Juicebox Buyback Delegate Findings & Analysis Report

2023-07-26

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 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 outlined in this document, C4 conducted an analysis of the Juicebox Buyback Delegate smart contract system written in Solidity. The audit took place between May 18—May 22, 2023.

Wardens

85 Wardens contributed reports to the Juicebox Buyback Delegate:

1. 0x4non
2. 0xHati
3. 0xMosh
4. 0xRobocop
5. 0xSmartContract
6. 0xStalin
7. 0xWaitress
8. 0xhacksmithh
9. 0xnacho
10. 0xnev
11. 0xprinc
12. ABA
13. Arabadzhiev
14. Arz
15. BLACK-PANDA-REACH (Mis4nthr0pic, escrow, Muhab, deliriusz, savi0ur, lopotras, Naubit, santipu_, devscrooge, ret2basic, and Draco000)
16. Deekshith99
17. Dimagu (marcKn and dmtrbch)
18. HHK
19. Haipls
20. JCN
21. K42
22. KKat7531

23. Kose
24. LosPollosHermanos (scaraven, LemonKurd, and jc1)
25. Madalad
26. MohammedRizwan
27. QiuhaoLi
28. RaymondFam
29. Rickard
30. Rolezn
31. SAAJ
32. Sathish9098
33. Shubham
34. SmartGooofy
35. SpicyMeatball
36. T1MOH
37. Tripathi
38. Udsen
39. V1235816
40. adriro
41. arpit
42. ast3ros
43. aviggiano
44. ayden
45. bigtone
46. codeVolcan
47. d3e4
48. dwward3n
49. fatherOfBlocks
50. favelanky
51. hunter_w3b

52. jovemjeune
53. ktg
54. kutugu
55. lfzkoala
56. lukris02
57. matrix_0wl
58. max10afternoon
59. minhquanym
60. ni8mare
61. niser93
62. parsely
63. pfapostol
64. pxng0lin
65. radev_sw
66. ravikiranweb3
67. rbserver
68. sces60107
69. souilos
70. tnevler
71. turvy_fuzz
72. yellowBirdy

This audit was judged by LSDan.

Final report assembled by itsmetechjay.

Summary

The C4 analysis yielded an aggregated total of 3 unique vulnerabilities. Of these vulnerabilities, 0 received a risk rating in the category of HIGH severity and 3 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 57 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 11 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 Buyback Delegate repository, and is composed of 1 smart contract written in the Solidity programming language and includes 160 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities based on 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

For more information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website, specifically our section on Severity Categorization.

Medium Risk Findings (3)

[M-01] Delegate architecture forces users to set zero slippage

Submitted by adriro*, also found by* rbserver*,* 0xnacho*,* SpicyMeatball*,* max10afternoon*,* HHK*, and* 0xRobocop

The design of the delegate forces users to set a zero value for the `_minReturnedTokens` parameter when calling `pay()` in the terminal.

Technical details

In order to implement the swap functionality, the JBXBuybackDelegate needs to signal the terminal to not mint any tokens when the swap path is token. This is done by setting the `weight` variable to zero:

https://github.com/code-423n4/2023-05-juicebox/blob/main/juice-buyback/contracts/JBXBuybackDelegate.sol#L144-L171

```
144:     function payParams(JBPayParamsData calldata _data)
145:         external
146:         override
147:         returns (uint256 weight, string memory memo, JBPayI
148:     {
149:         // Find the total number of tokens to mint, as a fi
150:         uint256 _tokenCount = PRBMath.mulDiv(_data.amount.v
151:
152:         // Unpack the quote from the pool, given by the frc
153:         (,, uint256 _quote, uint256 _slippage) = abi.decode
154:
155:         // If the amount swapped is bigger than the lowest
156:         if (_tokenCount < _quote - (_quote * _slippage / SI
157:             // Pass the quote and reserve rate via a mutex
158:             mintedAmount = _tokenCount;
159:             reservedRate = _data.reservedRate;
160:
161:             // Return this delegate as the one to use, and
162:             delegateAllocations = new JBPayDelegateAllocati
163:             delegateAllocations[0] =
164:                 JBPayDelegateAllocation({delegate: IJBPayDe
165:
166:             return (0, _data.memo, delegateAllocations);
167:         }
168:
169:         // If minting, do not use this as delegate
170:         return (_data.weight, _data.memo, new JBPayDelegate
171:     }
```

This can be seen in line 166 in the previous snippet of code, where the function returns zero as the `weight` return value when going the swap path. This `weight` is then used to calculate the `tokenCount` in the JBSingleTokenPaymentTerminalStore3_1 contract:

https://github.com/jbx-protocol/juice-contracts-v3/blob/main/contracts/JBSingleTokenPaymentTerminalStore3_1.sol#L427

```
...
tokenCount = PRBMath.mulDiv(_amount.value, _weight, _weightRatio
...
```

This `tokenCount` variable is finally returned to the JBPayoutRedemptionPaymentTerminal3_1 contract, which uses it to calculate the amount of tokens to mint and validate the tokens assigned to the beneficiary are above the minimum:

https://github.com/jbx-protocol/juice-contracts-v3/blob/main/contracts/abstract/JBPayoutRedemptionPaymentTerminal3_1.sol#L1470-L1493

```
...

(_fundingCycle, _tokenCount, _delegateAllocations, _memo) = stor
  _payer,
  _bundledAmount,
  _projectId,
  baseWeightCurrency,
  _beneficiary,
  _memo,
  _metadata
);

// Mint the tokens if needed.
if (_tokenCount > 0)
  // Set token count to be the number of tokens minted for the b
  beneficiaryTokenCount = IJBController(directory.controllerOf(_
    _projectId,
    _tokenCount,
    _beneficiary,
```

```
        '',
        _preferClaimedTokens,
        true
      );

    // The token count for the beneficiary must be greater than or e
    if (beneficiaryTokenCount < _minReturnedTokens) revert INADEQUAT

    ...
```

This means that if the swap path is chosen, `weight` is going to be zero, which sets `_tokenCount` to zero, and implies that `beneficiaryTokenCount` is going to be zero as well. Setting `_minReturnedTokens` to any value different than zero will make the transaction revert.

However, the “buyback” path is not the only alternative. The delegate could also take the “vanilla” path in which it simply bypasses the delegate and tokens are minted in the terminal (line 170 in `payParams()`). This creates a conflict between both paths that forces the user to set `_minReturnedTokens` to zero, because the taken path is resolved at transaction time. Setting `_minReturnedTokens` to any value different than zero will make the “buyback” path revert, while setting `_minReturnedTokens` to zero nullifies the slippage check when the “vanilla” path is taken.

Impact

Medium. The current design forces the user to set `_minReturnedTokens` to zero and disables any type of check over the amount of minted tokens.

Recommendation

It is difficult to give a recommendation that doesn’t involve significant changes to the code, as the current design relies on returning a zero weight when choosing the “buyback” path, which forces the condition in `_minReturnedTokens` in order to prevent the transaction revert. Nevertheless, the situation should be revised, as setting a zero value for `_minReturnedTokens` can have a significant impact on users engaging with the protocol.

drgorillamd (Juicebox) disagreed with severity via duplicate issue #36 and commented:

> 2 things:
>
> - `_minReturnedToken` is not really used currently in the protocol, as it's rather for future-proof (for the day where we have terminals with various currencies, where a slippage might appear) - screenshot of the latest pay(..) for instance: 
> - In this setting, the delegate doesn't receive this arg (hence the use of the "free" metadata arg, to pass the quote/min received)

> The scenario of "the transaction is pending then current fc rolls over and the beneficiary receive less token" looks a bit super edge case (especially for a Medium risk)...

> Agreed it should be more documented though, as it might be confusing right now.

LSDan (judge) commented via issue #36:

> I'm going to leave this in place as a Medium. The code functions in unexpected ways based on external factors and it's actual function is very different than a user could reasonably expect. If the variable should always be zero, consider removing it. I don't agree with the assertion that including dead arguments which may be used in the future is future-proof.

> Also, see here with regard to protecting users from themselves.

[M-02] Low Liquidity in Uniswap V3 Pool Can Lead to ETH Lockup in `JBXBuybackDelegate` Contract

Submitted by minhquanym, *also found by* BLACK-PANDA-REACH, rbserver, Udsen, adriro, adriro, Udsen, sces60107, Madalad, max10afternoon, 0xStalin, *and* T1MOH

The `JBXBuybackDelegate` contract employs Uniswap V3 to perform ETH-to-project token swaps. When the terminal invokes the `JBXBuybackDelegate.didPay()` function, it provides the amount of ETH to be swapped for project tokens. The swap operation sets `sqrtPriceLimitX96` to the lowest possible price, and the slippage is checked at the callback.

However, if the Uniswap V3 pool lacks sufficient liquidity or being manipulated before the transaction is executed, the swap will halt once the pool's price reaches the `sqrtPriceLimitX96` value. Consequently, not all the ETH sent to the contract will be utilized, resulting in the remaining ETH becoming permanently locked within the contract.

Proof of Concept

The `_swap()` function interacts with the Uniswap V3 pool. It sets `sqrtPriceLimitX96` to the minimum or maximum feasible value to ensure that the swap attempts to utilize all available liquidity in the pool.

```
try pool.swap({
    recipient: address(this),
    zeroForOne: !_projectTokenIsZero,
    amountSpecified: int256(_data.amount.value),
    sqrtPriceLimitX96: _projectTokenIsZero ? TickMath.MAX_SQRT_P
    data: abi.encode(_minimumReceivedFromSwap)
}) returns (int256 amount0, int256 amount1) {
    // Swap succeeded, take note of the amount of projectToken r
    _amountReceived = uint256(-(_projectTokenIsZero ? amount0 :
} catch {
    // implies _amountReceived = 0 -> will later mint when back
    return _amountReceived;
}
```

In the Uniswap V3 pool, [this check] stops the loop if the price limit is reached or the entire input has been used. If the pool does not have enough liquidity, it will still do the swap until the price reaches the minimum/maximum price.

```
// continue swapping as long as we haven't used the entire input
while (state.amountSpecifiedRemaining != 0 && state.sqrtPriceX96
    StepComputations memory step;

    step.sqrtPriceStartX96 = state.sqrtPriceX96;

    (step.tickNext, step.initialized) = tickBitmap.nextInitializ
        state.tick,
        tickSpacing,
        zeroForOne
```

```
        );
```

Finally, the `uniswapV3SwapCallback()` function uses the input from the pool callback to wrap ETH and transfer WETH to the pool. So, if `_amountToSend < msg.value`, the unused ETH is locked in the contract.

```
    function uniswapV3SwapCallback(int256 amount0Delta, int256 amour
        // Check if this is really a callback
        if (msg.sender != address(pool)) revert JuiceBuyback_Unauthc

        // Unpack the data
        (uint256 _minimumAmountReceived) = abi.decode(data, (uint256

        // Assign 0 and 1 accordingly
        uint256 _amountReceived = uint256(-(_projectTokenIsZero ? am
        uint256 _amountToSend = uint256(_projectTokenIsZero ? amount

        // Revert if slippage is too high
        if (_amountReceived < _minimumAmountReceived) revert JuiceBu

        // Wrap and transfer the weth to the pool
        weth.deposit{value: _amountToSend}();
        weth.transfer(address(pool), _amountToSend);
    }
```

Recommended Mitigation Steps

Consider returning the amount of unused ETH to the beneficiary.

LSDan (judge) decreased severity to Medium

drgorillamd (Juicebox) confirmed

[M-03] Funding cycles that use `JBXBuybackDelegate` as a redeem data source (or any derived class) will slash all redeemable tokens

Submitted by ABA, also found by ktg and RaymondFam

For a funding cycle that is set to use the data source for redeem and the data source is `JBXBuybackDelegate`, any and all redeemed tokens is 0 because of the returned 0 values from the empty `redeemParams` implementation. This means the beneficiary would wrongly receive 0 tokens instead of an intended amount.

While `JBXBuybackDelegate` was not designed to be used for redeem also, per protocol logic, this function should have been a pass-through (if no redeem such functionality was desired) because a redeem logic is by default used if not overwritten by `redeemParams`, as per **the documentation** (further detailed below). Impact is further increased as the function is not marked as `virtual`, as such, no inheriting class can modify the faulty implementation if any such extension is desired.

Vulnerability details

Project logic dictates that a contract can become a treasury data source by adhering to `IJBFundingCycleDataSource`, such as `JBXBuybackDelegate` is and that there are two functions that must be implemented, `payParams(...)` and `redeemParams(...)`. Either one can be **left empty** if the intent is to only extend the treasury's pay functionality or redeem functionality.

But by being left empty, it is specifically required to pass the input variables, not a 0 default value

https://docs.juicebox.money/dev/build/treasury-extensions/data-source/

> Return the JBRedeemParamsData.reclaimAmount value if no altered functionality is desired. Return the JBRedeemParamsData.memo value if no altered functionality is desired. Return the zero address if no additional functionality is desired.

What should have been done:

```
// This is unused but needs to be included to fulfill IJBFundi
function redeemParams(JBRedeemParamsData calldata _data)
  external
  pure
  override
```

```
    returns (
      uint256 reclaimAmount,
      string memory memo,
      IJBRedemptionDelegate delegate
    )
  {
    // Return the default values.
    return (_data.reclaimAmount.value, _data.memo, IJBRedemptior
  }
```

What is implemented:

https://github.com/code-423n4/2023-05-juicebox/blob/main/juice-buyback/contracts/JBXBuybackDelegate.sol#L235-L239

```
    function redeemParams(JBRedeemParamsData calldata _data)
        external
        override
        returns (uint256 reclaimAmount, string memory memo, JBRe
    {}
```

While an overwritten `memo` with an empty default string is not such a large issue, and incidentally `delegateAllocations` is actually returned correctly as a zero address, the issue lies with the `reclaimAmount` amount which is not returned as 0.

Per documentation:

> `reclaimAmount` is the amount of tokens in the treasury that the terminal should send out to the redemption beneficiary as a result of burning the amount of project tokens tokens

and in case of redemptions, by default, the value is:

> a reclaim amount determined by the standard protocol bonding curve based on the redemption rate the project has configured into its current funding cycle which is provided to the data source function in JBRedeemParamsData.reclaimAmount

but in this case, it will be overwritten with 0 thus the redemption beneficiary will be deprived of funds.

`redeemParams` is also lacking the `virtual` keyword, as such, no inheriting class can modify the faulty implementation if any such extension is desired.

Recommended Mitigation Steps

Change the implementation of the function to respect protocol logic and add the `virtual` keyword, for example:

```
// This is unused but needs to be included to fulfill IJBFundi
function redeemParams(JBRedeemParamsData calldata _data)
  external
  pure
  override
  virtual
  returns (
    uint256 reclaimAmount,
    string memory memo,
    IJBRedemptionDelegate delegate
  )
{
  // Return the default values.
  return (_data.reclaimAmount.value, _data.memo, IJBRedemptior
}
```

drgorillamd (Juicebox) disagreed with severity and commented:

> The funding cycle has two delegate-related parameters, `useDelegateForPay` and `useDelegateForRedemption`. This is a pay delegate, to have a situation where the redemption becomes faulty would require the project owner to proactively submit a reconfiguration, to activate the use of the delegate on redemption. This is a user bug, not a protocol one.

> Should be documented to avoid confusion.

LSDan (judge) commented:

> I respect that this is user error, but I also think it's a very easy error to prevent. Medium stands.

Low Risk and Non-Critical Issues

For this audit, 54 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by **ABA** received the top score from the judge.

The following wardens also submitted reports: d3e4, 0xMosh, dwward3n, Udsen, Arabadzhiev, KKat7531, SmartGooofy, Shubham, 0x4non, rbserver, adriro, codeVolcan, Deekshith99, sces60107, favelanky, lukris02, BLACK-PANDA-REACH, Dimagu, parsely, yellowBirdy, QiuhaoLi, LosPollosHermanos, SAAJ, MohammedRizwan, minhquanym, tnevler, radev_sw, bigtone, jovemjeune, turvy_fuzz, Tripathi, ni8mare, ayden, RaymondFam, 0xSmartContract, Rickard, Kose, V1235816, 0xnev, pxng0lin, 0xprinc, Rolezn, fatherOfBlocks, 0xhacksmithh, Sathish9098, souilos, 0xWaitress, lfzkoala, 0xHati, matrix_0wl, kutugu, arpit, *and* ravikiranweb3 .

Low Risk Issues

Total: 1 instance over 1 issues

#	Issue	Instances
[L-01]	`JBXBuybackDelegate` deployer should decide if held fees should be refunded	1

[L-01] `JBXBuybackDelegate` deployer should decide if held fees should be refunded

When minting of tokens is done, via `JBXBuybackDelegate::_mint` , ETH is sent back to the terminal via a call to `addToBalanceOf` .

There are 2 versions of `addToBalanceOf` . One that accepts a flag `_shouldRefundHeldFees` indicating if held fees should be refunded based on the amount being added.

And the other, with one less argument, that is used by our protocol that sets it to false by default.

This option should not be hardcoded as it is, a contract deployer should determine if he opts in or not.

Instances (1)

https://github.com/code-423n4/2023-05-juicebox/blob/main/juice-buyback/contracts/JBXBuybackDelegate.sol#L347-L350

Recommendation

Add a constructor parameter that is then used with the `addToBalanceOf` when `_mint` is called.

Non-Critical Issues

Total: 15 instances over 7 issues

#	Issue	Instances
[N-01]	Use `JBTokenAmount::decimals` instead of hardcoding it in `payParams`	1
[N-02]	Memo is not passed in all cases	3
[N-03]	Missing, misleading, incorrect, or incomplete comments	7
[N-04]	Events missing key information	1
[N-05]	Empty methods should be thoroughly documented	1
[N-06]	Better and documentation naming for `_projectTokenIsZero`	1
[N-07]	Use named function calls	1

[N-01] Use `JBTokenAmount::decimals` instead of hardcoding it in `payParams`

In `payParams` the number of tokens to mint is determined by multiplying value with weight and dividing by *1e18*.

```
// Find the total number of tokens to mint, as a fixed p
uint256 _tokenCount = PRBMath.mulDiv(_data.amount.value,
```

As the token to be used by the project has 18 decimals this is not an issue for now. Any other project token to be added will require this modified in order to keep the premise that the total number of tokens to mint is a fixed point number with 18 decimals.

Out of the formula, `_data.weight` has 18 decimals:

https://github.com/jbx-protocol/juice-contracts-v3/blob/12d852f28d372dd44987586f8009c56b0fe247a9/contracts/JBSingleTokenPaymentTerminalStore3_1.sol#L351-L352

```
// The weight according to which new token supply is to be m
uint256 _weight;
```

So the 1e18 must be the decimal count of the `JBTokenAmount` in order to be fully safe.

Instances (1)

https://github.com/code-423n4/2023-05-juicebox/blob/main/juice-buyback/contracts/JBXBuybackDelegate.sol#L149-L150

Recommendation

Use the decimals propriety of `JBTokenAmount` instead of a hardcoded *1e18*:

```
uint256 _tokenCount = PRBMath.mulDiv(_data.amount.value,
```

[N-02] Memo is not passed in all cases

All controller operations of `mintTokensOf` and `burnTokensOf` or terminal operation of `addToBalanceOf` have a `_memo` parameter.

The information in the memo will be passed to an event. The event is sent regardless so adding the memo will make off-chain analytics clearer.

Instances (3)

https://github.com/code-423n4/2023-05-juicebox/blob/main/juice-buyback/contracts/JBXBuybackDelegate.sol#L297

https://github.com/code-423n4/2023-05-juicebox/blob/main/juice-buyback/contracts/JBXBuybackDelegate.sol#L319

https://github.com/code-423n4/2023-05-juicebox/blob/main/juice-buyback/contracts/JBXBuybackDelegate.sol#L349

Recommendation

Pass the memo information to the mentioned calls.

[N-03] Missing, misleading, incorrect, or incomplete comments

There are cases where the comments are either missing, incomplete or incorrect throughout the codebase.

Instances (7)

- `_swap` :
    - at line 246 *toke_beforeTransferTon* should be *token_beforeTransferToken* (code)
    - at line 251 *fc* should be properly described as *funding cycle* (code)
    - at line 301 the comments after `result:` are ambiguous `_amountReceived-reserved here, reservedToken in reserve`, consider making them more clear, example: `(_amountReceived - reserved) here, (reserved (token)) in reserve` (code)
    - at lines 250 and 252 *ie* should be *i.e.* (code)
- `_mint` : at line 329 there is a double *in in*, remove one *in* (code)

- `uniswapV3SwapCallback`:
    - at line 214 typo in `controle`, an extra *e*, remove the extra *e* (code)
    - at line 212 typo, “should happen” or rephrase to “happens” (code)

Recommendation

Resolve the mentioned issues.

[N-04] Events missing key information

Some events are missing key information when emitted.

Instances (1)

- in `_mint` the `JBXBuybackDelegate_Mint` event should contain the minted amount and beneficiary to whom it was minted (code)

Recommendation

Add the missing information.

[N-05] Empty methods should be thoroughly documented

The function `redeemParams` variable has no code implementation.

It is both lacking any NatSpec and any internal comments to indicate its purpose.

Instances (1)

https://github.com/code-423n4/2023-05-juicebox/blob/main/juice-buyback/contracts/JBXBuybackDelegate.sol#L235-L239

Recommendation

Properly document the method, why it is needed and why it is empty.

[N-06] Better naming and documentation for `_projectTokenIsZero`

A variable named `_projectTokenIsZero` is used to indicate if the project token address is less then the address terminal token by sort order.

It describes it simply as:

```
/**
 * @notice Address project token < address terminal token ?
 */
bool private immutable _projectTokenIsZero;
```

then it is used to identify which swap return amount is to be used.

```
_amountReceived = uint256(-(_projectTokenIsZero ? am
```

This is needed because Uniswap `IUniswapV3Pool::pool` returns the token amounts corresponding to the token sorted addresses order.

This is implicit because pools are created with the tokens as such (IUniswapV3Factory#PoolCreated)

The naming and description can be changed to better describe the need for this flag variable.

Instances (1)

https://github.com/code-423n4/2023-05-juicebox/blob/main/juice-buyback/contracts/JBXBuybackDelegate.sol#L60-L63

Recommendation

Add a more descriptive name and documentation.

Example:

```
/**
 * @notice sores if: address project token < address termina
 * @dev used to identify which pair token (0 or 1) is the pr
```

```
     */
    bool private immutable _projectTokenIsPairToken0;
```

[N-07] Use named function calls

Code base has an extensive use of named function calls, but it somehow missed one instance where this would be appropriate.

```
        jbxTerminal.addToBalanceOf{value: _data.amount.value}(
            _data.projectId, _data.amount.value, JBTokens.ETH, '
        );
```

([addToBalanceOf](#))

Instances (1)

https://github.com/code-423n4/2023-05-juicebox/blob/main/juice-buyback/contracts/JBXBuybackDelegate.sol#L348-L350

Recommendation

Use named function calls on function call, as such:

```
    jbxTerminal.addToBalanceOf{value: _data.amount.value}({
        _projectId: _data.projectId,
        _amount: _data.amount.value,
        _token: JBTokens.ETH,
        _memo: "",
        _metadata: new bytes(0)
    });
```

drgorillamd (Juicebox) confirmed

Gas Optimizations

For this audit, 11 reports were submitted by wardens detailing gas optimizations. The report highlighted below by **JCN** received the top score from the judge.

The following wardens also submitted reports: Arz, Dimagu, QiuhaoLi, niser93, pfapostol, Tripathi, 0x4non, hunter_w3b, Sathish9098, *and* K42 .

Gas Optimizations Summary

All of the optimizations were benchmarked via the protocol's tests, i.e. using the following config: `solc version 0.8.19` . Some optimizations are also explained via EVM gas costs and opcodes.

Below are the overall average gas savings for the following tested functions (with all the optimizations applied):

Function	Before	After	Avg Gas Savings
JBXBuybackDelegate.didPay	242106	240737	1369
JBXBuybackDelegate.payParams	8271	5393	2878
JBXBuybackDelegate.uniswapV3SwapCallback	20125	19772	353

Total gas saved across all listed functions: 4600

Notes:

- The Gas report output, after all optimizations have been applied, can be found at the end of the report.
- The final diff for the contract, with all the optimizations applied, can be found here.
- The `Avg` gas for `didPay` changes between tests and so the `Max` column (which remains the same) is used when benchmarking this function.
- Only runtime gas is highlighted above, as it will inevitably outweight deployment gas costs throughout the lifetime of the protocol.
- Substantial deployment gas savings are highlighted in their appropriate instances and are only provided as a reference.

[G-01] State variables can be packed to use fewer storage slots

The EVM works with 32 byte words. Variables less than 32 bytes can be declared next to each other in storage and this will pack the values together into a single 32 byte storage slot (if the values combined are <= 32 bytes). If the variables packed together are retrieved together in functions we will effectively save ~2000 gas with every subsequent SLOAD for that storage slot. This is due to us incurring a `Gwarmaccess (100 gas)` versus a `Gcoldsload (2100 gas)`.

Note: This optimization will save ~20_000 deployment gas since we are avoiding one `Gsset (20000 gas)` during deployment. However, only runtime gas is bencharked since runtime gas is paid multiple times over, while deployment gas is only paid once.

https://github.com/code-423n4/2023-05-juicebox/blob/main/juice-buyback/contracts/JBXBuybackDelegate.sol#L106-L113

Gas Savings for `JBXBuybackDelegate.payParams`, obtained via protocol's tests: Avg 2743 gas

	Med	Max	Avg	# calls
Before	9981	12476	8271	10
After	6771	7523	5528	10

Pack `mintedAmount` and `reservedRate` into one storage slot to save 1 SLOT (~2000 gas)

```
File: juice-buyback/contracts/JBXBuybackDelegate.sol
106:     uint256 private mintedAmount = 1;
107:
108:     /**
109:      * @notice The current reserved rate
110:      *
111:      * @dev    This is a mutex 1-x-1
112:      */
113:     uint256 private reservedRate = 1;
```

```
diff --git a/contracts/JBXBuybackDelegate.sol b/contracts/JBXBuy
index 0ee751b..23e2aa2 100644
```

```
--- a/contracts/JBXBuybackDelegate.sol
+++ b/contracts/JBXBuybackDelegate.sol
@@ -103,14 +103,14 @@ contract JBXBuybackDelegate is IJBFunding(
      *
      * @dev    This is a mutex 1-x-1
      */
-    uint256 private mintedAmount = 1;
+    uint128 private mintedAmount = 1;

     /**
      * @notice The current reserved rate
      *
      * @dev    This is a mutex 1-x-1
      */
-    uint256 private reservedRate = 1;
+    uint128 private reservedRate = 1;

     /**
      * @dev No other logic besides initializing the immutables
@@ -155,8 +155,8 @@ contract JBXBuybackDelegate is IJBFundingCyc
         // If the amount swapped is bigger than the lowest rece
         if (_tokenCount < _quote - (_quote * _slippage / SLIPPA
             // Pass the quote and reserve rate via a mutex
-            mintedAmount = _tokenCount;
-            reservedRate = _data.reservedRate;
+            mintedAmount = uint128(_tokenCount);
+            reservedRate = uint128(_data.reservedRate);

             // Return this delegate as the one to use, and do r
             delegateAllocations = new JBPayDelegateAllocation[]
```

[G-02] Create immutable variable to avoid redundant external calls

In the instances below, an external call to retrieve the `directory` is performed each time `_swap` and `_mint` is invoked. We can avoid performing this call on each invocation by executing this external call once in the constructor and storing the `directory` as an immutable variable. Doing so will save 2 external calls each time `didPay` is called (`didPay` invokes `_swap` & `_mint`).

https://github.com/code-423n4/2023-05-juicebox/blob/main/juice-buyback/contracts/JBXBuybackDelegate.sol#L290

https://github.com/code-423n4/2023-05-juicebox/blob/main/juice-buyback/contracts/JBXBuybackDelegate.sol#L335

Gas Savings for `JBXBuybackDelegate.didPay` *, obtained via protocol's tests: Avg 602 gas*

	Med	Max	Avg	# calls
Before	148266	242106	161126	7
After	147664	241504	160575	7

```
File: juice-buyback/contracts/JBXBuybackDelegate.sol
290:         IJBController controller = IJBController(jbxTerminal

335:         IJBController controller = IJBController(jbxTerminal
```

```
diff --git a/contracts/JBXBuybackDelegate.sol b/contracts/JBXBuy
index 0ee751b..403216d 100644
--- a/contracts/JBXBuybackDelegate.sol
+++ b/contracts/JBXBuybackDelegate.sol
@@ -89,6 +89,8 @@ contract JBXBuybackDelegate is IJBFundingCycle
       */
      IJBPayoutRedemptionPaymentTerminal3_1 public immutable jbxT

+     IJBDirectory private immutable directory;
+
      /**
       * @notice The WETH contract
       */
@@ -124,6 +126,7 @@ contract JBXBuybackDelegate is IJBFundingCyc
          projectToken = _projectToken;
          pool = _pool;
          jbxTerminal = _jbxTerminal;
+         directory = jbxTerminal.directory();
          _projectTokenIsZero = address(_projectToken) < address(
          weth = _weth;
      }
@@ -287,7 +290,7 @@ contract JBXBuybackDelegate is IJBFundingCyc

          // If there are reserved token, add them to the reserve
          if (_reservedToken != 0) {
```

```
-            IJBController controller = IJBController(jbxTermina
+            IJBController controller = IJBController(directory.

             // 1) Burn all the reserved token, which are in thi
             controller.burnTokensOf({
@@ -332,7 +335,7 @@ contract JBXBuybackDelegate is IJBFundingCyc
      * @param  _amount the amount of token out to mint
      */
     function _mint(JBDidPayData calldata _data, uint256 _amount
-        IJBController controller = IJBController(jbxTerminal.di
+        IJBController controller = IJBController(directory.cont

         // Mint to the beneficiary with the fc reserve rate
         controller.mintTokensOf({
```

[G-03] Use assembly to perform efficient back-to-back calls

If similar external calls are performed back-to-back, we can use assembly to reuse any function signatures and function parameters that stay the same. In addition, we can also reuse the same memory space for each function call (`scratch space` + `free memory pointer`), which can potentially allow us to avoid memory expansion costs. In this case we are also able to efficiently store both function signatures together in memory as one word, saving one MLOAD in the process.

Note: In order to do this optimization safely we will cache and restore the free memory pointer after we are done with our function calls.

https://github.com/code-423n4/2023-05-juicebox/blob/main/juice-buyback/contracts/JBXBuybackDelegate.sol#L231-L232

Gas Savings for `JBXBuybackDelegate.didPay` *, obtained via protocol's tests: Avg 392 gas*

	Med	Max	Avg	# calls
Before	148266	242106	161126	7
After	147874	241714	160902	7

Gas Savings for `JBXBuybackDelegate.uniswapV3SwapCallback` *, obtained via protocol's tests: Avg 262 gas*

	Med	Max	Avg	# calls
Before	28794	30714	20125	6
After	28402	30322	19863	6

```
File: juice-buyback/contracts/JBXBuybackDelegate.sol
231:         weth.deposit{value: _amountToSend}();
232:         weth.transfer(address(pool), _amountToSend);
```

```
diff --git a/contracts/JBXBuybackDelegate.sol b/contracts/JBXBuy
index 0ee751b..9c252d5 100644
--- a/contracts/JBXBuybackDelegate.sol
+++ b/contracts/JBXBuybackDelegate.sol
@@ -228,8 +228,20 @@ contract JBXBuybackDelegate is IJBFundingCy
         if (_amountReceived < _minimumAmountReceived) revert Ju

         // Wrap and transfer the weth to the pool
-        weth.deposit{value: _amountToSend}();
-        weth.transfer(address(pool), _amountToSend);
+        IWETH9 _weth = weth;
+        IUniswapV3Pool _pool = pool;
+        assembly {
+            // function selectors for `deposit()` & `transfer(a
+            mstore(0x00, 0xd0e30db0a9059cbb)
+            if iszero(call(gas(), _weth, _amountToSend, 0x18, 0
+            // store memory pointer
+            let memptr := mload(0x40)
+            mstore(0x20, _pool)
+            mstore(0x40, _amountToSend)
+            if iszero(call(gas(), _weth, 0x00, 0x1c, 0x44, 0x00
+            // restore memory pointer
+            mstore(0x40, memptr)
+        }
     }

     function redeemParams(JBRedeemParamsData calldata _data)
```

[G-04] Use assembly in place of `abi.decode` to extract calldata values more efficiently

Instead of using `abi.decode`, we can use assembly to decode our desired calldata values directly. This will allow us to avoid decoding calldata values that we will not use.

Total Instances: `3`

https://github.com/code-423n4/2023-05-juicebox/blob/main/juice-buyback/contracts/JBXBuybackDelegate.sol#L153

Gas Savings for `JBXBuybackDelegate.payParams`*, obtained via protocol's tests: Avg 112 gas*

	Med	Max	Avg	# calls
Before	9981	12476	8271	10
After	9886	12357	8159	10

```
File: juice-buyback/contracts/JBXBuybackDelegate.sol
153:         (,, uint256 _quote, uint256 _slippage) = abi.decode(
```

```
diff --git a/contracts/JBXBuybackDelegate.sol b/contracts/JBXBuy
index 0ee751b..6319382 100644
--- a/contracts/JBXBuybackDelegate.sol
+++ b/contracts/JBXBuybackDelegate.sol
@@ -150,7 +150,15 @@ contract JBXBuybackDelegate is IJBFundingCy
         uint256 _tokenCount = PRBMath.mulDiv(_data.amount.value

         // Unpack the quote from the pool, given by the fronter
-        (,, uint256 _quote, uint256 _slippage) = abi.decode(_da
+        uint256 _quote;
+        uint256 _slippage;
+        { // used to discard `data` variable and avoid extra st
+            bytes calldata data = _data.metadata;
+            assembly {
+                _quote := calldataload(add(data.offset, 0x40))
+                _slippage := calldataload(add(data.offset, 0x6(
+            }
+        }

         // If the amount swapped is bigger than the lowest rece
```

```
        if (_tokenCount < _quote - (_quote * _slippage / SLIPPA
```

https://github.com/code-423n4/2023-05-juicebox/blob/main/juice-buyback/contracts/JBXBuybackDelegate.sol#L196

Gas Savings for `JBXBuybackDelegate.didPay` *, obtained via protocol's tests: Avg 95 gas*

	Med	Max	Avg	# calls
Before	148266	242106	161126	7
After	148171	242011	161025	7

```
File: juice-buyback/contracts/JBXBuybackDelegate.sol
196:         (,, uint256 _quote, uint256 _slippage) = abi.decode(
```

```
diff --git a/contracts/JBXBuybackDelegate.sol b/contracts/JBXBuy
index 0ee751b..ac60192 100644
--- a/contracts/JBXBuybackDelegate.sol
+++ b/contracts/JBXBuybackDelegate.sol
@@ -193,7 +193,15 @@ contract JBXBuybackDelegate is IJBFundingCy
         reservedRate = 1;

         // The minimum amount of token received if swapping
-        (,, uint256 _quote, uint256 _slippage) = abi.decode(_da
+        uint256 _quote;
+        uint256 _slippage;
+        { // used to discard `data` variable and avoid extra st
+            bytes calldata data = _data.metadata;
+            assembly {
+                _quote := calldataload(add(data.offset, 0x40))
+                _slippage := calldataload(add(data.offset, 0x60
+            }
+        }
         uint256 _minimumReceivedFromSwap = _quote - (_quote * _

         // Pick the appropriate pathway (swap vs mint), use min
```

https://github.com/code-423n4/2023-05-juicebox/blob/main/juice-buyback/contracts/JBXBuybackDelegate.sol#L221

Gas Savings for `JBXBuybackDelegate.uniswapV3SwapCallback`*, obtained via protocol's tests: Avg 75 gas*

	Med	Max	Avg	# calls
Before	28794	30714	20125	6
After	28725	30645	20050	6

```
File: juice-buyback/contracts/JBXBuybackDelegate.sol
221:         (uint256 _minimumAmountReceived) = abi.decode(data,
```

```
diff --git a/contracts/JBXBuybackDelegate.sol b/contracts/JBXBuy
index 0ee751b..b87a72a 100644
--- a/contracts/JBXBuybackDelegate.sol
+++ b/contracts/JBXBuybackDelegate.sol
@@ -218,7 +218,10 @@ contract JBXBuybackDelegate is IJBFundingCy
         if (msg.sender != address(pool)) revert JuiceBuyback_Ur

         // Unpack the data
-        (uint256 _minimumAmountReceived) = abi.decode(data, (ui
+        uint256 _minimumAmountReceived;
+        assembly {
+            _minimumAmountReceived := calldataload(data.offset)
+        }

         // Assign 0 and 1 accordingly
         uint256 _amountReceived = uint256(-(_projectTokenIsZerc
```

[G-05] Use assembly to emit events

We can use assembly to emit events efficiently by utilizing `scratch space` and the `free memory pointer`. This will allow us to potentially avoid memory expansion costs.

Note: In order to do this optimization safely, we will need to cache and restore the free memory pointer.

https://github.com/code-423n4/2023-05-juicebox/blob/main/juice-buyback/contracts/JBXBuybackDelegate.sol#L325

https://github.com/code-423n4/2023-05-juicebox/blob/main/juice-buyback/contracts/JBXBuybackDelegate.sol#L352

Gas Savings for `JBXBuybackDelegate.didPay` *, obtained via protocol's tests: Avg 38 gas*

	Med	Max	Avg	# calls
Before	148266	242106	161126	7
After	148228	242068	161085	7

```
File: juice-buyback/contracts/JBXBuybackDelegate.sol
325:         emit JBXBuybackDelegate_Swap(_data.projectId, _data.

352:         emit JBXBuybackDelegate_Mint(_data.projectId);
```

```
diff --git a/contracts/JBXBuybackDelegate.sol b/contracts/JBXBuy
index 0ee751b..0836a78 100644
--- a/contracts/JBXBuybackDelegate.sol
+++ b/contracts/JBXBuybackDelegate.sol
@@ -322,7 +322,19 @@ contract JBXBuybackDelegate is IJBFundingCy
             }
         }

-        emit JBXBuybackDelegate_Swap(_data.projectId, _data.amo
+        assembly {
+            let memptr := mload(0x40)
+            mstore(0x00, calldataload(0x44))
+            mstore(0x20, calldataload(0xa4))
+            mstore(0x40, _amountReceived)
+            log1(
+                0x00,
+                0x60,
+                // keccak256("JBXBuybackDelegate_Swap(uint256,u
+                0x01a4fda29d012874ff22866f5058fa420a4f8598b6f52
+            )
+            mstore(0x40, memptr)
+        }
```

```
     }

     /**
@@ -349,7 +361,15 @@ contract JBXBuybackDelegate is IJBFundingCy
             _data.projectId, _data.amount.value, JBTokens.ETH,
         );

-        emit JBXBuybackDelegate_Mint(_data.projectId);
+        assembly {
+            mstore(0x00, calldataload(0x44))
+            log1(
+                0x00,
+                0x20,
+                // keccak256("JBXBuybackDelegate_Mint(uint256)"
+                0x9dd9e06f6137ca3ab76ef60c229d956aa1d09df00d9f5
+            )
+        }
     }
```

[G-06] Use assembly to validate `msg.sender`

We can use assembly to efficiently validate msg.sender for the `didPay` and `uniswapV3SwapCallback` functions with the least amount of opcodes necessary. Additionally, we can use `xor()` instead of `iszero(eq())`, saving 3 gas. We can also potentially save gas on the unhappy path by using `scratch space` to store the error selector, potentially avoiding memory expansion costs.

https://github.com/code-423n4/2023-05-juicebox/blob/main/juice-buyback/contracts/JBXBuybackDelegate.sol#L185

Gas Savings for `JBXBuybackDelegate.didPay`*, obtained via protocol's tests: Avg 21 gas*

	Med	Max	Avg	# calls
Before	148266	242106	161126	7
After	148245	242085	161107	7

`File: juice-buyback/contracts/JBXBuybackDelegate.sol`

```
185:         if (msg.sender != address(jbxTerminal)) revert Juice
```

```
diff --git a/contracts/JBXBuybackDelegate.sol b/contracts/JBXBuy
index 0ee751b..fc5566e 100644
--- a/contracts/JBXBuybackDelegate.sol
+++ b/contracts/JBXBuybackDelegate.sol
@@ -182,7 +182,13 @@ contract JBXBuybackDelegate is IJBFundingCy
      */
     function didPay(JBDidPayData calldata _data) external payab
         // Access control as minting is authorized to this dele
-        if (msg.sender != address(jbxTerminal)) revert JuiceBuy
+        IJBPayoutRedemptionPaymentTerminal3_1 _jbxTerminal = jb
+        assembly {
+            if xor(caller(), _jbxTerminal) {
+                // bytes4(keccak256("JuiceBuyback_Unauthorized(
+                mstore(0x00, 0x84f56813)
+                revert(0x1c, 0x04)
+            }
+        }

         // Retrieve the number of token created if minting and
         uint256 _tokenCount = mintedAmount;
```

https://github.com/code-423n4/2023-05-juicebox/blob/main/juice-buyback/contracts/JBXBuybackDelegate.sol#L218

Gas Savings for `JBXBuybackDelegate.uniswapV3SwapCallback`*, obtained via protocol's tests: Avg 12 gas*

	Med	Max	Avg	# calls
Before	28794	30714	20125	6
After	28784	30704	20113	6

```
File: juice-buyback/contracts/JBXBuybackDelegate.sol
218:         if (msg.sender != address(pool)) revert JuiceBuyback
```

```
diff --git a/contracts/JBXBuybackDelegate.sol b/contracts/JBXBuy
index 0ee751b..fc5566e 100644
```

```
--- a/contracts/JBXBuybackDelegate.sol
+++ b/contracts/JBXBuybackDelegate.sol
@@ -215,7 +221,13 @@ contract JBXBuybackDelegate is IJBFundingCy
      */
     function uniswapV3SwapCallback(int256 amount0Delta, int256
         // Check if this is really a callback
-        if (msg.sender != address(pool)) revert JuiceBuyback_Ur
+        IUniswapV3Pool _pool = pool;
+        assembly {
+            if xor(caller(), _pool) {
+                // bytes4(keccak256("JuiceBuyback_Unauthorized(
+                mstore(0x00, 0x84f56813)
+                revert(0x1c, 0x04)
+            }
+        }

         // Unpack the data
         (uint256 _minimumAmountReceived) = abi.decode(data, (ui
```

[G-07] Use assembly to efficiently read from and write to packed storage slots

The EVM reads from and writes to storage 1 word (32 bytes) at a time. Therefore, if we pack multiple values together into one storage slot, the EVM will do extra operations (bit masking/shifting) to fit each value into one word. With assembly we can use the least amount of opcodes necessary to read from and write to packed storage slots.

Note: This optimzation is dependent on G-01 and thus benchmarking is done using the optimization in G-01 as a baseline.

https://github.com/code-423n4/2023-05-juicebox/blob/main/juice-buyback/contracts/JBXBuybackDelegate.sol#L158-L159

https://github.com/code-423n4/2023-05-juicebox/blob/main/juice-buyback/contracts/JBXBuybackDelegate.sol#L188-L193

Gas Savings for `JBXBuybackDelegate.payParams` *, obtained via protocol's tests: Avg 18 gas*

	Med	Max	Avg	# calls
Before	6771	7523	5528	10
After	6747	7497	5510	10

Gas Savings for `JBXBuybackDelegate.didPay` *, obtained via protocol's tests: Avg 33 gas*

	Med	Max	Avg	# calls
Before	148151	241991	161803	7
After	148120	241960	161770	7

```
File: juice-buyback/contracts/JBXBuybackDelegate.sol
158:             mintedAmount = _tokenCount;
159:             reservedRate = _data.reservedRate;

188:             uint256 _tokenCount = mintedAmount;
189:             mintedAmount = 1;
190:
191:             // Retrieve the fc reserved rate and reset the m
192:             uint256 _reservedRate = reservedRate;
193:             reservedRate = 1;
```

```
diff --git a/contracts/JBXBuybackDelegate.sol b/contracts/JBXBuy
index 0ee751b..6455fe4 100644
--- a/contracts/JBXBuybackDelegate.sol
+++ b/contracts/JBXBuybackDelegate.sol
@@ -103,14 +103,14 @@ contract JBXBuybackDelegate is IJBFundingC
      *
      * @dev    This is a mutex 1-x-1
      */
-    uint256 private mintedAmount = 1;
+    uint128 private mintedAmount = 1;

     /**
      * @notice The current reserved rate
      *
      * @dev    This is a mutex 1-x-1
      */
-    uint256 private reservedRate = 1;
```

```
+    uint128 private reservedRate = 1;

     /**
      * @dev No other logic besides initializing the immutables
@@ -155,8 +155,9 @@ contract JBXBuybackDelegate is IJBFundingCyc
         // If the amount swapped is bigger than the lowest rece
         if (_tokenCount < _quote - (_quote * _slippage / SLIPPA
             // Pass the quote and reserve rate via a mutex
-            mintedAmount = _tokenCount;
-            reservedRate = _data.reservedRate;
+            assembly {
+                sstore(mintedAmount.slot, or(shr(128, shl(128,
+            }

             // Return this delegate as the one to use, and do r
             delegateAllocations = new JBPayDelegateAllocation[]
@@ -185,12 +186,14 @@ contract JBXBuybackDelegate is IJBFundingC
         if (msg.sender != address(jbxTerminal)) revert JuiceBuy

         // Retrieve the number of token created if minting and
-        uint256 _tokenCount = mintedAmount;
-        mintedAmount = 1;
-
-        // Retrieve the fc reserved rate and reset the mutex
-        uint256 _reservedRate = reservedRate;
-        reservedRate = 1;
+        uint256 _tokenCount;
+        uint256 _reservedRate;
+        assembly {
+            let storage_var := sload(mintedAmount.slot)
+            _tokenCount := shr(128, shl(128, storage_var))
+            _reservedRate := shr(128, storage_var)
+            sstore(mintedAmount.slot, or(1, shl(128, 1)))
+        }
```

GasReport output with all optimizations applied

```
| contracts/JBXBuybackDelegate.sol:JBXBuybackDelegate contract |
|---------------------------------------------------------------|
| Deployment Cost                                               |
| 1138329                                                       |
| Function Name                                                 |
| didPay                                                        |
```

```
| payParams                                                    |
| uniswapV3SwapCallback                                        |
```

[drgorillamd (Juicebox) confirmed and commented]():

> I like this.

> Some would really be a trade-off between readability and gas saving though :).

Disclosures

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

C4 Audits incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Audit 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