ABDK CONSULTING

SMART CONTRACT AUDIT

Conduit

Framework

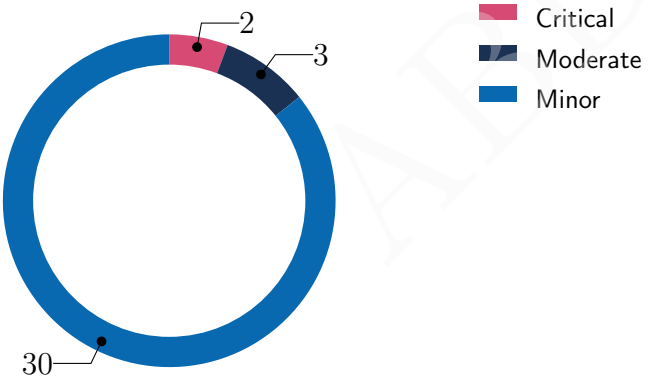
Solidity

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
6th July 2021

We've been asked to review the Framework Conduit smart contracts given in a private repo.
We have found several major issues as well as a number of ones of lesser importance.



Findings

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

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

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	July 5, 2021	D. Khovratovich and M. Vladimirov	Initial Draft
0.2	July 6, 2021	D. Khovratovich	Minor revision
1.0	July 6, 2021	D. Khovratovich and M. Vladimirov	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 audited the Conduit smart contracts at release v0.1.

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
- **Status** Opened
- **Category** Procedural
- **Source** trading-wallet.sol, verifier.sol, whitelist-contract.sol, whitelist-address.sol

Description Should be "ô.7.0" as Solidity 0.8.x contains a number of non-backward compatible changes.

Listing 1:

```
1 solidity >=0.7.0;
```

3.2 CVF-2

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** trading-wallet.sol

Recommendation The file should be named "TradingWallet.sol" according to a common best practice.

Listing 2:

```
14 TradingWallet is Verifier, WhitelistAddress, WhitelistContract {
```

3.3 CVF-3

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

Recommendation The "pair" argument should have type "IUniswapV2Pair".

Listing 3:

```
21 address pair_,

33 address pair_,
```

3.4 CVF-4

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** trading-wallet.sol

Description It is unclear which number format is used for fee.
Recommendation Consider explaining in documentation comment.

Listing 4:

```
22 uint256 fee, // should be 9970 for uniswap and 9975 for
     ↪ pancakeswap
```

3.5 CVF-5

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** trading-wallet.sol

Description The "verified" modifier only verifies that the message sender is eligible to execute some transaction on behalf of the owner, but doesn't verify the transaction parameters.
Recommendation Consider including the hash of 'msg.data' in the message being verified.

Listing 5:

```
28 ) external onlyWhitelistAddress onlyWhitelistContract(pair_)
     ↪ verified(v, r, s, owner(), msg.sender) whenNotPaused {
```

3.6 CVF-6

- **Severity** Moderate
- **Status** Opened
- **Category** Flaw
- **Source** trading-wallet.sol

Recommendation It should be the "computeAverageOrderSize" function to be called when the "average" argument is true, and the "computeLimitOrderSize" function to be called otherwise.

Listing 6:

```
62 if (average) {
     amountIn = computeLimitOrderSize(

67 } else {
     amountIn = computeAverageOrderSize(
```

3.7 CVF-7

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

Recommendation There should be a named constant for the fee denominator.

Listing 7:

```
76   amountIn = FullMath.mulDiv(amountIn, 10000, fee);

131 amountIn = amountIn.mul(10000).div(fee);
```

3.8 CVF-8

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** trading-wallet.sol

Description Using different denominators for prices and fees makes code harder to read.
Recommendation Consider using the same denominator (e.g. 1e18) everywhere.

Listing 8:

```
76   amountIn = FullMath.mulDiv(amountIn, 10000, fee);

131 amountIn = amountIn.mul(10000).div(fee);
```

3.9 CVF-9

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

Description The condition is a bit asymmetric: "<" when "aToB" is true and ">=" when "aToB" is false. Probably not an issue.

Listing 9:

```
95 require(FullMath.mulDiv(reserveA, 1 ether, reserveB) <
     ↪ priceAsRatioAtoB == aToB, 'ELO5');
```

3.10 CVF-10

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** trading-wallet.sol

Description Using "1 ether" for price denominator is confusing as a price is not an ether amount.

Listing 10:

```
95  require(FullMath.mulDiv(reserveA, 1 ether, reserveB) <
    ↪ priceAsRatioAtoB == aToB, 'ELO5');

97  uint256 leftSide  = FullMath.mulDiv((aToB ? reserveB : reserveA)
    ↪ , (aToB ? priceAsRatioAtoB : 1 ether), (aToB ? 1 ether :
    ↪ priceAsRatioAtoB));

114 require(FullMath.mulDiv(reserveA, 1 ether, reserveB) <
    ↪ limitPriceAsRatioAtoB == aToB, 'ELO6');

121         aToB ? limitPriceAsRatioAtoB : 1 ether,
            (aToB ? 1 ether : limitPriceAsRatioAtoB)
```

3.11 CVF-11

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

Recommendation The file should be named "Verifier.sol" according to a common best practice.

Listing 11:

```
9  contract Verifier is Ownable, Pausable {
```

3.12 CVF-12

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** verifier.sol

Recommendation Name is confusing. It should be about the message hash.

Listing 12:

```
25 function _signMessage(address sender) internal view returns (
   ↪ bytes32){
```

3.13 CVF-13

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

Description A hash generated here could be reused with other instances of the smart contract and on other blockchains.
Recommendation Consider hashing in the address of the smart contract and the chain ID.

Listing 13:

```
26 bytes32 messageHash = keccak256(abi.encodePacked(sender,
   ↪ _sequenceNumbers[sender]));
```

3.14 CVF-14

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

Recommendation As the amount of data hashed in the previous line is constant, double hashing is redundant. Just concatenate the data with ethereum signed prefix and hash it all at once.

Listing 14:

```
27 bytes32 ethSignedMessageHash = keccak256(abi.encodePacked("\
   ↪ x19Ethereum Signed Message:\n32", messageHash));
```

3.15 CVF-15

- **Severity** Critical
- **Status** Opened
- **Category** Flaw
- **Source** verifier.sol

Description The "ecrecover" function silently returns zero address on invalid signature. Thus, the current code allows signing anything on behalf of zero address. The contract should revert in case "ecrecover" returned zero address.

Listing 15:

```
32 return ecrecover(_ethSignedMessageHash, v, r, s);
```

3.16 CVF-16

- **Severity** Moderate
- **Status** Opened
- **Category** Unclear behavior
- **Source** verifier.sol

Description The 'msg.data' value is not hashed, so by signing a message, signer allows the sender to execute arbitrary function with arbitrary arguments. This looks weird.

Listing 16:

```
35 /* Modifier makes sure that the signer signed the message [
     ↪ sender, _sequenceNumbers[sender]] before executing
     ↪ function */
```

3.17 CVF-17

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

Description Modifying state in function modifiers makes the code less readable.
Recommendation Consider putting this operation into the modified function.

Listing 17:

```
38 _sequenceNumbers[msg.sender]++;
```

3.18 CVF-18

- **Severity** Critical
- **Status** Opened
- **Category** Flaw
- **Source** verifier.sol

Description The "verify" function hashes "_sequenceNumbers[sender]", while here "_sequenceNumbers[msg.sender]" value is incremented. This will not work correctly in case "sender" is not the same as "msg.sender".

Listing 18:

```
38 _sequenceNumbers[msg.sender]++;
```

3.19 CVF-19

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

Description Probably this code should be executed only if the verification is turned on.

Listing 19:

```
38 _sequenceNumbers[msg.sender]++;
```

3.20 CVF-20

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

Recommendation This function wouldn't be necessary if the "_sequenceNumbers" mapping was be public.

Listing 20:

```
42 function sequenceNumber(address addr) public view virtual
   ↪ returns (uint256) { return _sequenceNumbers[addr]; }
```

3.21 CVF-21

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

Description This function brings the contract into a stage when anybody may execute any verified function on behalf of any signed. This is weird. What are the scenarios for this function?

Listing 21:

```
44 function stopVerifying() public onlyOwner { _verifying = false;
   ↪ } // stop verifying that message is correctly signed
```

3.22 CVF-22

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

Recommendation These functions should log come events.

Listing 22:

```
44 function stopVerifying() public onlyOwner { _verifying = false;
   ↪ } // stop verifying that message is correctly signed

46 function startVerifying() public onlyOwner { _verifying = true;
   ↪ } // resume verifying that message is correctly signed
```

3.23 CVF-23

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

Description The contract's behavior doesn't depend on whether it is paused or not. Probably it shouldn't inherit from Pausable.

Listing 23:

```
48 function pause() public onlyOwner { _pause(); }

50 function unpause() public onlyOwner { _unpause(); }
```

3.24 CVF-24

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

Description These function look unrelated to verification.
Recommendation Consider moving them to some other place.

Listing 24:

```
53 function withdrawToken(uint256 amount, address token) external
   ↪ onlyOwner {

58 function withdrawEth(uint256 amount) external onlyOwner {

62 receive() external payable {}
```

3.25 CVF-25

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

Recommendation The "token" argument should have type "IERC20".

Listing 25:

```
53 function withdrawToken(uint256 amount, address token) external
   ↪ onlyOwner {
```

3.26 CVF-26

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** whitelist-contract.sol

Description This import is not used.

Listing 26:

```
3 './verifier.sol';
```

3.27 CVF-27

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** whitelist-contract.sol

Recommendation The file should be named "WhitelistContract.sol" according to a common best practice.

Listing 27:

```
7 contract WhitelistContract is Ownable {
```

3.28 CVF-28

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** whitelist-contract.sol

Recommendation These functions probably should log some event.

Listing 28:

```
19 function whitelistContract(address addr) public onlyOwner {
     ↪ _contractWhitelist[addr] = true; }

21 function removeFromContractWhitelist(address addr) public
     ↪ onlyOwner {
```

3.29 CVF-29

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

Description There is not similar check in the "whitelistContract" function.
Recommendation Consider adding such there for consistency.

Listing 29:

```
22 require(_contractWhitelist[addr], 'contract address not in
     ↪ contract whitelist');
```

3.30 CVF-30

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** whitelist-contract.sol

Recommendation This function wouldn't be necessary in case the "_contractWhitelist" mapping was public.

Listing 30:

```
26 function contractWhitelist(address addr) public view virtual
     ↪ returns (bool) { return _contractWhitelist[addr]; }
```

3.31 CVF-31

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** whitelist-address.sol

Recommendation This import is not used.

Listing 31:

```
3 "./verifier.sol";
```

3.32 CVF-32

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** whitelist-address.sol

Recommendation The file should be named "WhitelistAddress.sol" according to a common best practice.

Listing 32:

```
6 contract WhitelistAddress is Ownable {
```

3.33 CVF-33

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** whitelist-address.sol

Recommendation These functions should probably log some event.

Listing 33:

```
18 function whitelistAddress(address addr) public onlyOwner {
     ↪ _whitelist[addr] = true; }

20 function removeFromWhitelist(address addr) public onlyOwner {
```

3.34 CVF-34

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

Description The “whitelistAddress“ function doesn’t have similar check.
Recommendation Consider adding it there for consistency.

Listing 34:

```
21 require(_whitelist[addr], 'address not in whitelist');
```

3.35 CVF-35

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** whitelist-address.sol

Recommendation This function won’t be needed if ’_whitelist’ array was public.

Listing 35:

```
25 function whitelist(address addr) public view virtual returns (
   ↪ bool) { return _whitelist[addr]; }
```