SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: VeChain
Date: 28 Nov, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for VeChain
Approved By	Niccolò Pozzolini | Lead Solidity SC Auditor Maksym Fedorenko | Co-Auditor Paul Fomichov | Approver
Tags	Marketplace
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://www.vechain.org/
Changelog	16.11.2023 - Initial Review 28.11.2023 - Second Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by VeChain (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

System Overview

The audit scope is composed by three implementations of marketplaces for ERC721 tokens (VIP181 since the project is going to be deployed on VeChain):

- OfferContractVIP180 - a contract which allows to set offers for specific NFTs or NFT collections, the purchase offer might be made with ERC-20/VIP-180 tokens.
- WorldOfVBidAuction - a contract for NFT auctions, which allows to set up auctions for the NFT and specify the token which should be used for purchase. Purchase might be made via an ERC-20/VIP-180 token or native token.
- WorldOfVFixedPriceNonCustodial - non custodial marketplace where the seller creates a listing and anyone as a buyer may execute the sale by providing the price requested by the seller.

Privileged roles

- The role DEFAULT_ADMIN_ROLE can set “rotate” code to change the operator, and modify the foundation fee.
- The role OPERATOR_ROLE can:
  - Cancel any listing on WorldOfVFixedPriceNonCustodial contract;
  - Cancel any auction on WorldOfVBidAuction contract;
  - Pause the contract to block such functionality:
    - OfferContrctVIP180
      - create a purchase offer for NFT or collection;
      - close purchase offer;
    - WorldOfBidAuction
      - create auction place a bid;
      - execute sale;
      - cancel auction;
    - WorldOfVFixedPriceNonCustodial
      - create listing;
      - make a purchase;
      - cancel listing;
      - cancel listing by admin.
  - Whitelist/de-whitelist VIP-180 and VIP-181 tokens which might be used;
  - Modify enterprise fees;

- Withdraw VTHO tokens from the contracts.

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are provided;
- Technical description is provided;
- NatSpec is complete.

Code quality

The total Code Quality score is **10** out of **10**.

- The development environment is configured;

Test coverage

Code coverage of the project is **90.2%** (branch coverage).

- Deployment and basic user interactions are covered with tests.

Security score

As a result of the audit, the code contains **no** severity issue. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.6**. The system users should acknowledge all the risks summed up in the risks section of the report.

1
2
3
4
5
6
7
8
9
10
The final score

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
15 November 2023	7	1	3	0
28 November 2023	0	0	0	0

Risks

- Fee-on-transfer tokens should not be whitelisted as they are not compatible with the marketplaces.

- The marketplaces allow for enterprise and foundation fees, each one is capped to 20%. There is thus the possibility for the marketplace fees to be 40% of the paid price.
- The fees amount specified by NFT contracts following the ERC2981 standard, is overridden to be <= 50%.
- The contracts allow OPERATOR_ROLE to perform actions on behalf of the users. Such functionality should be carefully implemented through backend checks to prevent misuse from malicious users.
- The contract WorldOfVBidAuction allows to create an auction with a start date in the past, while checking that the end date is in the future.
- Auctions on WorldOfVBidAuction are not enforced: the seller can cancel them even when they are expired.
- Operators can cancel any auction on WorldOfVBidAuction and any listing on WorldOfVFixedPriceNonCustodial
- Fees in WorldOfVFixedPriceNonCustodial are retroactive: when changed by the administrators, also open listings are affected.
- VTHO tokens might be withdrawn from the contract by the Operator.
- The contracts are upgradable, the logic and implementation might be changed by the admin.
- The proxy implementation is out of scope.
- When the OfferContractVIP180 contract is paused any existing offer might still be accepted.

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

H01. Requirement Violation

Impact	High
Likelihood	High

The NatSpec of the function *makeBuyOfferToToken()* specifies that "Amount of the offer is put in escrow until the offer is withdrawn or superseded", but this is not happening.

The function only checks that the user making the offer is allowing the offer amount to the contract.

Path: ./contracts/OfferContractVIP180.sol : makeBuyOfferToToken()

Recommendation: Align the NatSpec requirements to the code logic, by either adjusting the NatSpec or implementing the missing functionality.

Found in: ab09639

Status: Fixed (Revised commit: b2568fc)

H02. Denial of Service - Broken Functionality of the Auction

Impact	High
Likelihood	High

The marketplace *WorldOfBidAuction* implements an auction sale model. When a user gets outbidden, the previous highest bid is refund through a push based approach susceptible to Denial of Service.

This refund should be pull based, because the implemented push based design is vulnerable to a malicious contract reverting the transaction when receiving the funds. It suffices for such a malicious contract to implement a *receive()* function with a revert.

As a consequence, the auction being targeted cannot be outbidden, as any attempt will revert. The malicious user would win any auction with only one bid.

The auction cannot even be canceled since the *cancelAuction()* function has the same problem.

Proof of Concept:

1. Deploy a contract with the following features:
    - A *receive()* function which reverts
    - A *makeBid()* function to perform the bid on the marketplace
    - *ERC721TokenReceiver*'s *onERC721Received()* function to be able to receive the NFT
2. Make a bid on an auction through the *makeBid()* function of the malicious contract
3. Any further *bid()* on the marketplace will revert even if the funds are higher than the malicious bid. The same applies for *cancelAuction()*

Path: ./contracts/WorldOfBidAuction.sol : bid(), cancelAuction()

Recommendation: The general solution as already mentioned is to implement a pull based approach: the user must come back to the contract and call a specifically designed function to withdraw their funds.

When a user gets outbidden, the contract should just increase an internal balance for him to come and withdraw the related funds.

Found in: ab09639

Status: Fixed (Revised commit: 7282b76)

H03. Requirements Violation, Admin Cannot Cancel Users' Auctions

Impact	High
Likelihood	High

According to the implementation of the *WorldOfBidAuction* marketplace, any auction might be canceled at any time during the auction by the admin. This is done using the *cancelAdminAuction()* function, which sends the NFT back to the seller by calling the *ERC721(token).safeTransferFrom()* function. However, if the seller is a contract, this transfer execution might be reverted in the *onERC721Received* callback function, when the seller does not wish to end the auction.

This leads to the requirement violation and results in the inability to cancel the auction by the admin.

Another minor situation arising from a possible abuse of the *ERC721(token).safeTransferFrom()* functionality, is in the *executeSale()* function. A malicious max bidder could prevent the execution of *executeSale()* from the seller by reverting the NFT receival. This combined with issue H03 would lead to the lock of the NFT on sale, which could be unlocked only by outbidding the malicious user. The likelihood of this minor situation is low, since the attacker would also have his bidden funds locked together with the NFT.

Proof of Concept:

1. The seller is a malicious contract which reverts on the *onERC721Received()* function call when some flag is set and works according to the standard otherwise.
2. The seller creates an auction from the "seller contract" and transfers its NFT from it.
3. The seller sets the "seller contract" to revert the execution when *onERC721Received()* function is called.

The Admin may not cancel the auction, *OPERATOR_ROLE* or *maxBidUser* are not able to execute the sale until the seller decides to.

Path: ./contracts/WorldOfBidAuction.sol : cancelAdminAuction(), executeSale()

Recommendation: Implement a pull based approach: the user must call a specifically designed function to withdraw their NFTs, so the NFT transfer may not block the function execution.

Found in: ab09639

Status: Fixed (Revised commit: 0036400)

Medium

M01. Redundant receive() functionality

Impact	Medium
Likelihood	Low

WorldOfVFixedPriceNonCustodial and *WorldOfVBidAuction* contracts implement a *receive()* function, which is not required.

Mistakes from users would lead to funds loss. The funds would not even be recovered by the administrators since a withdraw function is lacking.

Path: ./contracts/WorldOfVFixedPriceNonCustodial.sol : receive()

./contracts/WorldOfVBidAuction.sol : receive()

Recommendation: Remove the *receive()* functions as they pose an unnecessary risk for the users.

Found in: ab09639

Status: Fixed (Revised commit: f83a19e)

Low

L01. Check-Effect-Interaction Pattern Violation

Impact	Medium
Likelihood	Low

It is considered following best practices to avoid unclear situations and prevent common attack vectors.

The Checks-Effects-Interactions pattern is violated. During the *bid* function call, the state variables *auction.maxBidUser*, *auction.maxBid* are updated after the external calls.

This may lead to reentrancies, race conditions, and denial of service vulnerabilities during implementation of new functionality.

Paths: ./contracts/WorldOfVBidAuction.sol : createTokenAuction(), bid()

Recommendation: Follow common best practices and implement the function according to the Checks-Effects-Interactions pattern.

Found in: ab09639

Status: Fixed (Revised commit: f14c207)

L02. Funds Collected By Operators Instead Of By Admins

Impact	Medium
Likelihood	Low

The function *adminWithdrawEnergy()* lets operators withdraw VTHO tokens from the marketplace contracts.

If such operation is part of the protocol operativity, it is okay to be performed by operators, but the collected funds should not end up in their control. VTHO tokens should be withdrawn to an address solely controlled by the marketplace administrators.

Path: ./contracts/BaseMarketplaceContract.sol : adminWithdrawEnergy()

Recommendation: Change the funds recipient from the *msg.sender* operator to a fund controlled by the administrators.

Found in: ab09639

Status: Fixed (Revised commit: f2e625e)

L03. Redundant SafeMath Library

Impact	Medium

Likelihood	Low

The marketplace contracts adopt SafeMath library for math operations, while using solc version 0.8.22.

Prior to Solidity version 0.8.0, arithmetic overflows were not handled natively by the language, and developers were encouraged to use the SafeMath library as a safeguard against such errors.

However, with the release of Solidity version 0.8.0, the language introduced new arithmetic overflow and underflow protection features that made the SafeMath library redundant if using Solc versions above 0.8.0.

Paths: ./contracts/OfferContractVIP180.sol
./contracts/WorldOfVBidAuction.sol
./contracts/WorldOfVFixedPriceNonCustodial.sol

Recommendation: If the project utilizes a Solidity version above 0.8.0, it is recommended to avoid using the SafeMath library for arithmetic operations. Instead, leverage the native arithmetic overflow and underflow protection features provided by Solidity from version 0.8.0 onwards.

Found in: ab09639

Status: Fixed (Revised commit: 761237)

L04. Unclear Require Statement Error Strings

Impact	Medium
Likelihood	Low

Error strings in require statements are often unclear and confusing. It is suggested to also let the error sentence start with a capital letter.

If the error strings have been made short to save Gas, the require statement should be replaced by the *if/revert* pattern with custom errors.

Path:

./contracts/OfferContractVIP180.sol : lines 200 (the contract uses a whitelist, not a blacklist), 202, 253,

./contracts/WorldOfVBidAuction.sol : lines 142, 189,

./contracts/WorldOfVFixedPriceNonCustodial.sol : lines 107, 163, 164, 165, 227, 228, 252,

Recommendation: Change the error messages making them meaningful to let the users understand what happened and how to prevent the error from happening again.

Found in: ab09639

Status: Fixed (Revised commit: b302b29)

L05. Unvalidated Parameters

Impact	Medium
Likelihood	Low

The variables *WorldOfVBidAuction.timerDuration* and *BaseMarketplaceContract.CODE_VALIDITY* are heavily used throughout the contracts while not being validated when set during the contracts initialization.

Since no setter is available for such variables, an erroneous setting would require a contract redeploy.

Path:

./contracts/BaseMarketplaceContract.sol : CODE_VALIDITY

./contracts/WorldOfVBidAuction.sol : timerDuration

Recommendation: Validate the variables by checking that their value is not zero, and possibly bound it to reasonable values.

Found in: ab09639

Status: Fixed (Revised commit: 773c2a6)

L06. Missing Event Emitting

Impact	Medium
Likelihood	Low

The contracts within the system do not emit events during the important state changes such as fees changes, tokens withdrawals by the admin and tokens whitelisting/de-whitelisting.

Events for critical state changes should be emitted for tracking contract activity off-chain.

Paths:

./contracts/BaseMarketplaceContract.sol : rotate(), adminModifyFoundationFee(), adminModifyEnterpriseFee(), adminModifyVIP180(), adminModifyVIP181(), adminWithdrawEnergy().

Recommendation: Implement event emitting for all important state changes to inform users and tracking things off-chain.

Found in: ab09639

Status: Fixed (Revised commit: 8b74769)

L07. Unfinalized Code

Impact	Low
Likelihood	Medium

The contracts have multiple commented lines and TODO comments, qualifying the provided contracts as drafts.

Security audits are tied to specific code versions through commits, the audited code should thus be the final version to be deployed.

Path: ./contracts/OfferContractVIP180.sol : lines 150, 192, 203

./contracts/WorldOfVBidAuction.sol : line 39

./contracts/WorldOfVFixedPriceNonCustodial.sol : lines 33, 156, 246

Recommendation: Finalize the contracts development.

Found in: ab09639

Status: Fixed (Revised commit: 58db80c)

Informational

I01. Usage of Magic Numbers Instead Of Constants

The contracts make large use of numeric literals, which should be defined as constant variables to improve readability and code clarity, since the variable name would already detail the numbers' means.

Path: ./contracts/BaseMarketplaceContract.sol : init(), adminModifyFoundationFee(), adminModifyEnterpriseFee()

./contracts/OfferContractVIP180.sol : initialize(), handleTransfer()

./contracts/WorldOfVBidAuction.sol : executeSale()

./contracts/WorldOfVFixedPriceNonCustodial.sol : initialize(), handleProjectRoyaltySplit()

Recommendation: Declare numeric literal as constants and use the declared constant variables throughout the contracts' logic.

Found in: ab09639

Status: Fixed (Revised commit: 0c50377)

I02. Redundant Mapping Keys

Each marketplace contract declares a mapping to map three keys to the auction/sale/listing.

The mappings are:

- *OfferContractVIP180.offerToToken*
- *OfferContractVIP180.collectionOfferToToken*
- *WorldOfVBidAuction.tokenToAuction*
- *WorldOfVFixedPriceNonCustodial.tokenToSale*

The three keys are:

- *offerId/auctionId/saleId*
- The NFT collection address
- The NFT's *tokenID* - missing in *OfferContractVIP180.collectionOfferToToken* but the issue is still valid with a single redundant key

Since the *offerId/auctionId/saleId* are unique, the other two keys are redundant.

The variables *offerId/auctionId/saleId* are made non-overlapping through the marketplaces contract; at line 23 of *OfferContractVIP180* contract the developers noted in a comment "using this trick we can keep the saleId as parameter in the URL to ease the URL size", but this is not the case if the other two keys are not removed from the mappings.

Paths:

./contracts/OfferContractVIP180.sol

./contracts/WorldOfVBidAuction.sol

./contracts/WorldOfVFixedPriceNonCustodial.sol

Recommendation: Remove the redundant keys and enforce the relationship *offerId/auctionId/saleId* to NFT through other means, such as *require* statements or *if/revert*.

If the client desires to keep the mappings as-is, the adoption of named mappings is suggested to improve the code readability.

Found in: ab09639

Status: Fixed (Revised commit: e33f97a)

I03. Unindexed offerType Events Parameter

The marketplace contract *OfferContractVIP180* allows for two type of offers: offers to specific tokens and offers to any token of a specific collection.

The events are being shared between the two offers, and the *offerType* parameter is not indexed because the maximum number of indexed parameters has been reached.

Off-chain tracking will be inefficient, unless the events get duplicated for the two offer types.

Path: ./contracts/OfferContractVIP180.sol

Recommendation: To improve off-chain events tracking, duplicate the events to have one set for token offers and one set for collection offers.

Found in: ab09639

Status: Fixed (Revised commit: e33f97a)

I04. Code Duplication

The functions *acceptCollectionBuyOffer()* and *acceptTokenBuyOffer()* implement the same code besides the *offerInfo* variable declaration.

A large portion of code is also shared by the functions *cancelAuction()* and *cancelAdminAuction()* in *WorldOfVBidAuction* contract.

Code duplication within the contract leads to increased deployment gas costs and decreased code quality.

Path: ./contracts/OfferContractVIP180.sol

Recommendation: Refactor the duplicated code segments into reusable functions or employ appropriate design patterns to eliminate code duplication.

Found in: ab09639

Status: Fixed (Revised commit: cf4d39f)

I05. Redundant _msgSender(), Meta-Transactions Not Implemented

The *_msgSender()* function is needed to handle the meta transactions, in the OpenZeppelin library, it is used to support the development of the libraries which might be used with the contracts with the specified *TrustedForwarder* or to be used in such contracts directly.

However, the current implementation is not a library and does not rewrite the *_msgSender()* function to support meta transactions.

This affects the code clarity.

Paths:

./contracts/OfferContractVIP180.sol

./contracts/WorldOfVBidAuction.sol

./contracts/WorldOfVFixedPriceNonCustodial.sol

./contracts/BaseMarketplaceContract.sol

Recommendation: Replace the *_msgSender()* with a regular *msg.sender*.

Found in: ab09639

Status: Fixed (Revised commit: a277a39)

I06. Default State Variable Visibility

The lack of variable visibility may cause unexpected variable visibility in derived contracts.

Path: ./contracts/BaseMarketplaceContract.sol : CODE_VALIDITY

Recommendation: Specify the needed visibility during the variable initialization.

Found in: ab09639

Status: Fixed (Revised commit: 19c054c)

I07. Redundant Conditional Statement

The *BaseMarketplaceContract* contract has the function *adminModifyVIP180* which allows to whitelist tokens which might be accepted within the marketplace, but it is required to transfer one token to the contract before whitelisting it. These tokens might not be retrieved from the contract.

Path: ./contracts/BaseMarketplaceContract.sol : adminModifyVIP180()

Recommendation: Remove the conditional statement which requires to transfer *1* token to the contract before whitelisting.

Found in: ab09639

Status: Fixed (Revised commit: 9ce4e38)

I08. Redundant Structure Declaration

The *OfferContractVIP180* contract declares the *GlobalOfferDetails* structure, which is never used within the contract. This affects the code readability.

Path: ./contracts/OfferContractVIP180.sol : GlobalOfferDetails

Recommendation: Remove unused structure declaration to simplify the code readability, verify that this is not a part of unfinalized smart contract functionality.

Found in: ab09639

Status: Fixed (Revised commit: b2568fc)

I09. Redundant Structure Field

The *OfferContractVIP180* contract declares the *TokenOfferDetails* and *GlobalCollectionDetails* structure, which have the *seller* field that is never used within the contract. The field is only set to zero value, which does not affect the logic.

Path: ./contracts/OfferContractVIP180.sol : TokenOfferDetails, GlobalCollectionDetails

Recommendation: Either remove the unused structure field or make use of the *seller* field by assigning the correct value, instead of *address(0)*. This adjustment is necessary if the value needs to be accessed on the frontend or by third-party smart contract systems.

Found in: ab09639

Status: Fixed (Revised commit: 0ab450a)

I10. Use of uint256 Instead of Enums

The *OfferContractVIP180* contract has multiple structures with the *offerType* field; the field has 3 possible values such as: *0*, *1*, *3*.

This effect is code readability and supportability.

Path: ./contracts/OfferContractVIP180.sol : TokenOfferDetails.offerType, GlobalOfferDetails.offerType, GlobalCollectionDetails.offerType

Recommendation: Declare the *OfferType* enum and update the *offerType* field type to this Enum.

Found in: ab09639

Status: Fixed (Revised commit: a5084f9)

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Risk Level	High Impact	Medium Impact	Low Impact
High Likelihood	Critical	High	Medium
Medium Likelihood	High	Medium	Low
Low Likelihood	Medium	Low	Low

Risk Levels

Critical: Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.

High: High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.

Medium: Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.

Low: Major deviations from best practices or major Gas inefficiency. These issues won't have a significant impact on code execution, don’t affect security score but can affect code quality score.

Impact Levels

High Impact: Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

Medium Impact: Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

Low Impact: Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

Likelihood Levels

High Likelihood: Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

Medium Likelihood: Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

Low Likelihood: Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Initial review scope

Repository	https://github.com/vechainfoundation/nft-maas-sc
Commit	ab09639829a
Contracts	File: BaseMarketplaceContract.sol SHA3: c4f55d4c6c2609cd3d1e40270e4d3cc75fcafdd4db0e7623ac65c74debee8472 File: OfferContractVIP180.sol SHA3: 5ea7cba1d2893253101c1c6b79440d4e84d28808e77bfcc27ba4483d49dda1d3 File: WorldOfVBidAuction.sol SHA3: 68763861e56be43df41ad1d44682a6d0ad4d7cb3c1b561cb8630f62e9e1ae3cc File: WorldOfVFixedPriceNonCustodial.sol SHA3: 2f432b1b44a635ce3dbe85a4250c1552469b521fda09cb9202a9ca588496e41d