Report

v. 1.0

Customer

The Arena

Smart Contract Audit

Starsarena fractional tickets

11th March 2025

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	11.03.25	A. Zveryanskaya	Initial Draft
0.2	11.03.25	A. Zveryanskaya	Minor revision
1.0	11.03.25	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.

3 Project scope

We were asked to review:

- Original Code
- Code with Fixes

Files:

/

StarsArena.sol

interfaces/

IStarsArenaBeforeUpgrade.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.
- **Recommendations** contain code style, best practices and other suggestions.

5 Our findings

We found 2 major, and a few less important issues.

Fixed 1 out of 2 issues

6 Major Issues

CVF-1 INFO

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

Description This means that the caller may get less for his shares just because the protocol doesn't have enough money.

Recommendation Revert or reduce the "amount" value in case TVL isn't enough.

Client Comment *Acknowledged. In practice it will never come to that and even if ti does, it will be a minuscule amount.*

321
```
// solvency guard
if(price > sharesTvl) {
    price = sharesTvl;
}
```

CVF-2 FIXED

- **Category** Flaw
- **Source** StarsArena.sol

Description If there is no referrer for the user, a zero referred will be used and returned which could be dangerous.

Recommendation Explicitly check that "referrer" isn't zero, before using the "referrer" value.

Client Comment *Fixed as suggested. A check is introduced and if the referrer is zero, the function directly returns zero address as the referrer.*

535
```
address referrer = userToReferrer[user];
address signerOfReferrer = userToSigner[referrer];
```

7 Moderate Issues

CVF-3 INFO

- **Category** Suboptimal
- **Source** StarsArena.sol

Description Hardcoding Mainnet addresses makes it harder testing the code.

Recommendation Pass the addresses as constructor arguments and save them in immutable variables.

Client Comment *We would like to keep the flexibility of changing the protocol fee addresses on the fly without a re-deployment. We find the storage read cost acceptable.*

```
10 protocolFeeDestination = address(0
      ↪ xAc0388Fe24D65358f2fF063ebCbEfa321A2a091d);
   protocolFeeDestination2 = address(0
      ↪ xd650f696816c3B635bb3B92A8146D05adcBf9d34);
```

CVF-4 FIXED

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

Recommendation These functions should emit some events.

```
98  function setPaused(bool _paused) external onlyOwner {
119 function setReferralFeePercent(uint256 _feePercent) public onlyOwner
       ↪  {
125 function setProtocolFeePercent(uint256 _feePercent) public onlyOwner
       ↪  {
131 function setSubjectFeePercent(uint256 _feePercent) public onlyOwner
       ↪ {
137 function setFeeDestination(address _feeDestination) external {
142 function setFeeDestination2(address _feeDestination2) external {
```

CVF-5 FIXED

- **Category** Suboptimal
- **Source** StarsArena.sol

Recommendation Should be "<=".

```
121 require(_feePercent < maxFeePercent, "Invalid␣fee␣setting");
```

```
127 require(_feePercent < maxFeePercent, "Invalid␣fee␣setting");
```

```
132 uint256 maxFeePercent = 8 ether / 100;
```

CVF-6 FIXED

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

Description Here, a too large amount may cause revert doe to underflow, and such revert won't have any meaningful error message.

Recommendation Put the "require" statement ahead of the line where too large "amount" value could cause a revert.

Client Comment *Fixed as suggested. Removed other redundant storage reads as well.*

```
211 uint256 price = getPrice(supply - amount, amount);
```

```
217 require(sharesBalance[sharesSubject][user] >= amount, "Insufficient␣
      ↪ shares");
```

CVF-7 INFO

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

Description This line may revert due to underflow.

Recommendation Perform an explicit check to ensure price doesn't exceed TVL, and if it does, revert with a meaningful error message.

Client Comment *The insolvency guard will take care of this issue.*

```
212 tvl[sharesSubject] -= price;
```

CVF-8 INFO

- **Category** Suboptimal
- **Source** StarsArena.sol

Description This makes it impossible to unset signer.

Recommendation Either make it possible to unset signer, or clearly explain, why such operation should be prohibited.

Client Comment *Unsetting a signer is prohibited by intentional design.*

```
237 require(newSigner != address(0),"New␣signer␣can␣not␣be␣zero␣address!
    ↪ ");
```

CVF-9 FIXED

- **Category** Suboptimal
- **Source** StarsArena.sol

Description This conditional statement is redundant, as it signer isn't set for a user, then the "_verifySigner" function performs the "msg.sender == user" check.

Recommendation Remove the conditional statement along with the "else" branch, and leave only the content of the "then" branch.

```
242 if(userToSigner[user] != address(0)) {
        _verifySigner(user);
```

```
247 else {
        require(msg.sender == user, "Only␣original␣user␣can␣set␣a␣new␣a␣
            ↪ signer");
```

CVF-10 FIXED

- **Category** Procedural
- **Source** StarsArena.sol

Description Here an implicit underflow check is used to enforce a business-level constraint, which makes code harder to read and more error-prone.

Recommendation Add an explicit check to ensure the supply is sufficient.

```
319 uint256 price = getPriceForFractionalShares(supply - amount, amount)
    ↪ ;
```

CVF-11 FIXED

- **Category** Procedural
- **Source** StarsArena.sol

Description Here an implicit underflow check is used to enforce a business-level constraint, which makes code harder to read and more error-prone.

Recommendation Add an explicit check to ensure the balance is sufficient.

```
330 fractionalSharesBalance[sharesSubject][user]-=amount; // burn, will
      ↪ revert on overflow
```

CVF-12 FIXED

- **Category** Procedural
- **Source** StarsArena.sol

Description Here an implicit underflow check is used to enforce a business-level constraint, which makes code harder to read and more error-prone.

Recommendation Add an explicit check to ensure the balance is sufficient.

```
379 fractionalSharesBalance[sharesSubject][from] -= amount; // burn;
```

CVF-13 FIXED

- **Category** Procedural
- **Source** StarsArena.sol

Description The original errors message returned from the failed “call” is dropped, which could make it harder investigating problems.

Recommendation Return a named error, and include the original error message as a parameter.

```
581 (bool success,) = sharesSubject.call{value: subjectFee}("");
    require(success, "Unable to send funds");
```

CVF-14 FIXED

- **Category** Procedural
- **Source** StarsArena.sol

Description The original errors message returned from the failed “call” is dropped, which could make it harder investigating problems.

Recommendation Return a named error, and include the original error message as a parameter.

```
586 (bool success,) = protocolFeeDestination.call{value: protocolFee}("
       ↪ );
    require(success, "Unable to send funds");
```

8 Minor Issues

CVF-15 FIXED

- **Category** Procedural
- **Source** IStarsArenaBeforeUpgrade.sol

Description This interface isn't used and its role is unclear. The words "BeforeUpgrade" in the interface name look weird.

Recommendation Remove this interface or clearly explain its role.

Client Comment *Fixed as suggested. We moved the interface to the test folder.*

```
4 interface IStarsArenaBeforeUpgrade {
```

CVF-16 FIXED

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

Recommendation The type for the "sharesSubject" argument should be more specific.

Client Comment *If we understand correctly, the suggestion here is to make the address types to address payable for sharesSubject. We understand that this is a better practice and we should use this in order to use transfer or send but we are already using. call to transfer ether and changing the data type to address payable will require a lot of refactoring work on tests and break things on other external contracts that use our interface. Thus, we will keep the datatype as is.*

```
25 function getMyShares(address sharesSubject) external view returns (
   ↪ uint256);
27 function getSharesSupply(address sharesSubject) external view
   ↪ returns (uint256);
29 function getBuyPrice(address sharesSubject, uint256 amount) external
   ↪  view returns (uint256);
31 function getSellPrice(address sharesSubject, uint256 amount)
   ↪ external view returns (uint256);
33 function userToReferrer(address sharesSubject) external view returns
   ↪  (address);
```

```
35 function getReferrer(address sharesSubject) external view returns (
   ↪ address);
```

```
48 function getBuyPriceAfterFee(address sharesSubject, uint256 amount)
   ↪ external view returns (uint256);
```

```
50 function getSellPriceAfterFee(address sharesSubject, uint256 amount)
   ↪  external view returns (uint256);
```

```
52 function buySharesWithReferrer(address sharesSubject, uint256 amount
   ↪ , address referrer) external payable;
```

```
54 function sellSharesWithReferrer(address sharesSubject, uint256
   ↪ amount, address referrer) external payable;
```

```
56 function buyShares(address sharesSubject, uint256 amount) external
   ↪ payable;
```

```
58 function sellShares(address sharesSubject, uint256 amount) external
   ↪ payable;
```

CVF-17 FIXED

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

Description The “test” directory mentioned in production code look weird.

Recommendation Use different name.

```
6 import "../test/ABDKMath64x64.sol";
```

CVF-18 FIXED

- **Category** Procedural
- **Source** StarsArena.sol

Description Placing a function before the constructor looks odd.

Recommendation Place the constructor first.

```
9 function initialize() public initializer {
```

```
23 constructor() {
```

CVF-19 INFO

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

Recommendation These values should be named constants.

Client Comment *Acknowledged. We will keep them as is.*

```
10  protocolFeeDestination = address(0
       ↪ xAc0388Fe24D65358f2fF063ebCbEfa321A2a091d);
    protocolFeeDestination2 = address(0
       ↪ xd650f696816c3B635bb3B92A8146D05adcBf9d34);
    subjectFeePercent = 7 ether / 100;
    protocolFeePercent = 2 ether / 100;
    referralFeePercent = 1 ether / 100;
    initialPrice = 1 ether / 250;
    subscriptionDuration = 30 days;
```

```
120 uint256 maxFeePercent = 2 ether / 100;
```

```
126 uint256 maxFeePercent = 4 ether / 100;
```

```
132 uint256 maxFeePercent = 8 ether / 100;
```

CVF-20 FIXED

- **Category** Suboptimal
- **Source** StarsArena.sol

Description Using the “ether” unit is confusing, as these values have nothing to do with ether amounts.

Recommendation Use the WAD constant (equal to 1e18) instead.

```
12      subjectFeePercent = 7 ether / 100;
        protocolFeePercent = 2 ether / 100;
        referralFeePercent = 1 ether / 100;
        initialPrice = 1 ether / 250;
```

```
44  uint256 constant DEFAULT_WEIGHT_A = 80 ether / 100;
    uint256 constant DEFAULT_WEIGHT_B = 50 ether / 100;
```

```
120     uint256 maxFeePercent = 2 ether / 100;
```

```
126     uint256 maxFeePercent = 4 ether / 100;
```

```
132     uint256 maxFeePercent = 8 ether / 100;
```

```
177     uint256 protocolFee = price * protocolFeePercent / 1 ether;
178     uint256 subjectFee = price * getSubjectFeePercent(sharesSubject)
            ↪  / 1 ether;
179     uint256 referralFee = price * referralFeePercent / 1 ether;
```

```
214     uint256 protocolFee = price * protocolFeePercent / 1 ether;
215     uint256 subjectFee = price * getSubjectFeePercent(sharesSubject)
            ↪  / 1 ether;
216     uint256 referralFee = price * referralFeePercent / 1 ether;
```

```
283     uint256 protocolFee = price * protocolFeePercent / 1 ether;
284     uint256 subjectFee = price * getSubjectFeePercent((sharesSubject
            ↪ )) / 1 ether;
285     uint256 referralFee = price * referralFeePercent / 1 ether;
```

```
327     uint256 protocolFee = price * protocolFeePercent / 1 ether;
328     uint256 subjectFee = price * getSubjectFeePercent(sharesSubject)
            ↪  / 1 ether;
329     uint256 referralFee = price * referralFeePercent / 1 ether;
```

```
364     uint256 individualFeePercent =  feeAmountInPercent * 1 ether /
            ↪ 100;
```

```
400     uint256 price = DEFAULT_WEIGHT_B * summation * initialPrice / 1
            ↪ ether / 1 ether;
```

```
434     uint256 protocolFee = price * protocolFeePercent / 1 ether;
435     uint256 subjectFee = price * subjectFeePercent / 1 ether;
436     uint256 referralFee = price * referralFeePercent / 1 ether;
```

```
442     uint256 protocolFee = price * protocolFeePercent / 1 ether;
443     uint256 subjectFee = price * subjectFeePercent / 1 ether;
444     uint256 referralFee = price * referralFeePercent / 1 ether;
```

```
471     uint256 finalPrice = (intermediateRes * initialPrice *
            ↪ DEFAULT_WEIGHT_B) / 1 ether / 1 ether;
```

```
510     uint256 protocolFee = price * protocolFeePercent / 1 ether;
511     uint256 subjectFee = price * getSubjectFeePercent(sharesSubject)
            ↪   / 1 ether;
512     uint256 referralFee = price * referralFeePercent / 1 ether;
```

```
518     uint256 protocolFee = price * protocolFeePercent / 1 ether;
519     uint256 subjectFee = price * getSubjectFeePercent(sharesSubject)
            ↪  / 1 ether;
520     uint256 referralFee = price * referralFeePercent / 1 ether;
```

CVF-21 FIXED

- **Category** Readability
- **Source** StarsArena.sol

Recommendation This value could be rendered as “0.07 ether”.

Client Comment *Changed ether denomination as WAD as suggested in CVF-20.*

```
12 subjectFeePercent = 7 ether / 100;
```

CVF-22 FIXED

- **Category** Readability
- **Source** StarsArena.sol

Recommendation This value could be rendered as “0.02 ether”.

Client Comment *Changed ether denomination as WAD as suggested in CVF-20.*

```
13 protocolFeePercent = 2 ether / 100;
```

CVF-23 FIXED

- **Category** Readability
- **Source** StarsArena.sol

Recommendation This value could be rendered as “0.01 ether”.

Client Comment *Changed ether denomination as WAD as suggested in CVF-20.*

```
14 referralFeePercent = 1 ether / 100;
```

CVF-24 FIXED

- **Category** Readability
- **Source** StarsArena.sol

Recommendation This value could be rendered as “0.004 ether”.

Client Comment *Changed ether denomination as WAD as suggested in CVF-20.*

```
15 initialPrice = 1 ether / 250;
```

CVF-25 INFO

- **Category** Suboptimal
- **Source** StarsArena.sol

Recommendation Unchained initializers should be used to prevent double initialization.

Client Comment *Acknowledged. This contract is already initialized and we dont intend to re-initialize in the future. If our understanding is correct, this shouldnt be an issue in our case.*

```
18 __Ownable_init();
   __ReentrancyGuard_init();
```

CVF-26 INFO

- **Category** Procedural
- **Source** StarsArena.sol

Recommendation The value calculated here is actually constant and could be precomputed.

Client Comment *Acknowledged. We believe having this as an immutable variable is good enough.*

```
25 DEFAULT_WEIGHT_C_ABDK = ABDKMath64x64.fromUInt(DEFAULT_WEIGHT_C);
```

CVF-27 FIXED

- **Category** Documentation
- **Source** StarsArena.sol

Description The number format for these variables is unclear.

Recommendation Explain in a documentation comment.

```
33 uint256 public protocolFeePercent;
   uint256 public subjectFeePercent;
   uint256 public referralFeePercent;
```

CVF-28 INFO

- **Category** Suboptimal
- **Source** StarsArena.sol

Recommendation It would be more efficient to merge these mappings into a single mapping whose keys are addresses and values are structs, encapsulating the values of the original mappings.

Client Comment *Acknowledged. Most of those mappings are remmants of the initial contract and are not used or wont be used. We cant remove them due to storage layout preservation. The rest, there are some who are called together but many to warrant a change. We will keep the mappings as is.*

```
38 mapping(address => uint256) public weightA;
   mapping(address => uint256) public weightB;
40 mapping(address => uint256) public weightC;
   mapping(address => uint256) public weightD;
   mapping(address => bool) private weightsInitialized;
```

```
60 mapping(address => uint256) public revenueShare;
   mapping(address => uint256) public subscriptionPrice;
   mapping(address => bool) public subscriptionsEnabled;
```

```
65 mapping(address => mapping(address => uint256)) public subscribers;
```

```
67 mapping(address => address[]) public shareholders;
```

```
70 mapping(address => mapping(address => uint256)) public sharesBalance
      ↪ ;
```

```
73 mapping(address => uint256) public sharesSupply;
```

```
75 mapping(address => address) public subscriptionTokenAddress;
   mapping(address => bool) public allowedTokens;
   mapping(address => uint256) public pendingWithdrawals;
   mapping(address => mapping(address => uint256)) public
      ↪ pendingTokenWithdrawals;
```

```
82 mapping(address => uint256) tvl;
```

```
84 mapping(address => address) public userToSigner;
   mapping(address => mapping(address => uint256)) public
      ↪ fractionalSharesBalance;
   mapping(address => uint256) public fractionalSharesSupply;
   mapping (address => uint256) public subjectToFeePercent;
```

CVF-29 FIXED

- **Category** Documentation
- **Source** StarsArena.sol

Description The number format for the values of these mappings is unclear.

Recommendation Explain in a documentation comment.

Client Comment *Fixed with note. Only subjectToFeePercent is used among those mappings. We have added the documentation for this and added commets to clarify that those mappings are not used.*

```
38 mapping(address => uint256) public weightA;
   mapping(address => uint256) public weightB;
40 mapping(address => uint256) public weightC;
   mapping(address => uint256) public weightD;
```

```
60 mapping(address => uint256) public revenueShare;
   mapping(address => uint256) public subscriptionPrice;
```

```
87 mapping (address => uint256) public subjectToFeePercent;
```

CVF-30 FIXED

- **Category** Readability
- **Source** StarsArena.sol

Recommendation This value could be rendered as “0.8 ether”.

Client Comment *This is a constant bonding curver parameter. This will never be changed. Changed to WAS as suggested in CVF-20.*

```
44 uint256 constant DEFAULT_WEIGHT_A = 80 ether / 100;
```

CVF-31 FIXED

- **Category** Readability
- **Source** StarsArena.sol

Recommendation This value could be rendered as “0.5 ether”.

Client Comment *This is a constant bonding curver parameter. This will never be changed. Changed to WAS as suggested in CVF-20.*

```
45 uint256 constant DEFAULT_WEIGHT_B = 50 ether / 100;
```

CVF-32 FIXED

- **Category** Documentation
- **Source** StarsArena.sol

Description The number format for these constants is unclear.

Recommendation Explain in a documentation comment.

Client Comment *Added documentation.*

```
46 uint256 constant DEFAULT_WEIGHT_C = 2;
   uint256 constant DEFAULT_WEIGHT_D = 0;
```

CVF-33 FIXED

- **Category** Documentation
- **Source** StarsArena.sol

Description The number format for the “buyPrice” parameters is unclear.

Recommendation Explain in a documentation comment.

Client Comment *Added documentation*

```
52 event Trade(address trader, address subject, bool isBuy, uint256
   ↪ shareAmount, uint256 amount, uint256 protocolAmount, uint256
   ↪ subjectAmount, uint256 referralAmount, uint256 supply, uint256
   ↪  buyPrice, uint256 myShares);
   event TradeFractionalShares(address trader, address subject, bool
   ↪ isBuy, uint256 shareAmount, uint256 amount, uint256
   ↪ protocolAmount, uint256 subjectAmount, uint256 referralAmount,
   ↪  uint256 fractionalSupply, uint256 buyPrice, uint256
   ↪ myFractionalShares);
```

CVF-34 INFO

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

Recommendation Events are usually named via nouns, such as “Referral” or “Signer”.

Client Comment *Acknowledged. We have already some indexerrs that listen to those events so an update on the names will also mean additional maintenance work on those. Thus, we will leave the event names as is.*

```
54 event ReferralSet(address user, address referrer);
   event SignerSet(address user,address signer, address previousSigner)
      ↪ ;
   event IndividualSubjectFeeSet(address user,uint256
      ↪ individualFeePercent);
   event TransferFractionalShares(address sharesSubject, address from,
      ↪ address to, uint256 amount);
```

CVF-35 INFO

- **Category** Procedural
- **Source** StarsArena.sol

Recommendation Address parameters should be indexed.

Client Comment *Acknowledged. We have already some indexerrs that listen to those events so an update on the names will also mean additional maintenance work on those. Thus, we will leave the events as is.*

```
54 event ReferralSet(address user, address referrer);
   event SignerSet(address user,address signer, address previousSigner)
      ↪ ;
   event IndividualSubjectFeeSet(address user,uint256
      ↪ individualFeePercent);
   event TransferFractionalShares(address sharesSubject, address from,
      ↪ address to, uint256 amount);
```

CVF-36 FIXED

- **Category** Procedural
- **Source** StarsArena.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

```
89 receive() external payable {}
```

CVF-37 INFO

- **Category** Suboptimal
- **Source** StarsArena.sol

Recommendation It would be more efficient to pass a single array of structs with two fields, rather than two parallel arrays. This would also make the length check unnecessary.

Client Comment *Acknowledged. This function is not intended for frequent use so we will keep it as is.*

```
111 function updateReferrers(address[] calldata users, address[]
    ↪ calldata referrers) external onlyOwner {
```

CVF-38 FIXED

- **Category** Readability
- **Source** StarsArena.sol

Recommendation This value could be rendered as “0.02 ether”.

Client Comment *We have swapped ether to WAD as suggested in CVF-20.*

```
120 uint256 maxFeePercent = 2 ether / 100;
```

CVF-39 FIXED

- **Category** Readability
- **Source** StarsArena.sol

Recommendation This value could be rendered as “0.04 ether”.

Client Comment *We have swapped ether to WAD as suggested in CVF-20.*

```
126 uint256 maxFeePercent = 4 ether / 100;
```

CVF-40 FIXED

- **Category** Readability
- **Source** StarsArena.sol

Recommendation This value could be rendered as “0.08 ether”.

Client Comment *We have swapped ether to WAD as suggested in CVF 20.*

```
132 uint256 maxFeePercent = 8 ether / 100;
```

CVF-41 INFO

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

Description This function should return the total amount of ether consumed.

Client Comment *Acknowledged. We will keep the function as is.*

```
168 function buySharesForUser(address sharesSubject, address user,
    ↪ uint256 amount) public payable nonReentrant {
```

CVF-42 FIXED

- **Category** Suboptimal
- **Source** StarsArena.sol

Description The value “sharesSupply[sharesSubject]” is read from the storage twice.

Recommendation Read once and reuse.

Client Comment *Fixed as suggested. The commit also addresses a compilation issue introduced in another commit.*

```
172 require(sharesSupply[sharesSubject] > 0, "User␣other␣methods␣for␣
    ↪ fractionalized␣tickets!");
uint256 supply = sharesSupply[sharesSubject];
```

CVF-43 FIXED

- **Category** Documentation
- **Source** StarsArena.sol

Description The error message seems incorrect. Also, exclamation sign in an error message looks odd.

Recommendation Rephrase the error message.

Client Comment *Updated the error message as suggested.*

```
172 require(sharesSupply[sharesSubject] > 0, "User␣other␣methods␣for␣
    ↪ fractionalized␣tickets!");
```

```
209 require(sharesSupply[sharesSubject] > 0, "User␣other␣methods␣for␣
    ↪ fractionalized␣tickets!");
```

CVF-44 INFO

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

Description Phantom overflow is possible here, i.e. a situation when the final calculation result would fit into the destination type while some intermediary calculation overflows.

Recommendation Use the “mulDiv” function.

Client Comment *Acknowledged. We appreciate the suggestion and understand that this is the best practise but for our use case, its unrealsitic to reach that large of prices or intermediate sums for this to happen. We will keep those variables as is.*

```
177 uint256 protocolFee = price * protocolFeePercent / 1 ether;
    uint256 subjectFee = price * getSubjectFeePercent(sharesSubject) / 1
    ↪  ether;
    uint256 referralFee = price * referralFeePercent / 1 ether;
```

```
214 uint256 protocolFee = price * protocolFeePercent / 1 ether;
    uint256 subjectFee = price * getSubjectFeePercent(sharesSubject) / 1
    ↪  ether;
    uint256 referralFee = price * referralFeePercent / 1 ether;
```

```
283 uint256 protocolFee = price * protocolFeePercent / 1 ether;
    uint256 subjectFee = price * getSubjectFeePercent((sharesSubject)) /
    ↪  1 ether;
    uint256 referralFee = price * referralFeePercent / 1 ether;
```

```
327 uint256 protocolFee = price * protocolFeePercent / 1 ether;
    uint256 subjectFee = price * getSubjectFeePercent(sharesSubject) / 1
       ↪  ether;
    uint256 referralFee = price * referralFeePercent / 1 ether;

364 uint256 individualFeePercent =  feeAmountInPercent * 1 ether / 100;

400 uint256 price = DEFAULT_WEIGHT_B * summation * initialPrice / 1
       ↪ ether / 1 ether;

434 uint256 protocolFee = price * protocolFeePercent / 1 ether;
    uint256 subjectFee = price * subjectFeePercent / 1 ether;
    uint256 referralFee = price * referralFeePercent / 1 ether;

442 uint256 protocolFee = price * protocolFeePercent / 1 ether;
    uint256 subjectFee = price * subjectFeePercent / 1 ether;
    uint256 referralFee = price * referralFeePercent / 1 ether;

471 uint256 finalPrice = (intermediateRes * initialPrice *
       ↪ DEFAULT_WEIGHT_B) / 1 ether / 1 ether;

510 uint256 protocolFee = price * protocolFeePercent / 1 ether;
    uint256 subjectFee = price * getSubjectFeePercent(sharesSubject)  /
       ↪ 1 ether;
    uint256 referralFee = price * referralFeePercent / 1 ether;

518 uint256 protocolFee = price * protocolFeePercent / 1 ether;
    uint256 subjectFee = price * getSubjectFeePercent(sharesSubject) / 1
       ↪  ether;
520 uint256 referralFee = price * referralFeePercent / 1 ether;
```

CVF-45 FIXED

- **Category** Suboptimal
- **Source** StarsArena.sol

Description The value “price + protocolFee + subjectFee + referralFee” is calculated twice.

Recommendation Calculate once and reuse.

```
180 require(msg.value >= price + protocolFee + subjectFee + referralFee,
       ↪  "Insufficient␣payment");

190 uint256 refundAmount = msg.value - (price + protocolFee + subjectFee
       ↪  + referralFee);
```

CVF-46 FIXED

- **Category** Suboptimal
- **Source** StarsArena.sol

Description The storage address of “sharesBalance[sharesSubject][user]” is calculated twice.

Recommendation Use the “+=” operator.

```
182 sharesBalance[sharesSubject][user] = sharesBalance[sharesSubject][
       ↪ user] + amount;
```

CVF-47 FIXED

- **Category** Documentation
- **Source** StarsArena.sol

Description It is unclear, what “true” does mean here.

Recommendation Put the parameter name as a comment next to the “true” value.

Client Comment *Fixed as suggested for other events as well.*

```
201 emit Trade(user, sharesSubject, true, amount, price, protocolFee,
       ↪ subjectFee, referralFee, totalShares, nextPrice, myShares);
```

CVF-48 INFO

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

Description This function should return the total amount of ether paid to ”msg.sender”.

Client Comment *Acknowledged. We will keep the function as is.*

```
205 function sellSharesForUser(address sharesSubject, address user,
       ↪ uint256 amount) public payable nonReentrant {
```

CVF-49 FIXED

- **Category** Suboptimal
- **Source** StarsArena.sol

Description The value “sharedBalance[sharesSubject][user]” is read from the storage twice.

Recommendation Read once and reuse.

Client Comment *Fixed with CVF-51.*

```
217 require(sharesBalance[sharesSubject][user] >= amount, "Insufficient␣
       ↪ shares");
    sharesBalance[sharesSubject][user] = sharesBalance[sharesSubject][
       ↪ user] - amount;
```

CVF-50 FIXED

- **Category** Suboptimal
- **Source** StarsArena.sol

Description The storage address of “sharesBalance[sharesSubject][user]” is calculated twice.

Recommendation Use the “-=” operator.

```
218 sharesBalance[sharesSubject][user] = sharesBalance[sharesSubject][
       ↪ user] - amount;
```

CVF-51 FIXED

- **Category** Suboptimal
- **Source** StarsArena.sol

Description Here, a value just written into the storage is read back again.

Recommendation Reuse the written value.

```
218 sharesBalance[sharesSubject][user] = sharesBalance[sharesSubject][
       ↪ user] - amount;
```

```
221 uint256 myShares = sharesBalance[sharesSubject][user];
```

CVF-52 FIXED

- **Category** Suboptimal
- **Source** StarsArena.sol

Description The fraction shares supply value is read from the storage twice.

Recommendation Read once and reuse.

Client Comment *Fixed as suggested with better naming.*

278
```
uint256 supply = getTotalFractionalSupply(sharesSubject);
```

289
```
fractionalSharesSupply[sharesSubject] += amount;
```

CVF-53 FIXED

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

Recommendation The value “10” should be a named constant.

Client Comment *Fixed as suggested.*

279
```
uint256 price = getPriceForFractionalShares(supply, amount) + 10; //
    ↪  to prevent a possibly curve illiquidity due to precision
    ↪ errors
```

491
```
return getPriceForFractionalShares(totalSupply,amount) + 10;
```

CVF-54 FIXED

- **Category** Suboptimal
- **Source** StarsArena.sol

Description The value “price + protocolFee + subjectFee + referralFee” is calculated twice.

Recommendation Calculate once and reuse.

Client Comment *Fixed as suggested.*

286
```
require(msg.value >= price + protocolFee + subjectFee + referralFee,
    ↪  "Insufficient␣payment");
```

297
```
uint256 refundAmount = msg.value - (price + protocolFee + subjectFee
    ↪  + referralFee);
```

CVF-55 FIXED

- **Category** Suboptimal
- **Source** StarsArena.sol

Description The fractional shares supply value is read from the storage twice.

Recommendation Read once and reuse.

Client Comment *Fixed as suggested in addition to other redundant storage read fixes.*

318
```
uint256 supply = getTotalFractionalSupply(sharesSubject);
```

331
```
fractionalSharesSupply [sharesSubject] -= amount; // decrease total
   ↪ supply, will revert on overflow
```

CVF-56 INFO

- **Category** Suboptimal
- **Source** StarsArena.sol

Recommendation This could be simplified as: uint256 sum1 = (adjustedSupply - 1) * (adjustedSupply) * (2 * adjustedSupply - 1) / 6;

Client Comment *Acknowledged. We choose to keep it that way for easier readibility. Merging everything makes it difficult to underrstand the underlying bonding curve formula.*

397
```
uint256 sum1 = (adjustedSupply - 1) * (adjustedSupply) * (2 * (
   ↪ adjustedSupply - 1) + 1) / 6;
```

CVF-57 FIXED

- **Category** Suboptimal
- **Source** StarsArena.sol

Recommendation This could be optimized as: uint256 price = DEFAULT_WEIGHT_B * summation * initialPrice / (1 ether * 1 ether);

Client Comment *Fixed as suggested. We are also using WAD instead of ether as suggested.*

400
```
uint256 price = DEFAULT_WEIGHT_B * summation * initialPrice / 1
   ↪ ether / 1 ether;
```

CVF-58 FIXED

- **Category** Readability
- **Source** StarsArena.sol

Recommendation Should be “else return” for readability.

```
404 return price;
```

CVF-59 FIXED

- **Category** Suboptimal
- **Source** StarsArena.sol

Recommendation This could be simplified and optimized as: if (amount == 0 || sharesSupply[sharesSubject] < amount) return 0;

```
420 if (sharesSupply[sharesSubject] == 0) {
        return 0;
    }
    if (amount == 0) {
        return 0;
    }
    if (sharesSupply[sharesSubject] < amount) {
        return 0;
    }
```

CVF-60 FIXED

- **Category** Suboptimal
- **Source** StarsArena.sol

Description The value “sharesSupply[sharesSubject]” is read from the storage several times.

Recommendation Read once and reuse.

```
420 if (sharesSupply[sharesSubject] == 0) {
```

```
426 if (sharesSupply[sharesSubject] < amount) {
```

```
429 return getPrice(sharesSupply[sharesSubject] - amount, amount);
```

CVF-61 FIXED

- **Category** Suboptimal
- **Source** StarsArena.sol

Recommendation The value calculated here is actually a constant and should be pre-computed.

```
465 int128 abdk_6 = ABDKMath64x64.fromUInt(6);
```

```
622 int128 abdk_3 = ABDKMath64x64.fromUInt(3);
    int128 abdk_2 = ABDKMath64x64.fromUInt(2);
```

CVF-62 FIXED

- **Category** Suboptimal
- **Source** StarsArena.sol

Recommendation This could be optimized as: uint256 finalPrice = (intermediateRes * initialPrice * DEFAULT_WEIGHT_B) / (1 ether * 1 ether);

```
471 uint256 finalPrice = (intermediateRes * initialPrice *
       ↪ DEFAULT_WEIGHT_B) / 1 ether / 1 ether;
```

CVF-63 FIXED

- **Category** Suboptimal
- **Source** StarsArena.sol

Recommendation This could be simplified and optimized as: if (amount == 0 || amount > totalSupply) return 0;

```
496 if (totalSupply == 0) {
        return 0;
    }
    if (amount == 0) {
500     return 0;
    }
    if(amount > totalSupply) {
        return 0;
    }
```

CVF-64 FIXED

- **Category** Readability
- **Source** StarsArena.sol

Recommendation Should be “else return”.

```
505 return getPriceForFractionalShares(totalSupply - amount,amount);
```

CVF-65 FIXED

- **Category** Readability
- **Source** StarsArena.sol

Recommendation Should be “else return”.

```
531 return subjectFeePercent;
```

CVF-66 FIXED

- **Category** Readability
- **Source** StarsArena.sol

Recommendation Should be “else return”.

```
540 return referrer;
```

CVF-67 FIXED

- **Category** Suboptimal
- **Source** StarsArena.sol

Recommendation Inner brackets are redundant.

```
599 if (traderReferrer != address(0) && traderReferrer != sender && (
      ↪ userToSigner[sender] != traderReferrer)) {
```

CVF-68 FIXED

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

Recommendation The value “30_000” should be a named constant.

```
600 (bool success,) = traderReferrer.call{value: referralFeeHalf, gas:
      ↪ 30_000}("");
```

```
609 (bool success,) = subjectReferrer.call{value: referralFeeHalf, gas:
      ↪ 30_000}("");
```

CVF-69 FIXED

- **Category** Suboptimal
- **Source** StarsArena.sol

Recommendation It should be “referralFee - referralFeeHalf” instead of just “referralFee-Half” just in case “referralFee” is odd.

```
609 (bool success,) = subjectReferrer.call{value: referralFeeHalf, gas:
      ↪ 30_000}("");
```

```
611     referrerExcess += referralFeeHalf;
```

```
614 referrerExcess += referralFeeHalf;
```

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