Report

v. 1.0

Customer

PRL

Smart Contract Audit

Buttonwood Protocol

15th May 2023

Report prepared by

ABDK Consulting

Contents

1 Changelog **4**

2 Introduction **5**

3 Project scope **6**

4 Methodology **7**

5 Our findings **8**

6 Critical Issues **9**
CVF-20. FIXED 9

7 Major Issues **10**
CVF-6. FIXED 10
CVF-7. INFO 10
CVF-21. FIXED 10
CVF-33. FIXED 11
CVF-42. INFO 11
CVF-47. FIXED 11

8 Moderate Issues **12**
CVF-14. FIXED 12
CVF-28. INFO 12

9 Minor Issues **13**
CVF-1. INFO 13
CVF-2. INFO 13
CVF-3. FIXED 14
CVF-4. FIXED 14
CVF-5. FIXED 14
CVF-8. FIXED 15
CVF-9. FIXED 15
CVF-10. FIXED 15
CVF-11. INFO 16
CVF-12. FIXED 16
CVF-13. FIXED 16
CVF-15. INFO 17
CVF-16. INFO 17
CVF-17. INFO 17
CVF-18. INFO 18
CVF-19. INFO 18
CVF-22. INFO 18
CVF-23. FIXED 19

CVF-24. INFO . . . 19
CVF-25. FIXED . . . 19
CVF-26. INFO . . . 20
CVF-27. FIXED . . . 20
CVF-29. INFO . . . 21
CVF-30. FIXED . . . 22
CVF-31. INFO . . . 22
CVF-32. INFO . . . 22
CVF-34. FIXED . . . 23
CVF-35. INFO . . . 23
CVF-36. FIXED . . . 23
CVF-37. INFO . . . 24
CVF-38. INFO . . . 24
CVF-39. FIXED . . . 24
CVF-40. FIXED . . . 25
CVF-41. FIXED . . . 25
CVF-43. FIXED . . . 25
CVF-44. INFO . . . 26
CVF-45. INFO . . . 26
CVF-46. FIXED . . . 26
CVF-48. FIXED . . . 27
CVF-49. INFO . . . 27
CVF-50. INFO . . . 28
CVF-51. FIXED . . . 29
CVF-52. FIXED . . . 29
CVF-53. FIXED . . . 29
CVF-54. INFO . . . 30
CVF-55. FIXED . . . 30
CVF-56. FIXED . . . 30
CVF-57. FIXED . . . 30
CVF-58. INFO . . . 31
CVF-59. INFO . . . 31
CVF-60. FIXED . . . 32
CVF-61. FIXED . . . 32
CVF-62. FIXED . . . 32
CVF-63. FIXED . . . 33
CVF-64. INFO . . . 33
CVF-65. FIXED . . . 34
CVF-66. FIXED . . . 34
CVF-67. INFO . . . 34

1 Changelog

#	Date	Author	Description
0.1	15.05.23	A. Zveryanskaya	Initial Draft
0.2	15.05.23	A. Zveryanskaya	Minor revision
1.0	15.05.23	A. Zveryanskaya	Release

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting (Mikhail Vladimirov and Dmitry Khovratovich) at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.

Buttonwood is a set of open protocols designed to enable a new approach to long-term debt.

3 Project scope

We were asked to review:

- Original Code
- Code with Fixes

Files:

/		
	DualAuction.sol	DualAuctionFactory.sol
utils/		
	AuctionConversions.sol	Auction ImmutableArgs.sol

4 Methodology

The methodology is not a strict formal procedure, but rather a selection of methods and tactics combined differently and tuned for each particular project, depending on the project structure and technologies used, as well as on client expectations from the audit.

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows best code practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places as well as their visibility scopes and access levels are relevant. At this phase, we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and done properly. At this phase, we understand user roles and permissions, as well as what assets the system ought to protect.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check if code actually does what it is supposed to do, if that algorithms are optimal and correct, and if proper data types are used. We also make sure that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

We classify issues by the following severity levels:

- **Critical issue** directly affects the smart contract functionality and may cause a significant loss.
- **Major issue** is either a solid performance problem or a sign of misuse: a slight code modification or environment change may lead to loss of funds or data. Sometimes it is an abuse of unclear code behaviour which should be double checked.
- **Moderate issue** is not an immediate problem, but rather suboptimal performance in edge cases, an obviously bad code practice, or a situation where the code is correct only in certain business flows.
- **Minor issues** contain code style, best practices and other recommendations.

5 Our findings

We found 1 critical, 6 major, and a few less important issues. All identified Critical issues have been fixed.

Issues	Active	Fixed
Severity		
Critical	**0**	**1**
Major	**2**	**4**
Moderate	**1**	**1**
Minor	**26**	**32**

Fixed 38 out of 67 issues

6 Critical Issues

CVF-20. FIXED

- **Category** Flaw
- **Source** DualAuction.sol

Description You cannot be sure that you actually did receive amountIn on the contract balance, because of possible fees-on-transfer or deflationary mechanism inside erc20 token.

```
117 SafeTransferLib.safeTransferFrom(
        bidAsset(),
        msg.sender,
120     address(this),
        amountIn
    );
    _mint(msg.sender, price, amountIn, "");
```

```
140 SafeTransferLib.safeTransferFrom(
        askAsset(),
        msg.sender,
        address(this),
        amountIn
    );
    _mint(msg.sender, toAskTokenId(price), amountIn, "");
```

7 Major Issues

CVF-6. FIXED

- **Category** Suboptimal
- **Source** AuctionConversions.sol

Description This call abuses a suspicious behaviour of the “toBidTokenId” function, that leads to incorrect handling of prices with the top bit set to one.

Recommendation Consider not calling the “toBidTokenId” function here, but just clearing the top bit.

```
36 return toBidTokenId(tokenId);
```

CVF-7. INFO

- **Category** Overflow/Underflow
- **Source** AuctionConversions.sol

Description This function suffers from phantom overflow problem, i.e. a situation when the final calculation result would fit into the destination type, but some intermediary calculation overflows.

Recommendation Consider using the “muldiv” function as described here: https://xn–2-mb.com/21/muldiv/

Client Comment *This is not a real issue because the muldiv function cannot suffer a phantom-overflow without reverting (simple proof-by-contradiction)*

```
51 FixedPointMathLib.mulDivDown(
```

```
71 FixedPointMathLib.mulDivDown(
```

CVF-21. FIXED

- **Category** Flaw
- **Source** DualAuction.sol

Recommendation Should be “toBidTokenId(price)” instead of just “price”.

```
123 _mint(msg.sender, price, amountIn, "");
```

CVF-33. FIXED

- **Category** Flaw
- **Source** DualAuction.sol

Recommendation Should be “bitToTokenId(currentBid)” instead of just “currentBid”.

```
183 totalSupply(currentBid),
```

CVF-42. INFO

- **Category** Flaw
- **Source** DualAuction.sol

Description The values “bidTokensClearedAtClearing” and “totalSupply(toAskTokenId(price))” do change when people redeem, thus the result of this formula for the same shareAmount and price may change after settlement.

Recommendation Consider freezing the bidTokensClearedAtClearing” and “totalSupply(toAskTokenId(price))” values at settlement time to avoid such effect.

```
306 uint256 cleared = (shareAmount * bidTokensClearedAtClearing) /
        totalSupply(toAskTokenId(price));
```

CVF-47. FIXED

- **Category** Flaw
- **Source** AuctionImmutableArgs.sol

Description Actually, a price is the number of bidAsset base units per askAsset whole token. This means that the price precision depends on the bidAsset base unit, and askAsset whole token values. If bitAsset base unit is expensive or askAsset whole token is extremely cheap, the price precision will be poor.

Recommendation Consider not relying on bid and ask asset granularities and using the same price format for all pairs. For example, consider expression a price as the number of bidAsset base unit per 2^128 askAsset base units. This would also allow using more efficient mulshift (a * b » c) and shiftdiv ((a « b) / c) functions instead of muldiv for converting asset values.

```
45 * @dev prices are denominated in terms of bidAsset per askAsset
```

8 Moderate Issues

CVF-14. FIXED

- **Category** Flaw
- **Source** DualAuction.sol

Description Exactly at the end date, both conditions are false, thus the auction is active and ended at the same time.

Recommendation Consider changing “>” to “>=” in the former condition and also changing “<” to “<=” in the line 101.

```
65 if (block.timestamp > endDate()) revert AuctionEnded();
```

```
73 if (block.timestamp < endDate()) revert AuctionActive();
```

CVF-28. INFO

- **Category** Suboptimal
- **Source** DualAuction.sol

Description This function is very inefficient as it linearly iterates through price ticks.

Recommendation Consider doing the following optimizations: 1. Calculate the settlement result (clearing price, clearing amounts at clearing price) off-chain, passing these values to the “settle” function and only checking them on-chain. 2. Aggregate tick data to make the function complexity O(ln N) instead of O(n). See the following memo for details: https://hackmd.io/@abdk/ByK1egBq9

Client Comment *Implementing our own O(sqrt(N)) algorithm instead in AuctionV2*

```
154 function settle() external onlyAuctionEnded returns (uint256) {
```

9 Minor Issues

CVF-1. INFO

- **Category** Procedural
- **Source** AuctionConversions.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as "^0.8.0". Also relevant for the next files: DualAuction.sol, AuctionImmutableArgs.sol, DualAuctionFactory.sol.

Client Comment *We keep interfaces carated and concrete contracts peg to be explicit about what our contracts support*

```
2 pragma solidity 0.8.10;
```

CVF-2. INFO

- **Category** Procedural
- **Source** AuctionConversions.sol

Description The file is not a part of the audit.

```
4 import {FixedPointMathLib} from "solmate/utils/FixedPointMathLib.sol
    ↪ ";
```

CVF-3. FIXED

- **Category** Suboptimal
- **Source** AuctionConversions.sol

Description This method does not seem to work correctly for price >= 2**255 .

Recommendation Consider adding ”require(price < 2**255)”.

16
```
function toAskTokenId(uint256 price) public pure returns (uint256) {
    // 0x10000000... | price
    // sets the top bit to 1, leaving the rest unchanged
    return price | (2**255);
```

26
```
function toBidTokenId(uint256 price) public pure returns (uint256) {
    // 0x01111111... & price
    // sets the top bit to 0, leaving the rest unchanged
    return price & (2**255 - 1);
```

CVF-4. FIXED

- **Category** Suboptimal
- **Source** AuctionConversions.sol

Description 0×100.. | price sets not the top bit but fourth highest one.

Recommendation Consider writing 0×8000...

17
```
// 0x10000000... | price
// sets the top bit to 1, leaving the rest unchanged
```

CVF-5. FIXED

- **Category** Suboptimal
- **Source** AuctionConversions.sol

Recommendation Usage of ((price « 1) » 1) looks faster than (price & (2**255-1)), also you can save (2**255-1) as a constant to avoid recalculation of the same constant every time.

29
```
return price & (2**255 - 1);
```

CVF-8. FIXED

- **Category** Suboptimal
- **Source** AuctionConversions.sol

Recommendation These values could be precomputed and stored in immutable variables.

```
54 10**bidAssetDecimals()
```

```
73 10**bidAssetDecimals(),
```

CVF-9. FIXED

- **Category** Flaw
- **Source** AuctionConversions.sol

Description Zero price doesn't seem to be valid.

Recommendation Consider reverting in case of a zero price.

```
69 if (price == 0) return 0;
```

CVF-10. FIXED

- **Category** Suboptimal
- **Source** AuctionConversions.sol

Description This code is dependent on bid decimals and may produce big rounding error on small bid decimals. What if bid decimals=0?

Recommendation You can use fixed ratio for bidToAsk price, for example first 128 bits as a nominator and last 128 bits as a denominator.

```
71 FixedPointMathLib.mulDivDown(
       bidTokens,
       10**bidAssetDecimals(),
       price
```

CVF-11. INFO

- **Category** Procedural
- **Source** DualAuction.sol

Description We didn't review these files.

```
4  import {SafeTransferLib} from "solmate/utils/SafeTransferLib.sol";
   import {ERC20} from "solmate/tokens/ERC20.sol";

9  import {Clone} from "clones-with-immutable-args/Clone.sol";
10 import {IDualAuction} from "./interfaces/IDualAuction.sol";
```

CVF-12. FIXED

- **Category** Documentation
- **Source** DualAuction.sol

Description These comments are confusing, as it is unclear what are "highest bid" and "lowest ask".

Recommendation Consider explaining that these are the highest bid price and the lowest ask price.

```
29 /// @notice The highest bid received so far

32 /// @notice The lowest ask received so far
```

CVF-13. FIXED

- **Category** Suboptimal
- **Source** DualAuction.sol

Recommendation Group 2 if-conditions into one to save gas.

```
56 if (price < minPrice() || price > maxPrice()) revert InvalidPrice();
   if ((price - minPrice()) % tickWidth() != 0) revert InvalidPrice();
```

CVF-15. INFO

- **Category** Suboptimal
- **Source** DualAuction.sol

Description It's still possible to clone auction with no factory, it may break backend system, maybe add nonReentrant here or add onlyInitialized to the every function below?

Client Comment *We only support auctions that have been deployed via our factory contract. Creating auctions outside of that is not expected to necessarily work, and frontend users will not be exposed to those auctions.*

```
88 function initialize() external initializer {
```

CVF-16. INFO

- **Category** Suboptimal
- **Source** DualAuction.sol

Description This check is redundant. It is anyway possible to pass a dead asset address.

Client Comment *Just because we can't stop all bad addresses, doesn't mean we shouldn't prevent zero-address. Will keep.*

```
93 if (
       address(bidAsset()) == address(0) ||
       address(askAsset()) == address(0)
   ) revert InvalidAsset();
```

CVF-17. INFO

- **Category** Suboptimal
- **Source** DualAuction.sol

Recommendation Maybe also check that the price range is not too wide (to not face into out-of-gas).

Client Comment *Taken care of by hard-coded tick-limit of 100*

```
100 if ((maxPrice() - minPrice()) % tickWidth() != 0) revert
      ↪ InvalidPrice();
```

CVF-18. INFO

- **Category** Unclear behavior
- **Source** DualAuction.sol

Description Should it be maxPrice() to avoid misusing of the top bit value?

Client Comment *Keeping the minAsk and maxAsk out of the valid price range, guarantees that the unitialized values aren't confused for real bids/asks. However, we did add more unit tests to validate.*

```
102 minAsk = type(uint256).max;
```

CVF-19. INFO

- **Category** Suboptimal
- **Source** DualAuction.sol

Description The signature of this function is confusing. One could think that it spends at most "amountIn" of the bid asset to buy the ask asset at a price not exceeding the given maximum price. However, actually, it buys at most amountIn / price of the ask asset. In other words, in case the clearing price is less than the given price, it decreases the bid asset amount spent, rather than increase the ask asset amount obtained.

Recommendation It would be less confusing to accept the output ask asset amount instead of the input bid asset amount.

Client Comment *We disagree in changing the function signature at this point. This is overall a gas optimization as it aggregates the bidToAsk operations per price to once at settle() time, rather than during each bid.*

```
108 function bid(uint256 amountIn, uint256 price)
```

CVF-22. INFO

- **Category** Documentation
- **Source** DualAuction.sol

Recommendation It's better to add explicit comment into the code, that here we use amountIn as a tokenId because we are sure that the top-bit equals 0 because of the onlyValidPrice modifier

Client Comment *We have no idea what this means because Bid event is not recording tokenId and amountIn is not being used as one either.*

```
124 emit Bid(msg.sender, amountIn, amountIn, price);
```

CVF-23. FIXED

- **Category** Unclear behavior
- **Source** DualAuction.sol

Description Does it make sense to return the input argument? We think in this function we should calculate actual received amount via two balanceOf calls and return actual amountIn.

```
125 return amountIn;
```

```
148 return amountIn;
```

CVF-24. INFO

- **Category** Suboptimal
- **Source** DualAuction.sol

Description This event is redundant, as "_mint" emits an event that has all the necessary information.

Client Comment *As a result of CVF: 20, this ticket is wrong because the event emitted by '_mint' will only contain amountOut which may not equal amountIn for deflationary tokens.*

```
124 emit Bid(msg.sender, amountIn, amountIn, price);
```

```
147 emit Ask(msg.sender, amountIn, amountIn, price);
```

CVF-25. FIXED

- **Category** Unclear behavior
- **Source** DualAuction.sol

Description How minAsk could be 0? It is set to type(uint256).max inside initialization.

```
139 if (minAsk == 0 || price < minAsk) minAsk = price;
```

CVF-26. INFO

- **Category** Suboptimal
- **Source** DualAuction.sol

Recommendation Note, that bidders/askers do not want to call settle because it's gas expensive, people have no economical stimulation to call it. Maybe introduce some mechanism to compensate the gas.

Client Comment *Will implement in V2*

```
153 */
```

CVF-27. FIXED

- **Category** Suboptimal
- **Source** DualAuction.sol

Recommendation Safer to add nonReentrant modifier because this function potentially may be called from the inside another nonReentrant function and settle will modifie the state.

```
154 function settle() external onlyAuctionEnded returns (uint256) {
```

CVF-29. INFO

- **Category** Flaw
- **Source** DualAuction.sol

Description Possible out-of-gas, possible DoS attack if someone fills all price range to increase operations number. Introduce a mechanism to call this function in rounds to process e.g. 100 iterations in one call. This way we will always be able to finish it. You can also check if (maxPrice - minPrice) / tickWidth < 1000 to limit the iterations number. Think about using of balanced trees to do matching in LOG(N) time.

Recommendation Note, that block gaslimit may change in blockchain to smaller values in future, there is no guarantee that it will not change.

Client Comment *It's impossible to do matching in LOG(N) time.*

```
171 while (
        currentBid >= currentAsk &&
        currentBid >= minPrice() &&
        currentAsk <= maxPrice()
    ) {
        if (currentAskTokens == 0) {
            currentAskTokens = totalSupply(toAskTokenId(currentAsk));
            if (currentAskTokens > 0) lastBidClear = 0;
        }
180
        if (currentDesiredAskTokens == 0) {
            currentDesiredAskTokens = bidToAsk(
                totalSupply(currentBid),
                currentBid
            );

            if (currentDesiredAskTokens > 0) lastAskClear = 0;
        }

190     uint256 cleared = min(currentAskTokens, currentDesiredAskTokens)
           ↪ ;

        if (cleared > 0) {
            currentAskTokens -= cleared;
            currentDesiredAskTokens -= cleared;
            lastBidClear += cleared;
            lastAskClear += cleared;
            highAsk = currentAsk;
            lowBid = currentBid;
        }
200 } ...
```

CVF-30. FIXED

- **Category** Suboptimal
- **Source** DualAuction.sol

Description These conditions don't have to be checked on every iteration.

Recommendation Consider checking once before the loop.

173
```
currentBid >= minPrice() &&
currentAsk <= maxPrice()
```

CVF-31. INFO

- **Category** Suboptimal
- **Source** DualAuction.sol

Recommendation In case of "else" condition you can immediatelly continue to the next iteration, because it's clear that no match will happen in this iteration. So just add tickWidth to the price and chekc if it's in correct price range and continue the loop.

Client Comment *Added additional test-cases to invalidate this. Opted against this approach since we found that on the average case, it will perform worse. Experiments show that it's better to move both price-ticks at the same time. Moving one at a time causes worse performance since you're entering the loop more often and doing more comparisons while the askTokens pointer waits for the desiredAskTokens pointer to catch up.*

178
```
if (currentAskTokens > 0) lastBidClear = 0;
```

187
```
if (currentDesiredAskTokens > 0) lastAskClear = 0;
```

CVF-32. INFO

- **Category** Unclear behavior
- **Source** DualAuction.sol

Description What about dust? It may happen that we will just skip some dust amount because of the rounding error.

Client Comment *Given the granularity of priceDenominator(), it's more optimal to configure it such that the dust is minimally small, rather than trying to savage it. Will look into configuring ways to prevent this by restricting bid sizes for AuctionsV2.*

182
```
currentDesiredAskTokens = bidToAsk(
```

CVF-34. FIXED

- **Category** Suboptimal
- **Source** DualAuction.sol

Recommendation Cheaper to read tickWidth once to a local variable.

```
201 if (currentAskTokens == 0) currentAsk += tickWidth();
    if (currentDesiredAskTokens == 0) currentBid -= tickWidth();
```

CVF-35. INFO

- **Category** Suboptimal
- **Source** DualAuction.sol

Recommendation This value is calculated often (on every redeem), you can cache it inside the contract storage variable for the less gas usage.

Client Comment *Function is only called once at settle and once per redeemer. Since 2 sloads are 1/50th an sstore, it doesn't make sense for settler to eat the cost unless there are at least 49 redeemers.*

```
207 uint256 _clearingPrice = clearingPrice();
```

CVF-36. FIXED

- **Category** Suboptimal
- **Source** DualAuction.sol

Recommendation You can use explicit function "isBid(tokenId) {return tokenId & 1«255}"

```
226 bool isBid = toBidTokenId(tokenId) == tokenId;
```

CVF-37. INFO

- **Category** Documentation
- **Source** DualAuction.sol

Description It is unclear if it is ever possible that the min arguments are unequal.

Recommendation Consider documenting this case and checking it is okay.

233
```
bidTokens = min(bidTokens, bidAsset().balanceOf(address(this)));
```

240
```
askTokens = min(askTokens, askAsset().balanceOf(address(this)));
```

CVF-38. INFO

- **Category** Suboptimal
- **Source** DualAuction.sol

Recommendation Instead of reading 2 storage slots on every call you can calculate this value once and cache it.

Client Comment *More gas-efficient in the current implementation*

250
```
function clearingPrice() public view override returns (uint256) {
    uint256 _clearingBid = clearingBidPrice;
    uint256 _clearingAsk = clearingAskPrice;
    if (_clearingBid == _clearingAsk) {
        return _clearingBid;
    } else {
        return (_clearingBid + _clearingAsk) / 2;
    }
}
```

CVF-39. FIXED

- **Category** Suboptimal
- **Source** DualAuction.sol

Recommendation This seems like a rare case, you can always follow the logic at the line 256 to save gas on rare condition.

253
```
if (_clearingBid == _clearingAsk) {
```

CVF-40. FIXED

- **Category** Suboptimal
- **Source** DualAuction.sol

Recommendation You can check it comparing price and tokenId with ”if (tokenId == price)”.

```
275 if (toBidTokenId(tokenId) == tokenId) {
```

CVF-41. FIXED

- **Category** Overflow/Underflow
- **Source** DualAuction.sol

Description Phantom overflow is possible here.

Recommendation Consider using the “muldiv” function as described here: https://xn–2-mb.com/21/muldiv/

```
289 uint256 cleared = (shareAmount * askTokensClearedAtClearing) /
290     totalSupply(price);
```

CVF-43. FIXED

- **Category** Suboptimal
- **Source** DualAuction.sol

Recommendation You can use <= operator, it will save you a bit of gas in case if two values are equal.

```
310 uint256 notCleared = askValue < cleared
        ? 0
        : bidToAsk(askValue - cleared, _clearingPrice);
```

CVF-44. INFO

- **Category** Procedural
- **Source** AuctionImmutableArgs.sol

Description The file is not a part of the audit.

Client Comment *Libraries come from reputable sources*

```
5 import {Clone} from "clones-with-immutable-args/Clone.sol";
```

CVF-45. INFO

- **Category** Suboptimal
- **Source** AuctionImmutableArgs.sol

Description We are not sure if this approach really save gas, first of all because it takes a lot of gas on proxy deployment and then every time you do a call to the proxy it copies all args to calldata. Also this approach requires additional internal functions to access calldata value.

Recommendation The straight-forward approach to store everything in one contract on one mapping looks cheaper. You can also read immutable args not from calldata but from the contract bytecode directly.

Client Comment *We want to minimise the cost of creating a new auction since we expect end users to frequently create them. Each auction has a limited lifetime and users are expected to typically only place one bid/ask. As such auction operations suffer the gas cost in this tradeoff between them and auction creation cost. The proxy deployment is significantly smaller bytecode than the fullblown auction and is a substantial gas saving.*

```
9 * @dev using the clones-with-immutable-args library
```

CVF-46. FIXED

- **Category** Unclear behavior
- **Source** AuctionImmutableArgs.sol

Description There is no comment about price denomination as on the 46 line. Does it mean the the manner of usage is different?

```
35 * @dev using ClonesWithImmutableArgs pattern here to save gas
```

CVF-48. FIXED

- **Category** Suboptimal
- **Source** AuctionImmutableArgs.sol

Description In ERC-20 the “decimals” property is used by UI to render token amounts in a human-friendly way. Using this property in smart contracts is discouraged.

Recommendation Consider treating all token amounts as integers.

```
82 function bidAssetDecimals() public pure returns (uint256) {
```

```
92 function askAssetDecimals() public pure returns (uint256) {
```

CVF-49. INFO

- **Category** Procedural
- **Source** DualAuctionFactory.sol

Description The file is not a part of the audit.

Client Comment *Libraries come from reputable sources.*

```
4 import {ClonesWithImmutableArgs} from "clones-with-immutable-args/
    ↪ ClonesWithImmutableArgs.sol";
```

CVF-50. INFO

- **Category** Suboptimal
- **Source** DualAuctionFactory.sol

Description It's not clear why do we want to deploy separate Auction contract for the every new auction instead of keeping the separate mapping in one contract for all auctions, it could save a lot of gas.

Client Comment *Clones make it easier to change contracts/factories. Significant refactor required to make this change.*

```
11  contract DualAuctionFactory is IAuctionFactory {
        using SafeERC20Upgradeable for IERC20Upgradeable;
        using ClonesWithImmutableArgs for address;

        address public immutable implementation;

        constructor(address _implementation) {
            implementation = _implementation;
        }
20
        /**
         * @notice Creates a new auction
         * @param bidAsset The asset that bids are made with
         * @param askAsset The asset that asks are made with
         * @param minPrice The minimum allowed price in terms of
            ↪ bidAsset
         * @param maxPrice The maximum allowed price in terms of
            ↪ bidAsset
         * @param tickWidth The spacing between valid prices
         * @param endDate The timestamp at which the auction will end
         */
30      function createAuction(
            address bidAsset,
            address askAsset,
            uint256 minPrice,
            uint256 maxPrice,
            uint256 tickWidth,
            uint256 endDate
        )
    }
```

(... 40, 50)

CVF-51. FIXED

- **Category** Suboptimal
- **Source** DualAuctionFactory.sol

Recommendation The contract is not upgradeable, so there is no need to use upgradeable libraries.

```
12 using SafeERC20Upgradeable for IERC20Upgradeable;
```

CVF-52. FIXED

- **Category** Suboptimal
- **Source** DualAuctionFactory.sol

Description There is a common practice to check if address value is not zero to be sure the caller did not misuse it.

Recommendation Add require(value != address(0)) here

```
18 implementation = _implementation;
```

CVF-53. FIXED

- **Category** Documentation
- **Source** DualAuctionFactory.sol

Recommendation Add NatSpec description for the return value.

```
21 /**
 * @notice Creates a new auction
 * @param bidAsset The asset that bids are made with
 * @param askAsset The asset that asks are made with
 * @param minPrice The minimum allowed price in terms of bidAsset
 * @param maxPrice The maximum allowed price in terms of bidAsset
 * @param tickWidth The spacing between valid prices
 * @param endDate The timestamp at which the auction will end
 */
```

CVF-54. INFO

- **Category** Bad datatype
- **Source** DualAuctionFactory.sol

Recommendation The type of these arguments should be “IERC20”.

Client Comment *Casting doesn't improve performance and casting from openeppelin/IERC20 to solmate/ERC20 adds more issues (especially since the additional import will increase deployment size)*

```
31 address bidAsset,
   address askAsset,
```

CVF-55. FIXED

- **Category** Procedural
- **Source** DualAuctionFactory.sol

Recommendation The function should be defined external to safe some gas.

```
37 ) public returns (address) {
```

CVF-56. FIXED

- **Category** Bad datatype
- **Source** DualAuctionFactory.sol

Recommendation The return type should be “IDualAuction”.

```
37 ) public returns (address) {
```

CVF-57. FIXED

- **Category** Suboptimal
- **Source** DualAuctionFactory.sol

Description In ERC-20 the “decimals” property is used by UI to render token amounts in a human-friendly way. Using this property in smart contracts is discouraged.

Recommendation Consider treating all token amounts are integers.

```
45 IERC20MetadataUpgradeable(bidAsset).decimals(),
   IERC20MetadataUpgradeable(askAsset).decimals()
```

CVF-58. INFO

- **Category** Suboptimal
- **Source** DualAuctionFactory.sol

Recommendation It's much more straight-forward to set arguments on initialization rather that via immutable args clone. It is not crystal clear how does ImmutableArgsClone works, just pass arguments to "initialize" method.

Client Comment *More gas-efficient in the current way*

```
38  bytes memory data = abi.encodePacked(
        bidAsset,
40      askAsset,
        minPrice,
        maxPrice,
        tickWidth,
        endDate,
        IERC20MetadataUpgradeable(bidAsset).decimals(),
        IERC20MetadataUpgradeable(askAsset).decimals()
    );
    DualAuction clone = DualAuction(implementation.clone(data));

50  clone.initialize();
```

CVF-59. INFO

- **Category** Procedural
- **Source** IDualAuction.sol

Description Files are not part of the audit.

Client Comment *Libraries come from reputable sources*

```
4   import {SafeTransferLib} from "solmate/utils/SafeTransferLib.sol";
    import {ERC20} from "solmate/tokens/ERC20.sol";
```

```
9   import {Clone} from "clones-with-immutable-args/Clone.sol";
```

CVF-60. FIXED

- **Category** Bad naming
- **Source** IDualAuction.sol

Recommendation Rename with ”ZeroAddressAsset”.

```
13 error InvalidAsset();
```

CVF-61. FIXED

- **Category** Bad naming
- **Source** IDualAuction.sol

Recommendation Rename with ”ZeroAmount”.

```
16 error InvalidAmount();
```

CVF-62. FIXED

- **Category** Bad datatype
- **Source** IDualAuction.sol

Recommendation Rename following the grammar rules, e.g. rename ”AuctionActive” with ”AuctionIsActive” or ”ActiveAuction” for better human readability.

```
22 error AuctionActive();
```

```
25 error AuctionEnded();
```

```
28 error AuctionNotSettled();
```

```
31 error AuctionSettled();
```

```
37 error SettleFailed();
```

CVF-63. FIXED

- **Category** Suboptimal
- **Source** IDualAuction.sol

Recommendation Important fields should be ”indexed”, at least actor and price.

```
40 event Bid(
       address actor,
       uint256 amountIn,
       uint256 amountOut,
       uint256 price
   );
```

```
48 event Ask(
       address actor,
50     uint256 amountIn,
       uint256 amountOut,
       uint256 price
   );
```

```
56 event Settle(address actor, uint256 clearingPrice);
```

```
59 event Redeem(
60     address actor,
       uint256 tokenId,
       uint256 shareAmount,
       uint256 bidValue,
       uint256 askValue
   );
```

CVF-64. INFO

- **Category** Unclear behavior
- **Source** IDualAuction.sol

Description What is the difference between amountIn and amountOut?

Client Comment *Using amountOut to represent actual balance changes in the case of deflationary assets*

```
42 uint256 amountIn,
   uint256 amountOut,
```

```
50 uint256 amountIn,
   uint256 amountOut,
```

CVF-65. FIXED

- **Category** Suboptimal
- **Source** IAuctionFactory.sol

Description Interface does not define any external function.

Recommendation It's better to add "createAuction" method here since this is the only external way to use the contract.

```
7 interface IAuctionFactory {
```

CVF-66. FIXED

- **Category** Suboptimal
- **Source** IAuctionFactory.sol

Recommendation Important fields should be "indexed" for cheap searching inside the contract logs.

```
8 event AuctionCreated(
```

CVF-67. INFO

- **Category** Suboptimal
- **Source** IAuctionFactory.sol

Recommendation The type should be ERC20

Client Comment *Casting doesn't affect performance and and importing ERC20 library will increase deployment size.*

```
9  address bidAsset,
10 address askAsset,
```

ABDK
Consulting

About us

Established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function.

The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

Contact

Email

dmitry@abdkconsulting.com

Website

abdk.consulting

Twitter

twitter.com/ABDKconsulting

LinkedIn

linkedin.com/company/abdk-consulting