



ABDK CONSULTING

SMART CONTRACT
AUDIT

ZeroPool

Solidity



abdk.consulting

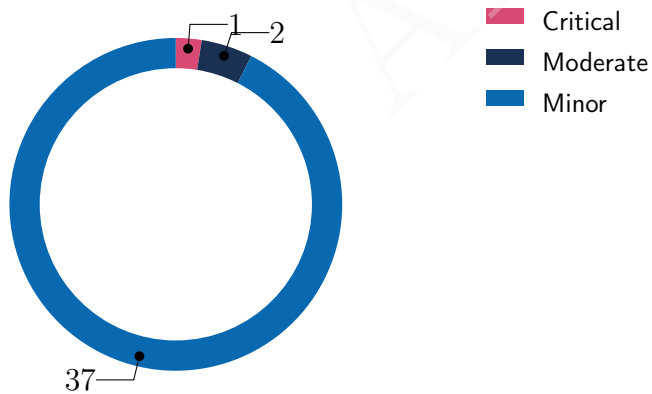
SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
31st January 2022

We've been asked to review the following files:

- [Parameters.sol](#)
- [Pool.sol](#)

We found 1 critical, and a few less important issues. The critical issue and a few other ones were fixed.



Findings

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

ID	Severity	Category	Status
CVF-28	Minor	Unclear behavior	Info
CVF-29	Minor	Documentation	Info
CVF-30	Minor	Suboptimal	Info
CVF-31	Minor	Overflow/Underflow	Info
CVF-32	Minor	Suboptimal	Info
CVF-33	Minor	Suboptimal	Info
CVF-34	Minor	Bad datatype	Info
CVF-35	Minor	Suboptimal	Info
CVF-36	Minor	Suboptimal	Info
CVF-37	Moderate	Flaw	Fixed
CVF-38	Minor	Flaw	Fixed
CVF-39	Critical	Flaw	Fixed
CVF-40	Minor	Suboptimal	Fixed

Contents

1	Document properties	7
2	Introduction	8
2.1	About ABDK	8
2.2	Disclaimer	8
2.3	Methodology	8
3	Detailed Results	10
3.1	CVF-1	10
3.2	CVF-2	10
3.3	CVF-3	11
3.4	CVF-4	12
3.5	CVF-5	13
3.6	CVF-6	14
3.7	CVF-7	14
3.8	CVF-8	15
3.9	CVF-9	16
3.10	CVF-10	16
3.11	CVF-11	16
3.12	CVF-12	17
3.13	CVF-13	17
3.14	CVF-14	17
3.15	CVF-15	18
3.16	CVF-16	18
3.17	CVF-17	18
3.18	CVF-18	18
3.19	CVF-19	19
3.20	CVF-20	19
3.21	CVF-21	19
3.22	CVF-22	20
3.23	CVF-23	20
3.24	CVF-24	20
3.25	CVF-25	21
3.26	CVF-26	21
3.27	CVF-27	21
3.28	CVF-28	22
3.29	CVF-29	22
3.30	CVF-30	23
3.31	CVF-31	23
3.32	CVF-32	24
3.33	CVF-33	24
3.34	CVF-34	25
3.35	CVF-35	25
3.36	CVF-36	26
3.37	CVF-37	26

3.38 CVF-38	26
3.39 CVF-39	27
3.40 CVF-40	27

ABDK

1 Document properties

Version

Version	Date	Author	Description
0.1	January 31, 2022	D. Khovratovich	Initial Draft
0.2	January 31, 2022	D. Khovratovich	Minor revision
1.0	January 31, 2022	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 **repository** with the following files:

- Parameters.sol
- Pool.sol

The fixes were provided in a **new commit**.

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 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** Info
- **Source** Parameters.sol

Description Should be "^{0.8.0}" unless there is something special about this particular version.
Client Comment Will not fix. 0.8.0 compiler gets internal compiler error when try to compile the source code.

Listing 1:

```
2 pragma solidity ^0.8.10;
```

3.2 CVF-2

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

Description This comment is confusing. Consider either removing it or adding some explanation about how to read it.

Listing 2:

```
4 /*
   uint256;*;32;transfer_nullifier;
   uint256;*;32;transfer_out_commit;
   uint48;*;6;transfer_index;
   int112;*;14;transfer_energy_amount;
   int64;*;8;transfer_token_amount;
10 uint256[8] calldata;*;256;transfer_proof
   uint256;32;*;tree_root_after
   uint256[8] calldata;*;256;tree_proof
   uint256;*;2;tx_type
   uint256;*;2;memo_data_size
   bytes calldata;*;memo_data_size();memo_data
   bytes32;*;32;sign_r
   bytes32;*;32;sign_vs
   uint256;{transfer_index};28;transfer_delta
   bytes calldata;{memo_data}+memo_fixed_size();memo_data_size()—
      ↪ memo_fixed_size();memo_message
20 uint256;{memo_data};8;memo_fee
   uint256;*;8;memo_native_amount
   uint256;*;20;memo_receiver
   */
```

3.3 CVF-3

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

Description This contract should be moved to a separate file named "CustomABIDecoder.sol".

Listing 3:

```
25 contract CustomABIDecoder {
```

3.4 CVF-4

- **Severity** Minor
- **Category** Bad naming
- **Status** Info
- **Source** Parameters.sol

Description Constants are usually named IN_UPPER_CASE

Client Comment Will not fix.

Listing 4:

```

26 uint256 constant transfer_nullifier_pos = 4;
   uint256 constant transfer_nullifier_size = 32;

46 uint256 constant transfer_index_pos = transfer_out_commit_pos +
   ↪ transfer_out_commit_size;
   uint256 constant transfer_index_size = 6;

53 uint256 constant transfer_energy_amount_pos = transfer_index_pos
   ↪ + transfer_index_size;
   uint256 constant transfer_energy_amount_size = 14;

60 uint256 constant transfer_token_amount_pos =
   ↪ transfer_energy_amount_pos + transfer_energy_amount_size;
   uint256 constant transfer_token_amount_size = 8;

67 uint256 constant transfer_proof_pos = transfer_token_amount_pos
   ↪ + transfer_token_amount_size;
   uint256 constant transfer_proof_size = 256;

77 uint256 constant tree_root_after_pos = transfer_proof_pos +
   ↪ transfer_proof_size;
   uint256 constant tree_root_after_size = 32;

84 uint256 constant tree_proof_pos = tree_root_after_pos +
   ↪ tree_root_after_size;
   uint256 constant tree_proof_size = 256;

94 uint256 constant tx_type_pos = tree_proof_pos + tree_proof_size;
   uint256 constant tx_type_size = 2;
   uint256 constant tx_type_mask = (1 << (tx_type_size*8)) - 1;

102 uint256 constant memo_data_size_pos = tx_type_pos + tx_type_size
   ↪ ;
   uint256 constant memo_data_size_size = 2;
   uint256 constant memo_data_size_mask = (1 << (
   ↪ memo_data_size_size*8)) - 1;
   (... 107, 125, 135, 161, 169, 177)

```

3.5 CVF-5

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

Description There is no access level specified for these constants, so internal access will be sued by default. Consider explicitly specifying an access level.

Client Comment Will not fix

Listing 5:

```
26 uint256 constant transfer_nullifier_pos = 4;
   uint256 constant transfer_nullifier_size = 32;

39 uint256 constant transfer_out_commit_pos =
   ↪ transfer_nullifier_pos + transfer_nullifier_size;
40 uint256 constant transfer_out_commit_size = 32;

46 uint256 constant transfer_index_pos = transfer_out_commit_pos +
   ↪ transfer_out_commit_size;
   uint256 constant transfer_index_size = 6;

53 uint256 constant transfer_energy_amount_pos = transfer_index_pos
   ↪ + transfer_index_size;
   uint256 constant transfer_energy_amount_size = 14;

60 uint256 constant transfer_token_amount_pos =
   ↪ transfer_energy_amount_pos + transfer_energy_amount_size;
   uint256 constant transfer_token_amount_size = 8;

67 uint256 constant transfer_proof_pos = transfer_token_amount_pos
   ↪ + transfer_token_amount_size;
   uint256 constant transfer_proof_size = 256;

77 uint256 constant tree_root_after_pos = transfer_proof_pos +
   ↪ transfer_proof_size;
   uint256 constant tree_root_after_size = 32;

84 uint256 constant tree_proof_pos = tree_root_after_pos +
   ↪ tree_root_after_size;
   uint256 constant tree_proof_size = 256;

94 uint256 constant tx_type_pos = tree_proof_pos + tree_proof_size;
   uint256 constant tx_type_size = 2;
   uint256 constant tx_type_mask = (1 << (tx_type_size*8)) - 1;
(... 102, 107, 125, 135, 161, 169, 177, 188)
```

3.6 CVF-6

- **Severity** Minor
- **Category** Bad naming
- **Status** Info
- **Source** Parameters.sol

Description The function name doesn't give a clue about where the function loads a value from. Consider renaming to "calldataload". It would make it obvious that the function is just a wrapper for the opcode with the same name.

Client Comment Will not fix. We use it, because inline assembly is not compatible with constant variables. Seems, it is issue of the compiler.

Listing 6:

```
29 function _loaduint256(uint256 pos) pure internal returns(uint256  
    ↪ r) {
```

3.7 CVF-7

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

Description Constant '32' should be named

Listing 7:

```
50 r = uint48(_loaduint256(transfer_index_pos+transfer_index_size  
    ↪ -32));  
  
64 r = int64(uint64(_loaduint256(transfer_token_amount_pos+  
    ↪ transfer_token_amount_size-32)));  
  
99 r = _loaduint256(tx_type_pos+tx_type_size-32) & tx_type_mask;
```

3.8 CVF-8

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

Description These non-elegant "pos+size-32" formulas wouldn't be needed in case the "_loaduint256" would return bytes32 instead of uint256.

Recommendation Consider the following code chunk:
<https://gist.github.com/3sGgpQ8H/87ab4df1c7b7ab91cbfd0f4b249a26be>

Client Comment Will not fix

Listing 8:

```
50 r = uint48(_loaduint256(transfer_index_pos+transfer_index_size
    ↪ -32));

57 r = int112(uint112(_loaduint256(transfer_energy_amount_pos+
    ↪ transfer_energy_amount_size-32)));

64 r = int64(uint64(_loaduint256(transfer_token_amount_pos+
    ↪ transfer_token_amount_size-32)));

99 r = _loaduint256(tx_type_pos+tx_type_size-32) & tx_type_mask;

109 r = _loaduint256(memo_data_size_pos+memo_data_size_size-32) &
    ↪ memo_data_size_mask;

139 r = _loaduint256(transfer_index_pos+transfer_delta_size-32) &
    ↪ transfer_delta_mask;

166 r = _loaduint256(memo_fee_pos+memo_fee_size-32) & memo_fee_mask;

174 r = _loaduint256(memo_native_amount_pos+memo_native_amount_size
    ↪ -32) & memo_native_amount_mask;

181 r = address(uint160(_loaduint256(memo_receiver_pos+
    ↪ memo_receiver_size-32)));
```

3.9 CVF-9

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

Description This constant must be related to 'transfer_proof_size'

Client Comment Will not fix.

Listing 9:

```
70 function _transfer_proof() pure internal returns (uint256[8]  
    ↪ calldata r) {
```

3.10 CVF-10

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

Description This constant must be related to 'tree_proof_size'

Client Comment Will not fix.

Listing 10:

```
87 function _tree_proof() pure internal returns (uint256[8]  
    ↪ calldata r) {
```

3.11 CVF-11

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

Description These constants should be named or, even better, belong to some enumerated type.

Client Comment Will not fix

Listing 11:

```
145 r = 8;  
147 r = 36;
```


3.12 CVF-12

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

Description This silently drops the leftmost 28 bytes of the pool ID. If these bytes are always zero, consider changing the pool ID type to "uint32".

Client Comment Will not fix, `_pool_id` assumed to be uint24 (range check implemented in the constructor).

Listing 12:

```
202 r[3] = _transfer_delta() + (_pool_id() << (transfer_delta_size * 8))  
    ↪ ;
```

3.13 CVF-13

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

Description This should be 'S_MAX'

Listing 13:

```
219 uint256(s) <= 0  
    ↪ x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681  
    ↪  
220 B20A0,
```

3.14 CVF-14

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

Description This will return 0 for invalid signature.

Recommendation Consider reverting in this case. Not reverting here allows anyone to sign anything on behalf of the zero address.

Client Comment Will not fix. Zero address cannot grant to the pool anything to spent.

Listing 14:

```
223 return ecrecover(prefixedHash, v, r, s);
```

3.15 CVF-15

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

Description Should be " $\wedge 0.8.0$ " unless there is something special about this particular version.

Listing 15:

```
2 pragma solidity ^0.8.10;
```

3.16 CVF-16

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

Description We did not review this file

Client Comment Will not fix.

Listing 16:

```
8 import "../consensus/IOperatorManager.sol";
```

3.17 CVF-17

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

Description This interface should be moved to a separate file named "ITransferVerifier.sol".

Listing 17:

```
12 interface ITransferVerifier {
```

3.18 CVF-18

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

Description This constant should relate to 'transfer_proof_size'

Client Comment Will not fix.

Listing 18:

```
15 uint256[8] memory p
```

3.19 CVF-19

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

Description This interface should be moved to a separate file named "ITreeVerifier.sol".

Listing 19:

```
19 interface ITreeVerifier {
```

3.20 CVF-20

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

Description This constant should relate to 'tree_proof_size'

Client Comment Will not fix.

Listing 20:

```
22 uint256[8] memory p
```

3.21 CVF-21

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

Description This interface should be moved to a separate file named "IMintable.sol".

Listing 21:

```
26 interface IMintable {
```

3.22 CVF-22

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

Description Consider using the "uint32" type for pool ID.

Client Comment Will not fix. We have already MAX_POOL_ID check and it is uint24.

Listing 22:

```
33 uint256 immutable public pool_id;  
44 uint256 constant internal MAX_POOL_ID = 0xffffffff;  
60 constructor(uint256 __pool_id, IERC20 _token, IMintable  
    ↪ _voucher_token, uint256 _denominator, uint256  
    ↪ _energy_denominator, uint256 _native_denominator,
```

3.23 CVF-23

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

Description This constant would be redundant in case the "uint32" type would be used for pool IDs.

Client Comment Will not fix.

Listing 23:

```
44 uint256 constant internal MAX_POOL_ID = 0xffffffff;
```

3.24 CVF-24

- **Severity** Minor
- **Category** Documentation
- **Status** Info
- **Source** Pool.sol

Description The semantics of the keys in these mappings is unclear.

Recommendation Consider documenting.

Client Comment Will not fix.

Listing 24:

```
53 mapping (uint256 => uint256) public nullifiers;  
mapping (uint256 => uint256) public roots;
```

3.25 CVF-25

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

Description These variables are used without being initialized.

Recommendation Consider initializing explicitly in the constructor.

Client Comment Will not fix. These variables initialized with zero value.

Listing 25:

```
55 uint256 public pool_index;  
   bytes32 public all_messages_hash;
```

3.26 CVF-26

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

Description There are no range checks for denominator arguments.

Recommendation Consider adding appropriate checks. E.g. ensure that the denominators are not zero.

Client Comment Will not fix.

Listing 26:

```
60 constructor(uint256 __pool_id, IERC20 _token, IMintable  
   ↪ _voucher_token, uint256 _denominator, uint256  
   ↪ _energy_denominator, uint256 _native_denominator,
```

3.27 CVF-27

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

Description This function should probably emit an event

Client Comment Will not fix.

Listing 27:

```
75 function initialize() public initializer{  
   roots[0] = first_root;
```

3.28 CVF-28

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

Description Consider reverting if the retrieved value is 0

Client Comment Will not fix. zkSNARKs cannot be proved with zero root hash.

Listing 28:

```
82 return roots[pool_index];  
86 return roots[_transfer_index()];
```

3.29 CVF-29

- **Severity** Minor
- **Category** Documentation
- **Status** Info
- **Source** Pool.sol

Description It is worth documenting what exactly these proofs verify and which contract variables are involved.

Client Comment Will not fix.

Listing 29:

```
95 require(transfer_verifier.verifyProof(_transfer_pub(),  
    ↪ _transfer_proof()), "bad transfer proof");  
137 require(tree_verifier.verifyProof(_tree_pub(), _tree_proof()), "  
    ↪ bad tree proof");
```

3.30 CVF-30

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

Description The expression "`_transfer_nullifier()`" is calculated twice.

Recommendation Consider calculating once and reusing.

Client Comment Will not fix. `CALLDATALOAD` cost 3 gas, and we does not use stack slots for all variables.

Listing 30:

```
96 require(nullifiers[_transfer_nullifier()]==0,"doublespend
    ↳ detected");

133 nullifiers[_transfer_nullifier()] = (1<<255) | (uint64(
    ↳ _transfer_token_amount()) << 160) | (uint112(
    ↳ _transfer_energy_amount()) << 48) | _pool_index;
```

3.31 CVF-31

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

Description Overflow is possible here for the `uint256` type of 'fee'.

Recommendation Consider using a shorter type for fee.

Client Comment Will not fix. `token_amount` is `uint64`.

Listing 31:

```
101 int256 token_amount = _transfer_token_amount() + int256(fee);
```

3.32 CVF-32

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

Description The expression "`_transfer_token_amount()`" is calculated twice.

Recommendation Consider calculating once and reusing.

Client Comment Will not fix. `CALLDATALOAD` cost 3 gas, and we does not use stack slots for all variables.

Listing 32:

```
101 int256 token_amount = _transfer_token_amount() + int256(fee);
133 nullifiers[_transfer_nullifier()] = (1<<255) | (uint64(
    ↪ _transfer_token_amount()) << 160) | (uint112(
    ↪ _transfer_energy_amount()) << 48) | _pool_index;
```

3.33 CVF-33

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

Description The expression "`_transfer_energy_amount`" is calculated twice.

Recommendation Consider calculating once and reusing.

Client Comment Will not fix. `CALLDATALOAD` cost 3 gas, and we does not use stack slots for all variables.

Listing 33:

```
102 int256 energy_amount = _transfer_energy_amount();
133 nullifiers[_transfer_nullifier()] = (1<<255) | (uint64(
    ↪ _transfer_token_amount()) << 160) | (uint112(
    ↪ _transfer_energy_amount()) << 48) | _pool_index;
```


3.34 CVF-34

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

Description There should be named constants for the valid transaction types.

Client Comment Will not fix.

Listing 34:

```
104 if (_tx_type()==0) { // Deposit
107 } else if (_tx_type()==1) { // Transfer
110 } else if (_tx_type()==2) { // Withdraw
```

3.35 CVF-35

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

Description The expression "`_tx_type()`" is calculated several times.

Recommendation Consider calculating once and reusing.

Client Comment Will not fix. `CALLDATALOAD` cost 3 gas, and we does not use stack slots for all variables.

Listing 35:

```
104 if (_tx_type()==0) { // Deposit
107 } else if (_tx_type()==1) { // Transfer
110 } else if (_tx_type()==2) { // Withdraw
```

3.36 CVF-36

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

Description The expression "`_memo_receiver()`" is calculated several times.

Recommendation Consider calculating once and reusing.

Client Comment Will not fix. `CALLDATALOAD` cost 3 gas, and we does not use stack slots for all variables.

Listing 36:

```
114 token.safeTransfer(_memo_receiver(), uint256(-token_amount)*
    ↪ denominator);
119 voucher_token.mint(_memo_receiver(), uint256(-energy_amount)*
    ↪ energy_denominator);
123 payable(_memo_receiver()).transfer(msg.value);
```

3.37 CVF-37

- **Severity** Moderate
- **Category** Flaw
- **Status** Fixed
- **Source** Pool.sol

Description The returned value is ignored. the exact semantics of the returned value is unclear, but if it is the error indicator, then consider reverting in case the returned value is false.

Recommendation Consider using "`send`" instead and protect against reentrancy in some other way..

Listing 37:

```
119 voucher_token.mint(_memo_receiver(), uint256(-energy_amount)*
    ↪ energy_denominator);
```

3.38 CVF-38

- **Severity** Minor
- **Category** Flaw
- **Status** Fixed
- **Source** Pool.sol

Description Using "`transfer`" is discouraged, as operation gas costs could change in the future, and operation that fit in 2300 gas in the past a not guaranteed to fit going forward.

Listing 38:

```
123 payable(_memo_receiver()).transfer(msg.value);
```

3.39 CVF-39

- **Severity** Critical
- **Category** Flaw
- **Status** Fixed
- **Source** Pool.sol

Description Reentrancy attack is possible, as the nullifier is marked after calling external contracts. Note, that some token contracts, namely those implementing ERC-777, may call hook on receiving smart contracts, making simple token transfer unsafe from the reentrancy point of view.

Recommendation Consider calling external contracts after updating the state and/or implementing some other kind of reentrancy protection.

Listing 39:

```
133 nullifiers[_transfer_nullifier()] = (1<<255) | (uint64(  
    ↪ _transfer_token_amount()) << 160) | (uint112(  
    ↪ _transfer_energy_amount()) << 48) | _pool_index;
```

3.40 CVF-40

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

Description The function always returns true.

Recommendation Consider returning nothing.

Listing 40:

```
150 return true;
```