ABDK CONSULTING

SMART CONTRACT AUDIT

Gambit

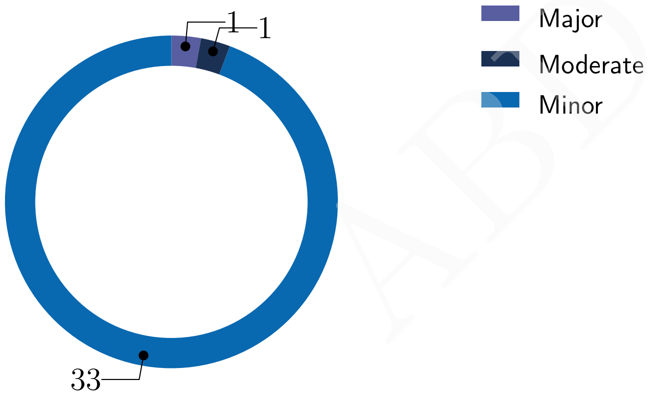
Orderbook

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
19th June 2021

We've been asked to review the Gambit Orderbook smart contracts given as a single file.



Findings

ID	Severity	Category	Status
CVF-1	Minor	Procedural	Opened
CVF-2	Minor	Readability	Opened
CVF-3	Minor	Bad datatype	Opened
CVF-4	Minor	Suboptimal	Opened
CVF-5	Minor	Bad datatype	Opened
CVF-6	Minor	Bad datatype	Opened
CVF-7	Minor	Bad datatype	Opened
CVF-8	Minor	Bad datatype	Opened
CVF-9	Minor	Suboptimal	Opened
CVF-10	Minor	Flaw	Opened
CVF-11	Minor	Suboptimal	Opened
CVF-12	Minor	Procedural	Opened
CVF-13	Minor	Flaw	Opened
CVF-14	Minor	Flaw	Opened
CVF-15	Minor	Readability	Opened
CVF-16	Minor	Suboptimal	Opened
CVF-17	Minor	Suboptimal	Opened
CVF-18	Minor	Unclear behavior	Opened
CVF-19	Minor	Unclear behavior	Opened
CVF-20	Minor	Flaw	Opened
CVF-21	Minor	Suboptimal	Opened
CVF-22	Minor	Suboptimal	Opened
CVF-23	Minor	Suboptimal	Opened
CVF-24	Major	Flaw	Opened
CVF-25	Minor	Flaw	Opened
CVF-26	Moderate	Flaw	Opened
CVF-27	Minor	Suboptimal	Opened

ID	Severity	Category	Status
CVF-28	Minor	Flaw	Opened
CVF-29	Minor	Unclear behavior	Opened
CVF-30	Minor	Readability	Opened
CVF-31	Minor	Suboptimal	Opened
CVF-32	Minor	Suboptimal	Opened
CVF-33	Minor	Readability	Opened
CVF-34	Minor	Suboptimal	Opened
CVF-35	Minor	Suboptimal	Opened

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	June 18, 2021	D. Khovratovich	Initial Draft
0.2	June 18, 2021	D. Khovratovich	Minor revision
1.0	June 19, 2021	D. Khovratovich	Release

Contact

D. Khovratovich

khovratovich@gmail.com

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting 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.

We have reviewed the contract in the Gambit repository, release v1.1:

- OrderBook.sol.

2.1 About ABDK

ABDK Consulting, 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.

2.2 Disclaimer

Note that the performed audit represents current best practices and smart contract standards which are relevant at the date of publication. After fixing the indicated issues the smart contracts should be re-audited.

2.3 Methodology

The methodology is not a strict formal procedure, but rather a collection of methods and tactics that combined differently and tuned for every particular project, depending on the project structure and and used technologies, as well as on what the client is expecting from the audit. In current audit we use:

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows code best 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 and that 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 is 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 that code actually does what it is supposed to do, that algorithms are optimal and correct, and that proper data types are used. We also check 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.

3 Detailed Results

3.1 CVF-1

- **Severity** Minor
- **Category** Procedural
- **Status** Opened
- **Source** OrderBook.sol

Recommendation Should be "Ô.6.0" according to a common best practice.

Listing 1:

```
olidity 0.6.12;
```

3.2 CVF-2

- **Severity** Minor
- **Category** Readability
- **Status** Opened
- **Source** OrderBook.sol

Recommendation These values could be specified as "1e30" and "1e18" respectively.

Listing 2:

```
int256 public constant PRICE_PRECISION = 10 ** 30;
int256 public constant USDG_PRECISION = 10 ** 18;
```

3.3 CVF-3

- **Severity** Minor
- **Category** Bad datatype
- **Status** Opened
- **Source** OrderBook.sol

Recommendation These fields should have type "IERC20" instead of just "address".

Listing 3:

```
ddress purchaseToken;

ddress collateralToken;
ddress indexToken;

ddress collateralToken;

ddress indexToken;

ddress[] path;
```

3.4 CVF-4

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** OrderBook.sol

Description These mappings would be redundant in case the orders would be stored in mappings of arrays rather than in mappings of mappings.

Listing 4:

```
apping (address => uint256) public increaseOrdersIndex;

apping (address => uint256) public decreaseOrdersIndex;

apping (address => uint256) public swapOrdersIndex;
```

3.5 CVF-5

- **Severity** Minor
- **Category** Bad datatype
- **Status** Opened
- **Source** OrderBook.sol

Recommendation This variable should have type "IWETH".

Listing 5:

```
ddress public weth;
```

3.6 CVF-6

- **Severity** Minor
- **Category** Bad datatype
- **Status** Opened
- **Source** OrderBook.sol

Recommendation This variable should have type "IRouter".

Listing 6:

```
ddress public router;
```

3.7 CVF-7

- **Severity** Minor
- **Category** Bad datatype
- **Status** Opened
- **Source** OrderBook.sol

Recommendation This variable should have type "IVault".

Listing 7:

```
ddress public vault;
```

3.8 CVF-8

- **Severity** Minor
- **Category** Bad datatype
- **Status** Opened
- **Source** OrderBook.sol

Recommendation These parameters should have type "IERC20".

Listing 8:

```
ddress purchaseToken,
ddress indexToken,
ddress purchaseToken,
ddress indexToken,
ddress purchaseToken,
ddress indexToken,
ddress collateralToken,
ddress indexToken,
ddress collateralToken,
ddress indexToken,
ddress collateralToken,
ddress indexToken,
```

3.9 CVF-9

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** OrderBook.sol

Description These events log some variables several times unchanged.
Recommendation Consider logging only the modified order parameters and its index.

Listing 9:

```
vent CancelIncreaseOrder(
vent ExecuteIncreaseOrder(
vent UpdateIncreaseOrder(
```

3.10 CVF-10

- **Severity** Minor
- **Category** Flaw
- **Status** Opened
- **Source** OrderBook.sol

Recommendation These functions should log some events.

Listing 10:

```
unction initialize(

unction setMinExecutionFee(uint256 _minExecutionFee) external onlyGov {

unction setMinPurchaseTokenAmountUsd(uint256 _minPurchaseTokenAmountUsd) external onlyGov {

unction setGov(address _gov) external onlyGov {
```

3.11 CVF-11

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** OrderBook.sol

Recommendation This parameter is redundant. It could be derived from the msg.value, _amountIn, _executionFee, and _path[0] values like this:
_shouldWrap = (msg.value == _amountIn + _executionFee && _path[0] == weth);

Listing 11:

```
ool _shouldWrap
```

3.12 CVF-12

- **Severity** Minor
- **Category** Procedural
- **Status** Opened
- **Source** OrderBook.sol

Recommendation These cheap checks should be done before the expensive '_transferInETH' call.

Listing 12:

```
equire(_path.length == 2 || _path.length == 3, "OrderBook: invalid _path.length");
equire(_path[0] != _path[_path.length - 1], "OrderBook: invalid _path");

equire(_executionFee > minExecutionFee, "OrderBook: insufficient execution fee");
```

3.13 CVF-13

- **Severity** Minor
- **Category** Flaw
- **Status** Opened
- **Source** OrderBook.sol

Recommendation Should be ">=".

Listing 13:

```
equire(_executionFee > minExecutionFee, "OrderBook: insufficient execution fee");

equire(_executionFee > minExecutionFee, "OrderBook: insufficient execution fee");

equire(_executionFee > minExecutionFee, "OrderBook: insufficient execution fee");
```

3.14 CVF-14

- **Severity** Minor
- **Category** Flaw
- **Status** Opened
- **Source** OrderBook.sol

Recommendation Probably '_amountIn' should be required to be non-zero in this case.

Listing 14:

```
equire(msg.value == _executionFee.add(_amountIn), "OrderBook: incorrect value transferred");
```

3.15 CVF-15

- **Severity** Minor
- **Category** Readability
- **Status** Opened
- **Source** OrderBook.sol

Recommendation If the "swapOrders" variable would be a mapping of arrays rather than a mapping of mappings, this code would be simpler and more efficient: uint256 _orderIndex = swapOrders[_account].push (SwapOrder ({ ... }) - 1;

Listing 15:

```
int256 _orderIndex = swapOrdersIndex[_account];
wapOrder memory order = SwapOrder(
    _account,
    _path,
    _amountIn,
    _minOut,
    _triggerRatio,
    _triggerAboveThreshold,
    _executionFee
;
wapOrdersIndex[_account] = _orderIndex.add(1);
wapOrders[_account][_orderIndex] = order;
```

3.16 CVF-16

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** OrderBook.sol

Recommendation Reading these values from memory is suboptimal, as they all are available on stack as local variables.

Listing 16:

```
rder.path,
rder.amountIn,
rder.minOut,
rder.triggerRatio,
rder.triggerAboveThreshold,
rder.executionFee
```

3.17 CVF-17

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** OrderBook.sol

Description The expression 'swapOrders[msg.sender][_orderIndex]' is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 17:

```
wapOrder memory order = swapOrders[msg.sender][_orderIndex];

elete swapOrders[msg.sender][_orderIndex];
```

3.18 CVF-18

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Opened
- **Source** OrderBook.sol

Description The main purpose of the decimals property is to render token amounts in a human-friendly way. Using decimals in calculations is discouraged as not all tokens have them, and some tokens may have very little or very many decimals, which could break the business logic.

Listing 18:

```
int256 otherTokenDecimals = IVault(vault).tokenDecimals(
    ↪ _otherToken);
```

3.19 CVF-19

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Opened
- **Source** OrderBook.sol

Description This function always returns the same value. Is this function really needed?

Listing 19:

```
unction getUsdgMaxPrice() public pure returns (uint256) {
```

3.20 CVF-20

- **Severity** Minor
- **Category** Flaw
- **Status** Opened
- **Source** OrderBook.sol

Description There is no range check for the length of the "_path" argument, while only paths of 2 or 3 elements are valid.
Recommendation Consider adding an explicit check.

Listing 20:

```
ddress[] memory _path,
```

3.21 CVF-21

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** OrderBook.sol

Description This parameter is redundant. It makes the code more complicated and less efficient. The called may alway require the returned value to be true.

Listing 21:

```
ool _raise
```

3.22 CVF-22

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** OrderBook.sol

Description These parameters are redundant as they have not changed

Listing 22:

```
sg.sender,

rder.path,
rder.amountIn,

rder.executionFee
```

3.23 CVF-23

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** OrderBook.sol

Description Reading this value from the storage is suboptimal, as the same value is available in a local variable.

Listing 23:

```
rder.minOut,
```

3.24 CVF-24

- **Severity** Major
- **Category** Flaw
- **Status** Opened
- **Source** OrderBook.sol

Description Return value is ignored

Listing 24:

```
alidateSwapOrderPriceWithTriggerAboveThreshold(
```

3.25 CVF-25

- **Severity** Minor
- **Category** Flaw
- **Status** Opened
- **Source** OrderBook.sol

Description There should be some way for the user to get WETH rather than ether.
Recommendation Consider accepting an additional boolean argument telling whether WETH should be unwrapped or not.

Listing 25:

```
transferOutETH(_amountOut, payable(order.account));
```

3.26 CVF-26

- **Severity** Moderate
- **Category** Flaw
- **Status** Opened
- **Source** OrderBook.sol

Description The recipient of the ether may reject the transfer causing the whole swap to be reverted, so gas will be spent, but the fee receiver will get nothing.
Recommendation Consider sending WETH in case ether transfer fails.

Listing 26:

```
transferOutETH(_amountOut, payable(order.account));

transferOutETH(amountOut, payable(order.account));
```

3.27 CVF-27

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** OrderBook.sol

Description These parameters are redundant since they have been already logged.

Listing 27:

```
rder.path,
rder.amountIn,
rder.minOut,
rder.triggerRatio,
rder.triggerAboveThreshold,
rder.executionFee,
```

3.28 CVF-28

- **Severity** Minor
- **Category** Flaw
- **Status** Opened
- **Source** OrderBook.sol

Recommendation Should be “_maximizePrice“.

Listing 28:

```
ool _maximisePrice,
```

3.29 CVF-29

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Opened
- **Source** OrderBook.sol

Recommendation Probably should be '>='

Listing 29:

```
equire(_purchaseTokenAmountUsd > minPurchaseTokenAmountUsd, "
    ↪ OrderBook: insufficient collateral");
```

3.30 CVF-30

- **Severity** Minor
- **Category** Readability
- **Status** Opened
- **Source** OrderBook.sol

Recommendation If the "increaseOrders" variable would be a mapping of arrays rather than a mapping of mappings, this code would be simpler and more efficient: uint256 _orderIndex = increaseOrders[_account].push (IncreaseOrder ({ ... }) - 1;

Listing 30:

```
int256 _orderIndex = increaseOrdersIndex[msg.sender];
ncreaseOrder memory order = IncreaseOrder(
    _account,
    _purchaseToken,
    _purchaseTokenAmount,
    _collateralToken,
    _indexToken,
    _sizeDelta,
    _isLong,
    _triggerPrice,
    _triggerAboveThreshold,
    _executionFee
;
ncreaseOrdersIndex[_account] = _orderIndex.add(1);
ncreaseOrders[_account][_orderIndex] = order;
```

3.31 CVF-31

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** OrderBook.sol

Description The expression “increseOrders[msg.sender][_orderIndex]” is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 31:

```
ncreaseOrder memory order = increaseOrders[msg.sender][_orderIndex];
```

```
elete increaseOrders[msg.sender][_orderIndex];
```

3.32 CVF-32

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** OrderBook.sol

Description This parameter is redundant, as it could be derived from msg.value.

Listing 32:

```
int256 _executionFee
```

3.33 CVF-33

- **Severity** Minor
- **Category** Readability
- **Status** Opened
- **Source** OrderBook.sol

Recommendation If the “decreaseOrders” variable would be a mapping of arrays rather than a mapping of mappings, this code would be simpler and more efficient: uint256 _orderIndex = decreaseOrders[_account].push (DecreaseOrder ({ ... }) - 1;

Listing 33:

```
int256 _orderIndex = decreaseOrdersIndex[_account];
ecreaseOrder memory order = DecreaseOrder(
    _account,
    _collateralToken,
    _collateralDelta,
    _indexToken,
    _sizeDelta,
    _isLong,
    _triggerPrice,
    _triggerAboveThreshold,
    _executionFee
;
ecreaseOrdersIndex[_account] = _orderIndex.add(1);
ecreaseOrders[_account][_orderIndex] = order;
```

3.34 CVF-34

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** OrderBook.sol

Description This should be done only when ’msg.value != 0’.

Listing 34:

```
WETH(weth).deposit{value: msg.value}();
```

3.35 CVF-35

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** OrderBook.sol

Description This transfer is redundant.
Recommendation Just redirect the swap output to the vault in the previous line.

Listing 35:

```
ERC20(_path[1]).safeTransfer(vault, midOut);
```