ABDK CONSULTING

SMART CONTRACT AUDIT

1inch

AggregationRouter

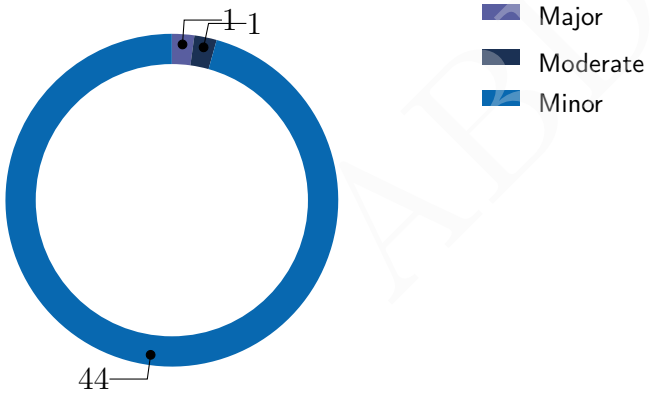
Solidity

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
4th October 2021

We've been asked to review the 7 files in a GitHub repo. We found 1 major, and a few less important issues.



Findings

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

ID	Severity	Category	Status
CVF-28	Minor	Unclear behavior	Fixed
CVF-29	Minor	Unclear behavior	Fixed
CVF-30	Minor	Bad datatype	Fixed
CVF-31	Minor	Procedural	Info
CVF-32	Minor	Bad naming	Fixed
CVF-33	Moderate	Flaw	Info
CVF-34	Minor	Readability	Info
CVF-35	Minor	Flaw	Info
CVF-36	Minor	Unclear behavior	Info
CVF-37	Minor	Procedural	Fixed
CVF-38	Minor	Suboptimal	Fixed
CVF-39	Minor	Suboptimal	Fixed
CVF-40	Minor	Procedural	Info
CVF-41	Minor	Suboptimal	Fixed
CVF-42	Minor	Suboptimal	Fixed
CVF-43	Minor	Suboptimal	Info
CVF-44	Minor	Overflow/Underflow	Info
CVF-45	Minor	Flaw	Info
CVF-46	Minor	Suboptimal	Info

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	October 3, 2021	D. Khovratovich	Initial Draft
0.2	October 4, 2021	D. Khovratovich	Minor revision
1.0	October 4, 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 contracts at commit 93868c:

- UnoswapV3Router.sol
- AggregationRouterV4.sol
- XLimitOrderProtocolRFQ.sol
- ClipperRouter.sol
- ArgumentsDecoder.sol
- Permitable.sol
- RevertReasonParser.sol

The fixes were applied in commit af3e7e2.

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** Info
- **Category** Procedural
- **Source** UnoswapV3Router.sol, AggregationRouterV4.sol, ClipperRouter.sol, ArgumentsDecoder.sol, EthReceiver.sol, Permitable.sol, RevertReasonParser.sol, UniERC20.sol.

Recommendation Should be "0̂.7.0" according to a common best practice unless there is something special about this particular version.
Client Comment Won't fix.

Listing 1:

```
3 solidity ^0.7.6;
```

3.2 CVF-2

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** UnoswapV3Router.sol

Description Hardcoding mainnet addresses is discouraged.
Recommendation Consider turning this constant into an immutable variable and passing its value as a constructor argument.
Client Comment Won't fix.

Listing 2:

```
28 IWETH private constant _WETH = IWETH(0
    ↪ xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2);
```

3.3 CVF-3

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** UnoswapV3Router.sol

Recommendation A better name would be "uniswapV3SwapToWithPermit" as this function accepts the recipient address as an argument.

Listing 3:

```
30 function uniswapV3SwapWithPermit(
```

3.4 CVF-4

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** UnoswapV3Router.sol

Recommendation This spells "recipient".

Listing 4:

```
51 address payable recepient,
```

3.5 CVF-5

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** UnoswapV3Router.sol

Description This calcualtes "i + 1" on every iteration.
Recommendation Consider calcualating "len - 1" once before the loop, saving in a local variable and using as the loop boundary. The same value would be useful after the loop as the index of the last pool, thus, it would be possible to declare the "i" variable inside the "for" statement brackets.

Listing 5:

```
68 for (; i + 1 < len; i++) {
```

3.6 CVF-6

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** UnoswapV3Router.sol

Recommendation It would be cheaper to pass the token addresses as parts of the "calldata" argument, rather than obtaining them through call to the UniswapV3Pool contract.
Client Comment Won't fix, as it requires too much changes in UniswapV3Router.

Listing 6:

```
111 if iszero(staticcall(gas(), caller(), emptyPtr, 0x4, resultPtr,
    ↪ 0x20)) {

115 if iszero(staticcall(gas(), caller(), add(emptyPtr, 0x4), 0x4,
    ↪ resultPtr, 0x20)) {
```

3.7 CVF-7

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** UnoswapV3Router.sol

Recommendation This should be a named constant.
Client Comment Won't fix.

Listing 7:

```
137 revertWithReason(0
    ↪ x00000010554e495633523a2062616420706f6f6c0000000000000000000000000
    ↪ , 0x54)  // UNIV3R: bad pool
```

3.8 CVF-8

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** UnoswapV3Router.sol

Recommendation This could be done after the conditional operation. Just define an "other-Amount" variable before the conditional statement, assign it differently in the "then" and "else" branched, and then use it in a return statement after the conditional operator.
Client Comment Won't fix.

Listing 8:

```
170 return uint256(-amount1);

179 return uint256(-amount0);
```

3.9 CVF-9

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** AggregationRouterV4.sol

Recommendation If these are bitmasks a shift like '1«1' would be more readable.

Listing 9:

```
21 uint256 private constant _PARTIAL_FILL = 0x01;
   uint256 private constant _REQUIRES_EXTRA_ETH = 0x02;
   uint256 private constant _SHOULD_CLAIM = 0x04;
```

3.10 CVF-10

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** AggregationRouterV4.sol

Recommendation If these constants are not used they should be removed.
Client Comment They'll be used on ETH forks that still support gastokens.

Listing 10:

```
24 // uint256 private constant _BURN_FROM_MSG_SENDER = 0x08;
   // uint256 private constant _BURN_FROM_TX_ORIGIN = 0x10;
```

3.11 CVF-11

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** AggregationRouterV4.sol

Recommendation Some parameters of this event should be indexed.
Client Comment Won't fix due to extra gas costs.

Listing 11:

```
38 event Swapped(
```

3.12 CVF-12

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** AggregationRouterV4.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.
Client Comment Self-explanatory.

Listing 12:

```
48 constructor(IClipperExchangeInterface _clipperExchange)
   ↪ ClipperRouter(_clipperExchange) {}
```

3.13 CVF-13

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** AggregationRouterV4.sol

Recommendation This commented out function should be removed.
Client Comment It'll be used on ETH forks that still support gastokens.

Listing 13:

```
50 // function discountedSwap(
```

3.14 CVF-14

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** AggregationRouterV4.sol

Description The "gasLeft" return value looks redundant. It is a legacy of some benchmarking efforts?
Client Comment It is used to accurately estimate spent gas after refunds.

Listing 14:

```
95 returns (uint256 returnAmount, uint256 gasLeft)
```

3.15 CVF-15

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** AggregationRouterV4.sol

Recommendation As "data" is a bytes array, a more appropriate message would be "data should be not empty".

Listing 15:

```
98 require(data.length > 0, "data should be not zero");
```

3.16 CVF-16

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** AggregationRouterV4.sol

Description This assignment should be done only if flags & _PARTIAL_FILL is not zero.

Listing 16:

```
117 uint256 initialSrcBalance = (flags & _PARTIAL_FILL != 0) ?
    ↪ srcToken.uniBalanceOf(msg.sender) : 0;
```

3.17 CVF-17

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** AggregationRouterV4.sol

Description The "initialSrcBalance" here is actually not an "initial" one, as source tokens are already sent to the "desc.srcReceiver" address at this point. This is confusing.
Recommendation Consider renaming to "balanceBeforeRefund", as this balance is used to calculate the refund amount in case of partial fill.

Listing 17:

```
117 uint256 initialSrcBalance = (flags & _PARTIAL_FILL != 0) ?
    ↪ srcToken.uniBalanceOf(msg.sender) : 0;
```

3.18 CVF-18

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** AggregationRouterV4.sol

Description This assignment should be made only when flags & _PARTIAL_FILL is zero.

Listing 18:

```
129 uint256 spentAmount = desc.amount;
```

3.19 CVF-19

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** AggregationRouterV4.sol

Description This formula is confusing, as one could expect that the "initialSrcBalance >= srcToken.uniBalanceOf(msg.sender)", while this is not the case.
Recommendation Consider rewriting like this: 'spentAmount = desc.amount.sub (srcToken.uniBalanceOf (msg.sender).sub (initialSrcBalance));'.

Listing 19:

```
133 spentAmount = initialSrcBalance.add(desc.amount).sub(srcToken.
    ↪ uniBalanceOf(msg.sender));
```

3.20 CVF-20

- **Severity** Minor
- **Status** Info
- **Category** Overflow/Underflow
- **Source** AggregationRouterV4.sol

Description Phantom overflow is possible here.
Recommendation Consider using a full 512 bits multiplication which could be done quite cheaply as described here: https://medium.com/wicketh/mathemagic-full-multiply-27650fec525d.
Client Comment Noted.

Listing 20:

```
134 require(returnAmount.mul(desc.amount) >= desc.minReturnAmount.
    ↪ mul(spentAmount), "Return amount is not enough");
```

3.21 CVF-21

- **Severity** Major
- **Status** Info
- **Category** Suboptimal
- **Source** AggregationRouterV4.sol

Description Self-destructing the contract may affect other contracts that rely on it.
Recommendation Consider removing the self-destruct logic, as it causes more harm than benefits.
Client Comment It'll be only used if an issue will be found that allows to drain users' approvals.

Listing 21:

```
156 selfdestruct(msg.sender);
```

3.22 CVF-22

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** LimitOrderProtocolRFQ.sol

Recommendation This parameter should be indexed.
Client Comment Won't fix due to extra gas costs.

Listing 22:

```
23 bytes32 orderHash,
```

3.23 CVF-23

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** LimitOrderProtocolRFQ.sol

Description Solidity compiler is smart enough to store several fields inside a single storage slot.
Recommendation Consider splitting this filed into two fields.
Client Comment However it is not smart enough to store it in a single calldata slot.

Listing 23:

```
28 uint256 info; // lowest 64 bits is the order id, next 64 bits
   ↪ is the expiration timestamp
```

3.24 CVF-24

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** LimitOrderProtocolRFQ.sol

Recommendation This function should probably log some event
Client Comment Won't fix.

Listing 24:

```
57 function cancelOrderRFQ(uint256 orderInfo) external {
```

3.25 CVF-25

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** LimitOrderProtocolRFQ.sol

Description The semantics of the returned values is unclear.
Recommendation Consider explaining in the documentation comment.

Listing 25:

```
71 ) external returns(uint256, uint256) {

82 ) external returns(uint256, uint256) {

93 ) public returns(uint256, uint256) {
```

3.26 CVF-26

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** LimitOrderProtocolRFQ.sol

Description The expression "_invalidator[maker]" is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 26:

```
103 uint256 invalidator = _invalidator[maker][invalidatorSlot];

105 _invalidator[maker][invalidatorSlot] = invalidator |
    ↪ invalidatorBit;
```

3.27 CVF-27

- **Severity** Minor
- **Status** Info
- **Category** Overflow/Underflow
- **Source** LimitOrderProtocolRFQ.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 calculations overflow.
Recommendation Consider using muldiv function as described here: https://2\begingroup\let\relax\relax\endgroup[Pleaseinsert\PrerenderUnicode{ĬĂ}intopreamble].com/21/muldiv/index.html or some other safe approach.
Client Comment Noted.

Listing 27:

```
117 takingAmount = (makingAmount.mul(orderTakerAmount).add(
    ↪ orderMakerAmount).sub(1)).div(orderMakerAmount);

120 makingAmount = takingAmount.mul(orderMakerAmount).div(
    ↪ orderTakerAmount);
```

3.28 CVF-28

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** LimitOrderProtocolRFQ.sol

Description This check should be done inside the "if (takingAmount == 0)" branch.

Listing 28:

```
127 require(makingAmount <= orderMakerAmount, "LOP: making amount
    ↪ exceeded");
```

3.29 CVF-29

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** LimitOrderProtocolRFQ.sol

Description This check should be done inside the "if (makingAmount == 0)" branch.

Listing 29:

```
128 require(takingAmount <= orderTakerAmount, "LOP: taking amount
    ↪ exceeded");
```

3.30 CVF-30

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** LimitOrderProtocolRFQ.sol

Recommendation 100 should be a named constant.

Listing 30:

```
163 require(makerAssetData.length >= 100, "LOP: bad makerAssetData.
    ↪ length");
    require(takerAssetData.length >= 100, "LOP: bad takerAssetData.
    ↪ length");
```

3.31 CVF-31

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** ClipperRouter.sol

Description Hardcoding mainnet addresses is discouraged.
Recommendation Consider turning this constant into an immutable variable and passing its value as a constructor argument.
Client Comment Won't fix.

Listing 31:

```
15 IWETH private constant _WETH = IWETH(0
    ↪ xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2);
```

3.32 CVF-32

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** ClipperRouter.sol

Recommendation A better name would be "clipperSwapToWithPermit" as this function accepts the recipient as an argument.

Listing 32:

```
28 function clipperSwapWithPermit(
```

3.33 CVF-33

- **Severity** Moderate
- **Status** Info
- **Category** Flaw
- **Source** ClipperRouter.sol

Description The returned values are ignored.
Client Comment Because it either returns true or reverts.

Listing 33:

```
59 _WETH.transferFrom(msg.sender, address(this), amount);

78 _WETH.transfer(recipient, returnAmount);
```

3.34 CVF-34

- **Severity** Minor
- **Status** Info
- **Category** Readability
- **Source** ClipperRouter.sol

Recommendation The 'srcETH' should be set false in this clause for readability.
Client Comment Won't fix.

Listing 34:

```
67 else {
```

3.35 CVF-35

- **Severity** Minor
- **Status** Info
- **Category** Flaw
- **Source** ClipperRouter.sol

Description The case when both, source and destination tokens are either ether or WETH is not handled.
Recommendation Consider explicitly forbidding such case.
Client Comment It is implicitly forbidden because Clipper does not handle WETH.

Listing 35:

```
74 returnAmount = _clipperExchange.sellEthForToken(dstToken,
   ↪ recipient, minReturn, _INCH_TAG);
```

3.36 CVF-36

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** ClipperRouter.sol

Description Sending the same constant in all calls looks odd. Is it really necessary?
Client Comment Yep, that's for accounting.

Listing 36:

```
74 returnAmount = _clipperExchange.sellEthForToken(dstToken,
     ↪ recipient, minReturn, _INCH_TAG);

76 returnAmount = _clipperExchange.sellTokenForEth(srcToken,
     ↪ address(this), minReturn, _INCH_TAG);

80 returnAmount = _clipperExchange.sellTokenForEth(srcToken,
     ↪ recipient, minReturn, _INCH_TAG);

82 returnAmount = _clipperExchange.sellTokenForToken(srcToken,
     ↪ dstToken, recipient, minReturn, _INCH_TAG);
```

3.37 CVF-37

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** ArgumentsDecoder.sol

Description These functions don't perform any length checks on the "data" argument.
Recommendation Consider adding such checks.

Listing 37:

```
 7 function decodeSelector(bytes memory data) internal pure returns
     ↪ (bytes4 selector) {

13 function decodeAddress(bytes memory data, uint256 argumentIndex)
     ↪  internal pure returns(address account) {

19 function decodeUint256(bytes memory data, uint256 argumentIndex)
     ↪  internal pure returns(uint256 value) {

25 function decodeTargetAndCalldata(bytes calldata data) internal
     ↪ pure returns(address target, bytes calldata args) {

32 function patchAddress(bytes memory data, uint256 argumentIndex,
     ↪ address account) internal pure {

38 function patchUint256(bytes memory data, uint256 argumentIndex,
     ↪ uint256 value) internal pure {
```

3.38 CVF-38

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** ArgumentsDecoder.sol

Recommendation It would be more efficient to accept an argument offset instead of an argument index, as in most cases such offset could be precomputed at compile time.

Listing 38:

```
13 function decodeAddress(bytes memory data, uint256 argumentIndex)
   ↪ internal pure returns(address account) {

19 function decodeUint256(bytes memory data, uint256 argumentIndex)
   ↪ internal pure returns(uint256 value) {

32 function patchAddress(bytes memory data, uint256 argumentIndex,
   ↪ address account) internal pure {

38 function patchUint256(bytes memory data, uint256 argumentIndex,
   ↪ uint256 value) internal pure {
```

3.39 CVF-39

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** ArgumentsDecoder.sol

Recommendation It would be cheaper to use left shift instead of multiplication.

Listing 39:

```
15 account := mload(add(add(data, 0x24), mul(argumentIndex, 0x20)))

21 value := mload(add(add(data, 0x24), mul(argumentIndex, 0x20)))

34 mstore(add(add(data, 0x24), mul(argumentIndex, 0x20)), account)

40 mstore(add(add(data, 0x24), mul(argumentIndex, 0x20)), value)
```

3.40 CVF-40

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** Permitable.sol

Description In case 'permit.length' is zero, the function silently does nothing and thus doesn't guarantee that the allowance after the call will not be less that the "amount" value.
Recommendation Consider reverting in such a case.
Client Comment Won't fix.

Listing 40:

```
16 if (permit.length > 0) {
```

3.41 CVF-41

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Permitable.sol

Description The reason encoded in the former line will be parsed back in the latter line.
Recommendation Consider refactoring.

Listing 41:

```
26 result = abi.encodeWithSignature("Error(string)", "Wrong permit
   ↪ length");

29 string memory reason = RevertReasonParser.parse(result, "Permit
   ↪ call failed: ");
```

3.42 CVF-42

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RevertReasonParser.sol

Recommendation It would be more efficient to extract the selector from "data" as a "bytes4" value and then compare it to know selector, rather than compare bytes one by one.

Listing 42:

```
12 if (data.length >= 68 && data[0] == "\x08" && data[1] == "\xc3"
   ↪ && data[2] == "\x79" && data[3] == "\xa0") {

30 else if (data.length == 36 && data[0] == "\x4e" && data[1] == "\
   ↪ x48" && data[2] == "\x7b" && data[3] == "\x71") {
```

3.43 CVF-43

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** RevertReasonParser.sol

Recommendation It is possible to use the "abi.decode" function to parse the ABI incoded arguments of "Error" and "Panic". Just chop the first four bytes from "data". Note, that it is possible to chop bytes from the beginning of an array in-place, without copying the whole array.
Client Comment Won't fix. The 'abi.decode' is much more gas hungry than simple assembly. Also chopping in-place is possible only for calldata types.

Listing 43:

```
12 if (data.length >= 68 && data[0] == "\x08" && data[1] == "\xc3"
   ↪ && data[2] == "\x79" && data[3] == "\xa0") {

30 else if (data.length == 36 && data[0] == "\x4e" && data[1] == "\
   ↪ x48" && data[2] == "\x7b" && data[3] == "\x71") {
```

3.44 CVF-44

- **Severity** Minor
- **Status** Info
- **Category** Overflow/Underflow
- **Source** RevertReasonParser.sol

Description Overflow is possible here.
Client Comment Won't fix.

Listing 44:

```
26 require(data.length >= 68 + bytes(reason).length, "Invalid
   ↪ revert reason");
```

3.45 CVF-45

- **Severity** Minor
- **Status** Info
- **Category** Flaw
- **Source** RevertReasonParser.sol

Recommendation Should be "else return".
Client Comment Won't fix.

Listing 45:

```
40 return string(abi.encodePacked(prefix, "Unknown(", _toHex(data),
   ↪ ")"));
```

3.46 CVF-46

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** RevertReasonParser.sol

Recommendation These functions could be significantly optimized. See the following gist for an idea how to do this: https://gist.github.com/3sGgpQ8H/7c00e48af22b91effd5c37adc44908f3
Client Comment Noted. We'll optimize this in the next version.

Listing 46:

```
43 function _toHex(uint256 value) private pure returns(string
   ↪ memory) {

47 function _toHex(bytes memory data) private pure returns(string
   ↪ memory) {
```