05	`constant`s should be defined rather than using magic numbers	37
N-06	Use a more recent version of solidity	1
N-07	Use a more recent version of solidity	3
N-08	Use scientific notation (e.g. `1e18`) rather than exponentiation (e.g. `10**18`)	1
N-09	Constant redefined elsewhere	11
N-10	Inconsistent spacing in comments	1
N-11	Lines are too long	49
N-12	Typos	17
N-13	File is missing NatSpec	29
N-14	NatSpec is incomplete	5
N-15	Event is missing `indexed` fields	34

	Issue	Instances
N-16	Not using the named return variables anywhere in the function is confusing	6

Total: 201 instances over 16 issues

[N-01] Confusing variable names

It was well into my review before I realized that 'configuration' means the timestamp at which the configuration is set, not the actual configuration details. It would be helpful to people reading the code to name it something like `configTimestamp` in all places. Below is one example of many.

There is 1 instance of this issue:

```
File: contracts/JBFundingCycleStore.sol   #1

332:     uint256 _configuration = block.timestamp;
```

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/blob/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBFundingCycleStore.sol#L332

[N-02] Return values of `approve()` not checked

Not all `IERC20` implementations `revert()` when there's a failure in `approve()`. The function signature has a `boolean` return value and they indicate errors that way instead. By not checking the return value, operations that should have marked as failed, may potentially go through without actually approving anything.

There is 1 instance of this issue:

```
File: contracts/JBERC20PaymentTerminal.sol   #1

99:       IERC20(token).approve(_to, _amount);
```

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/tree/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBERC20PaymentTerminal.sol#L99

[N-03] Adding a `return` statement when the function defines a named return variable, is redundant

There are 4 instances of this issue.

[N-04] Non-assembly method available

```
assembly{ id := chainid() } => uint256 id = block.chainid, assembly {
size := extcodesize() } => uint256 size = address().code.length
```

There is 1 instance of this issue:

```
File: contracts/JBFundingCycleStore.sol   #1

320:           _size := extcodesize(_ballot) // No contract at th
```

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/tree/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBFundingCycleStore.sol#L320

[N-05] `constant`s should be defined rather than using magic numbers

Even assembly can benefit from using readable constants instead of hex/numeric literals.

There are 37 instances of this issue.

[N-06] Use a more recent version of solidity

Use a solidity version of at least 0.8.12 to get `string.concat()` to be used instead of `abi.encodePacked(<str>,<str>)`.

There is 1 instance of this issue:

```
File: contracts/abstract/JBPayoutRedemptionPaymentTerminal.sol

2:    pragma solidity 0.8.6;
```

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/tree/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/abstract/JBPayoutRedemptionPaymentTerminal.sol#L2

[N-07] Use a more recent version of solidity

Use a solidity version of at least 0.8.13 to get the ability to use `using for` with a list of free functions.

There are 3 instances of this issue.

[N-08] Use scientific notation (e.g. `1e18`) rather than exponentiation (e.g. `10**18`)

There is 1 instance of this issue:

```
File: contracts/JBSingleTokenPaymentTerminalStore.sol   #1

868:         : PRBMath.mulDiv(_ethOverflow, 10**18, prices.priceF
```

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/tree/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBSingleTokenPaymentTerminalStore.sol#L868

[N-09] Constant redefined elsewhere

Consider defining in only one contract so that values cannot become out of sync when only one location is updated. A cheap way to store constants in a single location is to create an `internal constant` in a `library`. If the variable is a local cache of another contract's value, consider making the cache variable internal or

private, which will require external users to query the contract with the source of truth, so that callers don't get out of sync.

There are 11 instances of this issue.

[N-10] Inconsistent spacing in comments

Some lines use `// x` and some use `//x`. The instances below point out the usages that don't follow the majority, within each file.

There is 1 instance of this issue:

```
File: contracts/JBController.sol   #1

912:       //Transfer between all splits.
```

https://github.com/jbx-protocol/juice-contracts-v2-code4rena/tree/828bf2f3e719873daa08081cfa0d0a6deaa5ace5/contracts/JBController.sol#L912

[N-11] Lines are too long

Usually lines in source code are limited to 80 characters. Today's screens are much larger so it's reasonable to stretch this in some cases. Since the files will most likely reside in GitHub, and GitHub starts using a scroll bar in all cases when the length is over 164 characters, the lines below should be split when they reach that length.

There are 49 instances of this issue.

[N-12] Typos

There are 17 instances of this issue.

[N-13] File is missing NatSpec

There are 29 instances of this issue.

[N-14] NatSpec is incomplete

There are 5 instances of this issue.

[N-15] Event is missing `indexed` fields

Index event fields make the field more quickly accessible to off-chain tools that parse events. However, note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed per event (threefields). Each `event` should use three `indexed` fields if there are three or more fields, and gas usage is not particularly of concern for the events in question.

There are 34 instances of this issue.

[N-16] Not using the named return variables anywhere in the function is confusing

Consider changing the variable to be an unnamed one.

There are 6 instances of this issue.

Gas Optimizations

For this contest, 74 reports were submitted by wardens detailing gas optimizations. The report highlighted below by **0xA5DF** received the top score from the judge.

The following wardens also submitted reports: lllllll, horsefacts, joestakey, fatherOfBlocks, defsec, 0xKitsune, 0x1f8b, TomJ, Saintcode_, RedOneN, Meera, Limbooo, Lambda, jonatascm, c3phas, UnusualTurtle, Sm4rty, simon135, sashik_eth, sach1r0, robee, ReyAdmirado, rbserver, MiloTruck, m_Rassska, JohnSmith, durianSausage, cRat1st0s, brgltd, Bnke0x0, ajtra, 0xf15ers, _141345_, Waze, Tutturu, Tomio, rfa, oyc_109, Noah3o6, Metatron, Kaiziron, kaden, JC, jayfromthe13th, ignacio, djxploit, delfin454000, Ch_301, Aymen0909, 0xNazgul, 0v3rf10w, _Adam, Randyyy, mrpathfindr, hake, Funen, ElKu, asutorufos, apostle0x01, 0xdanial, 0x29A, Rohan16, rajatbeladiya, Picodes, mektigboy, kebabsec, Hawkeye, exd0tpy, codexploder, Chom, Cheeezzyyyy, 0xDjango, *and* 0x09GTO.

[G-01] Run checks first

Running checks before doing other operations can save gas in case the checks don't pass (since less operations were done before the revert).

Lines: [JBDirectory.sol#L270-L278]

Gas saved: Not measured by tests, can be a few dozen of thousands in case of revert (tested with a contract mocking the same behavior and 3 terminals)

```
-     // Delete the stored terminals for the project.
-     _terminalsOf[_projectId] = _terminals;
-
      // Make sure duplicates were not added.
+     // @audit run checks before assigning, to save gas in case
      if (_terminals.length > 1)
        for (uint256 _i; _i < _terminals.length; _i++)
          for (uint256 _j = _i + 1; _j < _terminals.length; _j++)
            if (_terminals[_i] == _terminals[_j]) revert DUPLICAT

+     // Delete the stored terminals for the project.
+     _terminalsOf[_projectId] = _terminals;
+
      emit SetTerminals(_projectId, _terminals, msg.sender);
```

[G-02] Store elements that are used multiple times

When the same array/mapping element is accessed more than once at the same block (without being modified) - it's cheaper to store the element as a var and access that var every time.

Gas saved: up to 2K units

Please see warden's **original report** *for full details.*

[G-03] Make loop increment unchecked

Overflowing loop index is virtually impossible, therefore it's cheaper to make the increment unchecked. It's also a bit cheaper to use ++i instead of i++.

Gas saved: up to 300 units

Lines:

- JBController.sol#L913
- JBController.sol#L1014
- JBDirectory.sol#L139
- JBDirectory.sol#L167
- JBDirectory.sol#L275-L276
- JBETHERC20SplitsPayer.sol#L466
- JBFundingCycleStore.sol#L724
- JBOperatorStore.sol#L85
- JBOperatorStore.sol#L138
- JBOperatorStore.sol#L171
- JBSingleTokenPaymentTerminalStore.sol#L862
- JBSplitsStore.sol#L204
- JBSplitsStore.sol#L211
- JBSplitsStore.sol#L229
- JBSplitsStore.sol#L304

```
diff --git a/contracts/JBController.sol b/contracts/JBControlle
index 26cd238..475a35a 100644
--- a/contracts/JBController.sol
+++ b/contracts/JBController.sol
@@ -910,7 +910,7 @@ contract JBController is IJBController, IJB
     JBSplit[] memory _splits = splitsStore.splitsOf(_projectId,

     //Transfer between all splits.
-    for (uint256 _i = 0; _i < _splits.length; _i++) {
+    for (uint256 _i = 0; _i < _splits.length; ) {
       // Get a reference to the split being iterated on.
       JBSplit memory _split = _splits[_i];

@@ -964,6 +964,9 @@ contract JBController is IJBController, IJB
         _tokenCount,
         msg.sender
```

```
        );
+       unchecked {
+         ++_i;
+       }
      }
    }

@@ -1011,7 +1014,7 @@ contract JBController is IJBController, IJ
      splitsStore.set(_projectId, _fundingCycle.configuration, _g

      // Set distribution limits if there are any.
-     for (uint256 _i; _i < _fundAccessConstraints.length; _i++)
+     for (uint256 _i; _i < _fundAccessConstraints.length; ) {
        JBFundAccessConstraints memory _constraints = _fundAccess

        // If distribution limit value is larger than 232 bits, r
@@ -1051,6 +1054,9 @@ contract JBController is IJBController, IJ
          _constraints,
          msg.sender
        );
+       unchecked {
+         ++_i;
+       }
      }

      return _fundingCycle.configuration;
diff --git a/contracts/JBDirectory.sol b/contracts/JBDirectory.s
index 865c719..442e704 100644
--- a/contracts/JBDirectory.sol
+++ b/contracts/JBDirectory.sol
@@ -137,9 +137,12 @@ contract JBDirectory is IJBDirectory, JBOpe

      IJBPaymentTerminal[] storage _terminalOf_projectId = _termi
      // Return the first terminal which accepts the specified to
-     for (uint256 _i; _i < _terminalOf_projectId.length; _i++) {
+     for (uint256 _i; _i < _terminalOf_projectId.length; ) {
        IJBPaymentTerminal _terminal = _terminalOf_projectId[_i];
        if (_terminal.acceptsToken(_token, _projectId)) return _t
+       unchecked {
+         ++_i;
+       }
      }

      // Not found.
@@ -165,8 +168,12 @@ contract JBDirectory is IJBDirectory, JBOpe
      override
```

```
     returns (bool)
   {
-    for (uint256 _i; _i < _terminalsOf[_projectId].length; _i++
+    for (uint256 _i; _i < _terminalsOf[_projectId].length; ) {
       if (_terminalsOf[_projectId][_i] == _terminal) return tru
+      unchecked {
+        ++_i;
+      }
+    }
     return false;
   }

@@ -272,9 +279,17 @@ contract JBDirectory is IJBDirectory, JBOpe
     // Make sure duplicates were not added.
     // @audit run checks before assigning, to save gas in case
     if (_terminals.length > 1)
-      for (uint256 _i; _i < _terminals.length; _i++)
-        for (uint256 _j = _i + 1; _j < _terminals.length; _j++)
+      for (uint256 _i; _i < _terminals.length; ) {
+        for (uint256 _j = _i + 1; _j < _terminals.length; ) {
           if (_terminals[_i] == _terminals[_j]) revert DUPLICAT
+          unchecked {
+            ++_j;
+          }
+        }
+        unchecked {
+          ++_i;
+        }
+      }

     // Delete the stored terminals for the project.
     _terminalsOf[_projectId] = _terminals;
diff --git a/contracts/JBETHERC20SplitsPayer.sol b/contracts/JBE
index 97a6517..6c344bd 100644
--- a/contracts/JBETHERC20SplitsPayer.sol
+++ b/contracts/JBETHERC20SplitsPayer.sol
@@ -463,7 +463,7 @@ contract JBETHERC20SplitsPayer is IJBSplitsP
     leftoverAmount = _amount;

     // Settle between all splits.
-    for (uint256 i = 0; i < _splits.length; i++) {
+    for (uint256 i = 0; i < _splits.length; ) {
       // Get a reference to the split being iterated on.
       JBSplit memory _split = _splits[i];

@@ -544,6 +544,9 @@ contract JBETHERC20SplitsPayer is IJBSplitsP
```

```
        }

        emit DistributeToSplit(_split, _splitAmount, _defaultBene
+       unchecked {
+         ++i;
+       }
      }
    }
  }
diff --git a/contracts/JBFundingCycleStore.sol b/contracts/JBFur
index 13fe9e6..79d16c3 100644
--- a/contracts/JBFundingCycleStore.sol
+++ b/contracts/JBFundingCycleStore.sol
@@ -312,8 +312,7 @@ contract JBFundingCycleStore is IJBFundingCy
     if (_data.weight > type(uint88).max) revert INVALID_WEIGHT(

     // Ballot should be a valid contract, supporting the correc
-    if(_data.ballot != IJBFundingCycleBallot(address(0))) {
-
+    if (_data.ballot != IJBFundingCycleBallot(address(0))) {
       address _ballot = address(_data.ballot);
       uint32 _size;
       assembly {
@@ -721,7 +722,7 @@ contract JBFundingCycleStore is IJBFundingCy
     // Apply the base funding cycle's discount rate for each cy
     uint256 _discountMultiple = _startDistance / _baseFundingCy

-    for (uint256 i = 0; i < _discountMultiple; i++) {
+    for (uint256 i = 0; i < _discountMultiple; ) {
       // The number of times to apply the discount rate.
       // Base the new weight on the specified funding cycle's w
       weight = PRBMath.mulDiv(
@@ -731,6 +732,9 @@ contract JBFundingCycleStore is IJBFundingCy
       // The calculation doesn't need to continue if the weight
       if (weight == 0) break;
+      unchecked {
+        ++i;
+      }
     }
   }

diff --git a/contracts/JBOperatorStore.sol b/contracts/JBOperatc
index e1e0241..e67f97d 100644
--- a/contracts/JBOperatorStore.sol
+++ b/contracts/JBOperatorStore.sol
```

```
@@ -82,13 +82,16 @@ contract JBOperatorStore is IJBOperatorStore
     uint256 _domain,
     uint256[] calldata _permissionIndexes
   ) external view override returns (bool) {
-    for (uint256 _i = 0; _i < _permissionIndexes.length; _i++)
+    for (uint256 _i = 0; _i < _permissionIndexes.length; ) {
       uint256 _permissionIndex = _permissionIndexes[_i];

       if (_permissionIndex > 255) revert PERMISSION_INDEX_OUT_O

       if (((permissionsOf[_operator][_account][_domain] >> _per
         return false;
+      unchecked {
+        ++_i;
+      }
     }
     return true;
   }
@@ -132,7 +135,7 @@ contract JBOperatorStore is IJBOperatorStore
     @param _operatorData The data that specify the params for e
   */
   function setOperators(JBOperatorData[] calldata _operatorData
-    for (uint256 _i = 0; _i < _operatorData.length; _i++) {
+    for (uint256 _i = 0; _i < _operatorData.length; ) {
       // Pack the indexes into a uint256.
       uint256 _packed = _packedPermissions(_operatorData[_i].pe

@@ -146,6 +149,9 @@ contract JBOperatorStore is IJBOperatorStore
         _operatorData[_i].permissionIndexes,
         _packed
       );
+      unchecked {
+        ++_i;
+      }
     }
   }

@@ -162,13 +168,16 @@ contract JBOperatorStore is IJBOperatorSto
     @return packed The packed value.
   */
   function _packedPermissions(uint256[] calldata _indexes) priv
-    for (uint256 _i = 0; _i < _indexes.length; _i++) {
+    for (uint256 _i = 0; _i < _indexes.length; ) {
       uint256 _index = _indexes[_i];

       if (_index > 255) revert PERMISSION_INDEX_OUT_OF_BOUNDS()
```

```
        // Turn the bit at the index on.
        packed |= 1 << _index;
+       unchecked {
+         ++_i;
+       }
      }
    }
  }
diff --git a/contracts/JBSingleTokenPaymentTerminalStore.sol b/c
index 4fc5d46..21be5ff 100644
--- a/contracts/JBSingleTokenPaymentTerminalStore.sol
+++ b/contracts/JBSingleTokenPaymentTerminalStore.sol
@@ -859,8 +859,12 @@ contract JBSingleTokenPaymentTerminalStore
      uint256 _ethOverflow;

      // Add the current ETH overflow for each terminal.
-     for (uint256 _i = 0; _i < _terminals.length; _i++)
+     for (uint256 _i = 0; _i < _terminals.length; ) {
        _ethOverflow = _ethOverflow + _terminals[_i].currentEthOv
+       unchecked {
+         ++_i;
+       }
+     }

      // Convert the ETH overflow to the specified currency if ne
      uint256 _totalOverflow18Decimal = _currency == JBCurrencies
diff --git a/contracts/JBSplitsStore.sol b/contracts/JBSplitsSto
index be0d17b..2c9d371 100644
--- a/contracts/JBSplitsStore.sol
+++ b/contracts/JBSplitsStore.sol
@@ -201,7 +201,7 @@ contract JBSplitsStore is IJBSplitsStore, JB
      JBSplit[] memory _currentSplits = _getStructsFor(_projectId

      // Check to see if all locked splits are included.
-     for (uint256 _i = 0; _i < _currentSplits.length; _i++) {
+     for (uint256 _i = 0; _i < _currentSplits.length; ) {
        JBSplit memory _currentSplit_i = _currentSplits[_i];
        // If not locked, continue.
        if (block.timestamp >= _currentSplit_i.lockedUntil) conti
@@ -209,7 +209,7 @@ contract JBSplitsStore is IJBSplitsStore, JB
        // Keep a reference to whether or not the locked split be
        bool _includesLocked = false;

-       for (uint256 _j = 0; _j < _splits.length; _j++) {
+       for (uint256 _j = 0; _j < _splits.length; ) {
```

```
          // Check for sameness.
          JBSplit memory _split_j = _splits[_j];
          if (
@@ -220,15 +220,22 @@ contract JBSplitsStore is IJBSplitsStore,
            // Allow lock extention.
            _split_j.lockedUntil >= _currentSplit_i.lockedUntil
          ) _includesLocked = true;
+         unchecked {
+           ++_j;
+         }
        }

        if (!_includesLocked) revert PREVIOUS_LOCKED_SPLITS_NOT_I
+
+       unchecked {
+         ++_i;
+       }
      }

      // Add up all the percents to make sure they cumulative are
      uint256 _percentTotal = 0;

-     for (uint256 _i = 0; _i < _splits.length; _i++) {
+     for (uint256 _i = 0; _i < _splits.length; ) {
        JBSplit memory _splits_i = _splits[_i];
        // The percent should be greater than 0.
        if (_splits_i.percent == 0) revert INVALID_SPLIT_PERCENT(
@@ -276,6 +283,9 @@ contract JBSplitsStore is IJBSplitsStore, JB
          delete _packedSplitParts2Of[_projectId][_domain][_group

        emit SetSplit(_projectId, _domain, _group, _splits_i, msg
+       unchecked {
+         ++_i;
+       }
      }

      // Set the new length of the splits.
@@ -304,7 +314,7 @@ contract JBSplitsStore is IJBSplitsStore, JB
      JBSplit[] memory _splits = new JBSplit[](_splitCount);

      // Loop through each split and unpack the values into struc
-     for (uint256 _i = 0; _i < _splitCount; _i++) {
+     for (uint256 _i = 0; _i < _splitCount; ) {
        // Get a reference to the fist packed data.
        uint256 _packedSplitPart1 = _packedSplitParts1Of[_project
```

```
@@ -335,6 +345,9 @@ contract JBSplitsStore is IJBSplitsStore, JB

       // Add the split to the value being returned.
       _splits[_i] = _split;
+      unchecked {
+        ++_i;
+      }
     }

     return _splits;
```

Gas diff:

```
┌─────────────────────────────────────────────────────────────┬──────────
| contracts/JBController.sol:JBController contract            |
╞═════════════════════════════════════════════════════════════╪══════════
| Deployment Cost                                             | Deploymen
├-------------------------------------------------------------+----------
-| 3979659                                                     | 20791
+| 3970050                                                     | 20743
├-------------------------------------------------------------+----------
| Function Name                                               | min
├-------------------------------------------------------------+----------
-| burnTokensOf                                                | 30462
+| burnTokensOf                                                | 30462
├-------------------------------------------------------------+----------
@@ -155,9 +155,9 @@ Test result: ok. 1 passed; 0 failed; finishe
├-------------------------------------------------------------+----------
-| launchProjectFor                                            | 287838
+| launchProjectFor                                            | 287838
├-------------------------------------------------------------+----------
-| mintTokensOf                                                | 20132
+| mintTokensOf                                                | 20058
├-------------------------------------------------------------+----------
@@ -172,21 +172,21 @@ Test result: ok. 1 passed; 0 failed; finis
╞═════════════════════════════════════════════════════════════╪══════════
| Deployment Cost                                             | Deployment
├-------------------------------------------------------------+----------
-| 1247680                                                     | 6698
+| 1232666                                                     | 6623
├-------------------------------------------------------------+----------
| Function Name                                               | min
├-------------------------------------------------------------+----------
-| isTerminalOf                                                | 633
```

```
+| isTerminalOf                                                  ┆ 633
 ├──────────────────────────────────────────────────────────────────────
-| primaryTerminalOf                                            ┆ 2295
+| primaryTerminalOf                                            ┆ 2295
 ├──────────────────────────────────────────────────────────────────────
-| setTerminalsOf                                               ┆ 54817
+| setTerminalsOf                                               ┆ 54817
 ├──────────────────────────────────────────────────────────────────────
 | terminalsOf                                                  ┆ 1389
 └──────────────────────────────────────────────────────────────┴───────
@@ -226,7 +226,7 @@ Test result: ok. 1 passed; 0 failed; finishe
 ├──────────────────────────────────────────────────────────────────────
 | Deployment Cost
 ├──────────────────────────────────────────────────────────────────────
-| 2292746
+| 2283531
 ├──────────────────────────────────────────────────────────────────────
 | Function Name
 ├──────────────────────────────────────────────────────────────────────
@@ -249,13 +249,13 @@ Test result: ok. 1 passed; 0 failed; finis
 ├──────────────────────────────────────────────────────────────────────
 | distributePayoutsOf
 ├──────────────────────────────────────────────────────────────────────
-| pay
+| pay
 ├──────────────────────────────────────────────────────────────────────
-| redeemTokensOf
+| redeemTokensOf
 ├──────────────────────────────────────────────────────────────────────
 | supportsInterface
 ├──────────────────────────────────────────────────────────────────────
-@@ -268,7 +268,7 @@ Test result: ok. 1 passed; 0 failed; finishe
+@@ -268,7 +268,7 @@ Test result: ok. 1 passed; 0 failed; finishe
 ├──────────────────────────────────────────────────────────────────────
 | Deployment Cost
 ├──────────────────────────────────────────────────────────────────────
-| 1055746
+| 1048339
 ├──────────────────────────────────────────────────────────────────────
 | Function Name
 ├──────────────────────────────────────────────────────────────────────
@@ -330,7 +330,7 @@ Test result: ok. 1 passed; 0 failed; finishe
 ├──────────────────────────────────────────────────────────────────────
 | Deployment Cost
 ├──────────────────────────────────────────────────────────────────────
-| 2551081
+
```

```
+| 2543674
  ├-------------------------------------------------------------------
  | Function Name
  ├-------------------------------------------------------------------
@@ -338,13 +338,13 @@ Test result: ok. 1 passed; 0 failed; finis
  ├-------------------------------------------------------------------
-| currentTotalOverflowOf
+| currentTotalOverflowOf
  ├-------------------------------------------------------------------
-| recordRedemptionFor
+| recordRedemptionFor
  └-------------------------------------------------------------------
@@ -353,15 +353,15 @@ Test result: ok. 1 passed; 0 failed; finis
  ╞═══════════════════════════════════════════════════════════╪═══════
  | Deployment Cost                                           ¦ Deploym
  ├-----------------------------------------------------------+-------
-| 751811                                                    ¦ 4116
+| 736791                                                    ¦ 4041
  ├-----------------------------------------------------------+-------
  | Function Name                                             ¦ min
  ├-----------------------------------------------------------+-------
-| set                                                       ¦ 2998
+| set                                                       ¦ 2998
  ├-----------------------------------------------------------+-------
-| splitsOf                                                  ¦ 2941
+| splitsOf                                                  ¦ 2941
```

drgorillamd (Juicebox) commented:

> Really nice PoC/gas analysis, thank you.

Mitigation Review

Mitigation review by **berndartmueller**

Reviewed pull requests: PR#1 (`b8e2472ce750ad084440c8db6090143807e79893`), **PR#4**

Mitigation Overview

The following is a high-level overview of the core changes introduced as the mitigation, arranged per the report findings.

- [H-01] Resolved. Appropriate validations to prevent price staleness, round incompleteness and a negative price is put in place now.
- [H-02] Resolved. Changing an already set project token is not possible anymore.
- [M-01] Acknowledged.
- [M-02] Resolved. `mustStartAtOrAfter` and the start date of an upcoming funding cycle are now validated to fit in `uint56`.
- [M-03] Resolved. OpenZeppelins' `SafeERC20` library is now used to ensure consistent handling of ERC20 token transfers.
- [M-04] Acknowledged.
- [M-05] Acknowledged.
- [M-06] Resolved. The delta of the token balance before and after a transfer is used instead of the amount stated to handle fee-on-transfer tokens appropriately.
- [M-07] Acknowledged.
- [M-08] Acknowledged.
- [M-09] Acknowledged.
- [M-10] Acknowledged.
- [M-11] Resolved. An additional check has been added to prevent adding a price feed for the inverse pair.
- [M-12] Resolved. Once a token is set for a project, it can not be changed anymore.
- [M-13] Acknowledged.
- [M-14] Acknowledged.
- [M-15] Resolved. Two additional sameness checks for the split properties `preferClaimed` and `preferAddToBalance` have been added.

Medium Risk Findings (1)

[M.M-01] Migrating from V2 to V3 will cause issues with funding cycle metadata

Context: [libraries/JBFundingCycleMetadataResolver.sol#L150-L157]

Status: Acknowledged. Juicebox projects have to instantiate new V3 funding cycles if they wish to migrate from V2 to V3.

Description: In V3, the funding cycle metadata bitmask changes due to removing the parameter `allowChangeToken` and adding the new parameter `preferClaimedTokenOverride`. However, projects migrating from V2 to V3 with certain funding cycle metadata bits set will experience a possibly different funding cycle configuration than anticipated.

The following table shows a comparison of the occupied bits of affected funding cycle metadata parameters in V2 and V3:

Metadata Param	Bit Previously	Bit Now
allowChangeToken	77	-
allowTerminalMigration	78	77
allowControllerMigration	79	78
holdFees	80	79
preferClaimedTokenOverride	-	80

```
function packFundingCycleMetadata(JBFundingCycleMetadata memory
    internal
    pure
    returns (uint256 packed)
{
  [..]

  // allow terminal migration in bit 77.
  if (_metadata.allowTerminalMigration) packed |= 1 &lt;&lt; 77;
  // allow controller migration in bit 78.
  if (_metadata.allowControllerMigration) packed |= 1 &lt;&lt; 7
  // hold fees in bit 79.
  if (_metadata.holdFees) packed |= 1 &lt;&lt; 79;
  // prefer claimed token override in bit 80.
  if (_metadata.preferClaimedTokenOverride) packed |= 1 &lt;&lt;
```

```
        [..]
    }
```

Recommendation

Consider not changing the existing bits and their representation. Instead, add new metadata parameters to available most significant bits.

For example, consider storing the newly added `_metadata.metadata` in bits **248-256** (instead of using bits 244-252) and use one of the 4 available bits **244-247** for `_metadata.preferClaimedTokenOverride`.

Low Risk Findings (1)

[M.L-01] `defaultBeneficiary` is not used consistently

Context: JBETHERC20ProjectPayer.sol, JBETHERC20SplitsPayer.sol

Status: Resolved in PR#5

Description: When deploying a new instance of a `JBETHERC20ProjectPayer` or `JBETHERC20SplitsPayer` contract, the deployer can provide a default beneficiary address `defaultBeneficiary`. If the beneficiary is `address(0)`, this default beneficiary address will receive the project's tokens when the project payer receives payments.

However, `defaultBeneficiary` is only used within the `receive()` function. In the `pay(..)` function, the beneficiary, if set to `address(0)`, will immediately default to `tx.origin`.

Recommendation

Consider using `defaultBeneficiary` consistently and if `defaultBeneficiary` is set to `address(0)`, only then default to `tx.origin`.

Non-Critical Findings (1)

[M.N-01] `msg.sender` address is not checked if it is a feeless address

Context: abstract/JBPayoutRedemptionPaymentTerminal.sol

Status: Acknowledged. Feeless sender for distributions is removed in V3. Comment adjusted in PR#6

Description: According to the comment on L897:

> *// If the fee is zero or if the fee is* **being used by an address that doesn't incur fees**, *set the discount to 100% for convenience.*

If the caller is an address that doesn't incur fees, the discount should be set to 100%. However, in the latest version, `msg.sender` is no longer checked if it is considered a feeless address.

See diff

```
// If the fee is zero or if the fee is being used by an address
- uint256 _feeDiscount = fee == 0 || isFeelessAddress[msg.sender
+ uint256 _feeDiscount = fee == 0
  ? JBConstants.MAX_FEE_DISCOUNT
  : _currentFeeDiscount(_projectId);
```

Recommendation

Consider reverting the change and check if `msg.sender` is a feeless address.

Disclosures

C4 is an open organization governed by participants in the community.

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Contest submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct

formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top