ABDK CONSULTING

SMART CONTRACT AUDIT

TORNADO

abdk.consulting

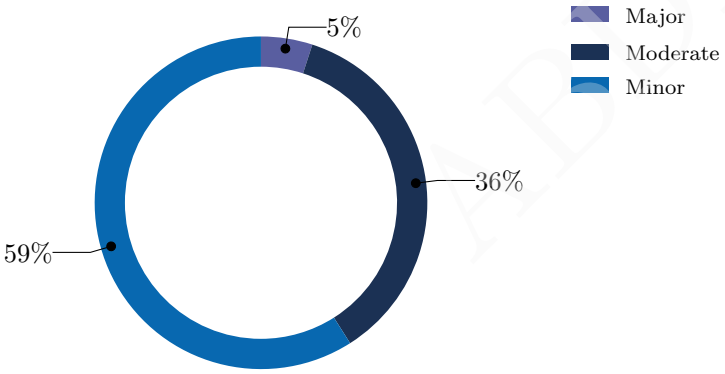
SMART CONTRACT AUDIT CONCLUSION

Mikhail Vladimirov and Dmitry Khovratovich

22nd December 2020

Summary

We've been asked to review the Tornado smart contracts given in separate files. At some point we were also given the formal spec.



Findings

ID	Issue	Description	Status
CVF-1	Minor	Suboptimal Solidity Version	Fixed
CVF-2	Minor	Generic interface name	Info
CVF-3	Minor	Reference to Poseidon	Info
CVF-4	Major	Large input type	Info
CVF-5	Moderate	Return absence	Fixed
CVF-6	Minor	Generic interface name	Info
CVF-7	Minor	Generic function name	Info
CVF-8	Minor	Incorrect function	Fixed
CVF-9	Minor	Incorrect function parameter naming	Fixed
CVF-10	Minor	Documentation Comment	Info
CVF-11	Minor	Complicated interface	Info
CVF-12	Minor	Uninitialized variable	Info
CVF-13	Minor	Bitwise operation	Info
CVF-14	Minor	Expensive deployment	Info
CVF-15	Moderate	Incorrect comment	Info
CVF-16	Minor	Documentation comment needed	Info
CVF-17	Moderate	The SafeMath.sub incorrect using	Info
CVF-18	Moderate	Unclear function behavior	Fixed
CVF-19	Moderate	Redundant variable	Fixed
CVF-20	Moderate	Common functionality	Info
CVF-21	Moderate	Suboptimal Deploy	Info
CVF-22	Moderate	Inefficient hashing	Info
CVF-23	Minor	Redundant word ”Data”	Info
CVF-24	Moderate	Not indexed parameters	Info
CVF-25	Moderate	Complicated Interface	Info
CVF-26	Moderate	The expensive deployment	Info
CVF-27	Minor	The redundant call	Info
CVF-28	Moderate	Gas spending	Info
CVF-29	Minor	Suboptimal Parameter	Info
CVF-30	Moderate	Incorrect Modifier	Fixed
CVF-31	Major	Numerous checks	Info

ID	Issue	Description	Status
CVF-32	Major	Inputs Overflow	Info
CVF-33	Moderate	Similar function	Info
CVF-34	Moderate	Missed modifier	Fixed
CVF-35	Minor	The name suggestion	Fixed
CVF-36	Minor	The name suggestion-2	Fixed
CVF-37	Moderate	The public function	Info
CVF-38	Minor	The confusing interface name	Info
CVF-39	Minor	The redundant word "New"	Fixed
CVF-40	Minor	The Typo	Fixed
CVF-41	Moderate	The complicated check	Info
CVF-42	Moderate	Gas efficient	Fixed
CVF-43	Moderate	Suboptimal loop	Info
CVF-44	Moderate	Suboptimal condition	Fixed
CVF-45	Minor	The suboptimal index	Info
CVF-46	Minor	Redundant loop	Info
CVF-47	Minor	Suboptimal delegating	Open
CVF-48	Minor	The Confusing function name	Info
CVF-49	Minor	Misleading file name	Info
CVF-50	Minor	Expensive deployment	Info
CVF-51	Minor	Inefficient function	Info
CVF-52	Minor	The return indexes absence	Info

Contents

1 Introduction

The following document provides the result of the audit performed by ABDK Consulting at the Tornado request. The next initial data were obtained:

- the code from a private repository at GitHub/TornadoCash at the commit 9ec05a681d9699a11733b3163dd44a1e90abc345.

The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.

1.1 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.

1.2 About Customer

Tornado Cash is a decentralized Ethereum Mixer. ABDK had audited previous versions of Tornado Cash, and is now reviewing the changes only.

1.3 Disclaimer

Note that the performed audit represents current best practice 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 Findings

IHasher.sol	
Severity	**Issues**
Major	1
Minor	3
IRewardSwap.sol	
Severity	Issues
Moderate	1
Minor	1
IVerifier.sol	
Severity	Issues
Minor	3
ITornado.sol	
Severity	Issues
Minor	4
MIner.sol	
Severity	Issues
Moderate	5
Minor	4
TornadoTrees.sol	
Severity	Issues
Major	2
Moderate	13
Minor	8

MerkleTreeWithHistory.sol	
Severity	Issues
Moderate	5
Minor	2
TornadoProxy.sol	
Severity	Issues
Minor	2
RewardSwap.sol	
Severity	Issues
Minor	3
OwnableMerkleTree.sol	
Severity	Issues
Minor	2
Total	58

3 Detailed Results

3.1 CVF-1 Suboptimal Solidity Version

- **Severity** Minor
- **Category** Suboptimal Code
- **Status** Closed
- **Source** IHasher.sol, IRewardSwap.sol, IVerifier.sol, ITornado.sol, TornadoTrees.sol, TornadoProxy.sol, OwnableMerkleTree.sol, ITornadoTrees

Description The current Solidity version should be 0.6.0 according to the common best practice.
Recommendation Change Solidity version.

Listing 1: Suboptimal Solidity Version

```
3 pragma solidity ^0.6.12;
```

3.2 CVF-2 Generic interface name

- **Severity** Minor
- **Category** Suboptimal Code
- **Status** Info
- **Source** IHasher.sol

Description The current interface name it too generic. The function name refer to Poseidon and input sizes supported related to particular business use cases.
Recommendation Consider renaming:

- functions to reflect corresponding use case. The `hashTreeNode` for the function with two inputs and the `hashDeposit` for the function with three inputs
- the interface for example `TornadoHasher`.

Listing 2: Generic interface name

```
5   interface IHasher {;
```

3.3 CVF-3 Reference to Poseidon

- **Severity** Minor
- **Category** Suboptimal Code
- **Status** Info
- **Source** IHasher.sol

Description The references to Poseidon in this line may interfere the another hash function switch.
Recommendation Change implementation.
Client Comment Implementation already has the `poseidon` function selector, we can't change it in the interface.

Listing 3: Reference to Poseidon

```
6   function poseidon(bytes32[2] calldata inputs) external
pure returns (bytes32);
```

3.4 CVF-4 Large input type

- **Severity** Major
- **Status** Info
- **Category** Suboptimal Code
- **Source** IHasher.sol

Description Poseidon-like hash functions typically accept inputs from domain smaller than bytes32, whereas bytes32 elements should be represented at least as a pair of input elements.
Recommendation Consider explicitly passing bytes31 or another smaller type to ensure there is no collision.
Client Comment This is a valid point but requires a fix in circomlib. In our project this function is only called from TornadoTrees.sol, where we know that all arguments are not overflowing: block number, address, and output of mimic hash (tornado commitment and nullifier)

Listing 4: Large input type

```
6   function poseidon(bytes32[2] calldata inputs) external
    pure returns (bytes32);
```

3.5 CVF-5 Return absence

- **Severity** Moderate
- **Status** Fixed
- **Category** Suboptimal Code
- **Source** IRewardSwap.sol

Description The `swap` function should return the number of the swapped tokens.
Recommendation Consider adding return.

Listing 5: Return absence

```
7    function swap(address recipient, uint amount) external;
```

3.6 CVF-6 Generic interface name

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal Code
- **Source** IVerifier.sol

Description The name of the `IVerifier` interface is quite generic, while the set of function corresponds to the particular business use cases of Tornado protocol.
Recommendation Consider renaming the interface to `ITornadoVerifier`. Also see 3.2 for function name suggestions.

Listing 6: Generic interface name

```
5   interface IVerifier {
```

3.7 CVF-7 Generic function name

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal Code
- **Source** IVerifier.sol

Description The numbers look arbitrary, while function name looks generic.
Recommendation If each number of inputs corresponds to a particular business scenario, then this should be reflected in the function name like `verifyTreeUpdateProof` for the function with 4 inputs, `verifyWithdrawProof` for the function with 7 inputs, `verifyRewardProof` for the function with 12 inputs.
Client Comment We can't change selectors because `verifier` contract is auto generated.

Listing 7: Generic function name

```
6 function verifyProof(bytes calldata proof, uint256[4] calldata input)
external view returns (bool);
```

3.8 CVF-8 Incorrect function

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

Description Perhaps, in this line a typo.
Recommendation Perhaps, the correct function should be **ITornadoInstance**.
Client Comment We can't change selectors because `verifier` contract is auto generated.

Listing 8: Incorrect function

```
5 interface ITornado {;
```

3.9 CVF-9 Incorrect function parameter naming

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

Description Unlike names of function parameters in other interfaces, in this line underscore prefixes are used.
Recommendation Consider using consistent naming.

Listing 9: Incorrect function parameter naming

```
6 function deposit(bytes32 _commitment) external payable;
```

3.10 CVF-10 Documentation Comment

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

Description It is unclear what happens with ether sent along with a call.
Recommendation Consider adding documentation comment.
Client Comment We think docs for this should be in tornado-core project rather than this interface.

Listing 10: Documentation Comment

```
16   ) external payable;
```

3.11 CVF-11 Complicated interface

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

Description Each verifier has functions to verify all three proofs: reward, withdraw and tree update.
Recommendation Consider splitting IVerifier interface into three interfaces (probably implemented in the same contract).

Listing 11: Complicated interface

```
16      IVerifier public immutable rewardVerifier;
17      IVerifier public immutable withdrawVerifier;
18      IVerifier public immutable treeUpdateVerifier;
```

3.12 CVF-12 Uninitialized variable

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

Description The `accountCount` variable is not initialized in the constructor.
Recommendation Consider to initialize the function.
Client Comment Its initial value of 0 is correct.

Listing 12: Uninitialized variable

```
27      uint256 public accountCount;
```

3.13 CVF-13 Bitwise operation

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

Recommendation 128 or other power of two would allow using bitwise operations for looping the history.
Client Comment We think bitwise operations are less readable than modulo

Listing 13: Bitwise operation

```
28        uint256 public constant ACCOUNT_ROOT_HISTORY_SIZE = 100;
```

3.14 CVF-14 Expensive deployment

- **Severity** Moderate
- **Status** Info
- **Category** Suboptimal
- **Source** MIner.sol

Description The `resolve` using makes deployment more expensive and less convenient in development environment.
Recommendation Consider passing the `iRewardSwap` address directly.
Client Comment Its is required to solve circular dependencies in our create2 deploy script.

Listing 14: Expensive deployment

```
98  rewardSwap = IRewardSwap(resolve(_rewardSwap));
99  governance = resolve(_governance);
100 tornadoTrees = TornadoTrees(resolve(_tornadoTrees));
101 rewardVerifier = IVerifier(resolve(verifiers[0]));
102 withdrawVerifier = IVerifier(resolve(verifiers[1]));
103 treeUpdateVerifier = IVerifier(resolve(verifiers[2]));
```

3.15 CVF-15 Incorrect comment

- **Severity** Moderate
- **Status** Info
- **Category** Suboptimal
- **Source** MIner.sol

Description The original comment *insert empty tree root without incriminating* `accountCount` *counter*. There is no guarantee that the root is empty
Recommendation Consider using hardcoded constant value here or guarantee emptiness in some other way.
Client Comment The constant depends on tree depth. It has to be passed correctly and the contract has no way to verify that (except very expensive way of calculating many poseidon hashes). Since after deployment everyone can verify that it was passed correctly, we decided to leave it as is.

Listing 15: Incorrect comment

```
105 // insert empty tree root without incrementing accountCount counter
```

3.16 CVF-16 Documentation comment needed

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

Description An old account is nullified with `inputNullifierHash`, and a new account is inserted, so that the new account has rate*block difference fee more money. The block difference is difference between deposit and withdrawal.
Recommendation Consider add a comment on what statement is asserted by the proof.
Client Comment All relevant docs/comments are in `rewardVerifier` circuit file.

Listing 16: Documentation comment needed

```
135  rewardVerifier.verifyProof(
```

3.17 CVF-17 SafeMath.sub incorrect using

- **Severity** Moderate
- **Status** Info
- **Category** Suboptimal
- **Source** MIner.sol

Description The `SafeMath.sub` is used to enforce business-level constraint. Generally it is supposed to be used as a second line of defence and catch coding errors and incorrect usage.
Client Comment Why it shouldn't be used for business level constraint?.

Listing 17: SafeMath.sub incorrect using

```
202  uint256 amount = _args.amount.sub(_args.extData.fee, "Amount should be
greater than fee");
```

3.18 CVF-18 Unclear function behavior

- **Severity** Moderate
- **Status** Fixed
- **Category** Unclear behavior
- **Source** MIner.sol

Description The `keccak252` function actually truncates keccak256 hash to 248 bits rather than 252. It is unclear to say intentional it or not?

Listing 18: Unclear function behavior

```
257  function keccak252(bytes memory data) internal pure returns (bytes32) {
```

3.19 CVF-19 Redundant variable

- **Severity** Moderate
- **Status** Fixed
- **Category** Unclear behavior
- **Source** MIner.sol

Description The `commitment` function seems to be redundant. It can be taken from args.

Listing 19: Redundant variable

```
268   require(_args.leaf == _commitment, "Incorrect commitment inserted");
```

3.20 CVF-20 Common functionality

- **Severity** Moderate
- **Status** Info
- **Category** Suboptimal
- **Source** TornadoTrees.sol

Description The `OwnableMerkleTree` is just a particular implementation of quite common functionality.
Recommendation The type of this storage variable should be turned to some interface like the `IMerkleTreeWithHistory`.

Listing 20: Common functionality

```
12   OwnableMerkleTree public immutable depositTree;
13   OwnableMerkleTree public immutable withdrawalTree;
```

3.21 CVF-21 Suboptimal deploy

- **Severity** Moderate
- **Status** Info
- **Category** Suboptimal
- **Source** TornadoTrees.sol

Description Deploying each merkle tree as a separate contract is suboptimal.
Recommendation More efficient solution would be to implement Merkle tree with history as a structure + a library and then allocate it in the contract's own storage.
Client Comment We typically do only 1 call to it per transaction. Changing it to storage + library will require a major rewrite and we don't think it's worth it.

Listing 21: Suboptimal deploy

```
12   OwnableMerkleTree public immutable depositTree;
```

3.22 CVF-22 Inefficient hashing

- **Severity** Moderate
- **Status** Info
- **Category** Suboptimal
- **Source** TornadoTrees.sol

Description external contract for hashing is inefficient.
Recommendation Consider implementing hashing as a library
Client Comment External library will cost the same in terms of gas, and we can't inline it in the same contract since it's written in pure evm assembly and also will not fit in contract size

Listing 22: Inefficient hashing

```
14  IHasher public immutable hasher;
```

3.23 CVF-23 Redundant word "Data"

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

Description the `data` word is redundant.
Client Comment This is not deposit itself, but rather an event that we received metadata about it. We think it should be left as is.

Listing 23: Redundant word "Data"

```
23  event DepositData(address instance, bytes32 indexed hash, uint256 block,
uint256 index);
24  event WithdrawalData(address instance, bytes32 indexed hash, uint256
block, uint256 index);
```

3.24 CVF-24 Not indexed parameters

- **Severity** Moderate
- **Status** Info
- **Category** Suboptimal
- **Source** TornadoTrees.soll

Description The `instance` parameters should probably be indexed.
Client Comment We can't imagine the case when it necessary. left as is.

Listing 24: Not indexed paremeters

```
23  event DepositData(address instance, bytes32 indexed hash, uint256 block,
uint256 index);
24  event WithdrawalData(address instance, bytes32 indexed hash, uint256
block, uint256 index);
```

3.25 CVF-25 Complicated Interface

- **Severity** Moderate
- **Status** Info
- **Category** Suboptimal
- **Source** TornadoTrees.sol

Description Both hashers implement both, 2- and 3-input hashing, while only one of them is used in each hasher.
Recommendation Consider splitting IHasher interface into two interfaces and name them according to business use cases they cover, such as `LeafHasher` and `NodeHasher`.
Client Comment It's nice to have it, but we decided to leave it as is.

Listing 25: Complicated Interface

```
39  bytes32 _hasher2,
40  bytes32 _hasher3,
```

3.26 CVF–26 The expensive deployment

- **Severity** Moderate
- **Status** Info
- **Category** Suboptimal
- **Source** TornadoTrees.sol

Description using the `resolve` makes deployment more expensive and less convenient in DEV environment.
Recommendation Consider passing `TornadoProxy` address directly.

Listing 26: The expensive deployment

```
43  tornadoProxy = resolve(_tornadoProxy);
44  hasher = IHasher(resolve(_hasher3));
45  depositTree = new OwnableMerkleTree(_levels,IHasher(resolve
(_hasher2)));
```

3.27 CVF-27 The redundant call

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

Description the `resolve` call is redundant, as `hasher2` was already resolved in the previous line.
Recommendation Consider caching in a local variable and reusing.
Client Comment It is done only once since it's a constructor, no big deal.

Listing 27: The redundant call

```
46  withdrawalTree = new OwnableMerkleTree(_levels, IHasher(resolve
    (_hasher2)));
```

3.28 CVF-28 Gas spending

- **Severity** Moderate
- **Status** Info
- **Category** Suboptimal
- **Source** TornadoTrees.sol

Description the `push` updates two storage slots: array element and array length. Thus, gas could be saved by batching multiple deposits together.
Recommendation Consider implementing bulk register deposit operation.
Client Comment This function is called on every tornado cash deposit. We don't do batching there.

Listing 28: Gas spending

```
50  deposits.push(keccak256(abi.encode(instance, commitment,
    blockNumber())));
54  withdrawals.push(keccak256(abi.encode(instance, nullifier,
    blockNumber())));
```

3.29 CVF-29 Suboptimal Parameter

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

Description Using `Poseidon` in this line here as in the `updateDepositTree` would allow shorter tree update calls.
Client Comment Poseidon call costs >1000 gas while keccak is only 30.

Listing 29: Suboptimal Parameter

```
50  deposits.push(keccak256(abi.encode(instance, commitment,
blockNumber())));
54  withdrawals.push(keccak256(abi.encode(instance, nullifier,
blockNumber())));
```

3.30 CVF-30 Incorrect Modifier

- **Severity** Moderate
- **Status** Fixed
- **Category** Suboptimal
- **Source** TornadoTrees.sol

Description the `memory` should be calldata, otherwise `calldata` modifiers of `updateRoots` parameters does not make sense.

Listing 30: Incorrect Modifier

```
62  function updateDepositTree(TreeLeaf[] memory _deposits) public {
```

3.31 CVF-31 Numerous checks

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

Description
Recommendation The length will be checked on every loop iteration while it would be enough to check once before the loop.
Client Comment How do we check length only once before the loop? Is it possible to remove array length check?.

Listing 31: Numerous checks

```
69  require(deposits[offset + i] == leafHash, "Incorrect deposit");
88  require(withdrawals[offset + i] == leafHash, "Incorrect
withdrawal");
```

3.32 CVF-32 Inputs overflow

- **Severity** Major
- **Status** Info
- **Category** Overflow
- **Source** TornadoTrees.sol

Description Some inputs may overflow the native poseidon domain size, thus creating overflows and maybe even collisions.
Recommendation Consider adding explicit range checks.
Client Comment Instance is of type address, it can't overflow. Block number cannot be that high too. And hash is the mimc hash taken from tornado cash instance and is guaranteed to be inside the field.

Listing 32: Inputs overflow

```
71  leaves[i] = hasher.poseidon([bytes32(uint256(deposit.instance)),
deposit.hash, bytes32(deposit.block)]);
90  leaves[i] = hasher.poseidon([bytes32(uint256(withdrawal.instance)),
withdrawal.hash, bytes32(withdrawal.block)]);
```

3.33 CVF-33 Similar function

- **Severity** Moderate
- **Status** Info
- **Category** Suboptimal
- **Source** TornadoTrees.sol

Description the `updateWithdrawalTree` function is very similar to `updateDepositTree` and almost all the code could be reused.
Recommendation Add the code into internal function that accepts storage reference to deposits/withdrawals array, last processed deposit/withdrawal, and deposit/withdrawal tree, and returns new value for last processed deposit/withdrawal.
Client Comment How to emit the right events inside the loop?

Listing 33: Similar function

```
81  function updateWithdrawalTree(TreeLeaf[] memory _withdrawals)
public {
```

3.34 CVF-34 Missed modifier

- **Severity** Moderate
- **Status** Fixed
- **Category** Suboptimal
- **Source** TornadoTrees.sol

Recommendation In the line instead of `memory` should be `calldata`, otherwise `calldata` modifiers of `updateRoots` parameters does not make sense.

Listing 34: Missed modifier

```
81  function updateWithdrawalTree(TreeLeaf[] memory _withdrawals) public {
```

3.35 CVF-35 The name suggestion

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

Recommendation Perhaps,the `areRecentRoots` name would be better.
Client Comment The `areRecentRoots` name assumes that it should return a boolean value. In our case the function reverts if input data is incorrect and returns nothing otherwise.

Listing 35: The name suggestion

```
100 function validateRoots(bytes32 _depositRoot, bytes32
_withdrawalRoot) public view {
```

3.36 CVF-36 The name suggestion-2

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

Recommendation The `withdrawals` variable does not replace `withdrawals` it would be better to name it `newWithdrawals`.

Listing 36: The name suggestion-2

```
121  function getRegisteredWithdrawals() external view returns(bytes32[]
memory _withdrawals) {
```

3.37 CVF-37 Public function

- **Severity** Moderate
- **Status** Info
- **Category** Suboptimal
- **Source** TornadoTrees.sol

Recommendation the `blockNumber()` function should be internal.
Client Comment We override it in TornadoTreesMock for testing. We will leave it as is.

Listing 37: Public function

```
129  function blockNumber() public view virtual returns (uint256) {
```

3.38 CVF-38 The confusing interface name

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

Description the `ITornadoTrees` interface name looks confusing. There is nothing related to trees inside. Trees are internal implementation details of the particular implementation.
Recommendation Consider renaming to something like `ITornadoRegistry`.

Listing 38: The confusing interface name

```
5   interface ITornadoTrees {
```

3.39 CVF-39 The redundant word "New"

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

Description the `new` word is probably redundant, as every deposit could be registered at most once.

Listing 39: The redundant word "New"

```
6   function registerNewDeposit(address instanse, bytes32 commitment)
    external;
8   function registerNewWithdrawal(address instanse, bytes32 nullifier)
    external;
```

3.40 CVF-40 The typo

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

Description There is a typo in the line.
Recommendation There should be `instance` instead of `instanse`

Listing 40: The typo

```
6   function registerNewDeposit(address instanse, bytes32 commitment) external;
7
8   function registerNewWithdrawal(address instanse, bytes32 nullifier) external;
```

3.41 CVF-41 The complicated check

- **Severity** Moderate
- **Status** Info
- **Category** Suboptimal
- **Source** MerkleTreeWithHistory.sol

Description In the

`insertIndex + _leaves.length < uint32(2)**levels`

inserted index would go down from

`2^levels - 1 to zero,`

rather than go up, then this check would look like

`(insertIndex >= _leaves.length).`

Also, other parts of the code would be simpler.

Listing 41: The complicated check

```
76  require(insertIndex + _leaves.length < uint32(2)**levels, "Merkle
    doesn't have enough capacity to add specified leaves");
```

3.42 CVF-42 Gas efficient

- **Severity** Moderate
- **Status** Fixed
- **Category** Suboptimal
- **Source** MerkleTreeWithHistory.sol

Description the `bytes32[] memory subtrees = filledSubtrees` reads all subtrees into memory, including those that are currently not used or will not be needed in the loop below. Probably, reading necessary values directly from textttfilledSubtrees only when need would be more gas efficient.
Recommendation Consider refactoring.

Listing 42: Gas efficient

```
78  bytes32[] memory subtrees = filledSubtrees;
```

3.43 CVF-43 Suboptimal loop

- **Severity** Moderate
- **Status** Info
- **Category** Suboptimal
- **Source** MerkleTreeWithHistory.sol

noindent**Description** the next loop

```
_leaves.length - 1
```

recomputes the inner tree hashes for each leaf, whereas it would be much more efficient to recompute the entire tree based on the new leaves.
Recommendation Consider refactoring.

Listing 43: Suboptimal loop

```
80  for (uint32 j = 0; j < _leaves.length - 1; j++)
```

3.44 CVF-44 Suboptimal Condition

- **Severity** Moderate
- **Status** Fixed
- **Category** Suboptimal
- **Source** MerkleTreeWithHistory.sol

Description the next condition `i < level84s` will never be false, as the loop is always exited via break statement.
Recommendation Consider removing this conditions to make code more readable.

Listing 44: Suboptimal Condition

```
55 84 96 for (uint32 i = 0; i < levels; i++) {
```

3.45 CVF-45 The suboptimal index

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

Description Perhaps, `»=1` would be more efficient and more clear.

Listing 45: The suboptimal index

```
92  index /= 2
```

3.46 CVF-46 Redundant loop

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

Description The `uint32 i = 0; i < levels; i++`loop is probably redundant,
as https://eips.ethereum.org/EIPS/eip-1283 makes it very cheap to overwrite storage slot with the same value.
Client comment We tried it in remix, and overwriting a storage slot with the same value costs 800 gas (1 SLOAD).

Listing 46: Redundant loop

```
97  // using local map to save on gas on writes if elements were not modified
```

3.47 CVF-47 Suboptimal delegating

- **Severity** Moderate
- **Status** Open
- **Category** Suboptimal
- **Source** MerkleTreeWithHistory.sol

Description Delegating the rest to `insert` makes code harder to read and is probably suboptimal.
Recommendation Consider implementing self-contained version of `bulkInsert` that calculates hashes from bottom to top rather then from left to right.
Client comment

Listing 47: Suboptimal delegating

```
5   interface ITornadoTrees {
```

3.48 CVF-48 Confusing function name

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

Description the `updateInstances` name of function is confusing. It actually adds/removes instance from the set of valid instances, rather than "updates" it.
Recommendation Consider splitting into two functions: `addInstance` and `removeInstance` or renaming to `setInstanceStatus`.
Client comment

Listing 48: Confusing function name

```
39    function updateInstances(ITornado instance, bool update) external
onlyGovernance {
```

3.49 CVF-49 Misleading file name

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

Description the word "float" is misleading, as the library actually implements fixed point, rather than floating point arithmetic's.
Recommendation Consider file renaming.
Client comment

Listing 49: Misleading file name

```
7    import "./utils/FloatMath.sol";
```

3.50 CVF-50 Expensive deployment

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

Description the `resolve(miner)` just makes deployment more expensive and less convenient in DEV environment.
Recommendation Consider passing Miner address directly.
Client comment

Listing 50: Expensive deployment

```
51    miner = resolve(_miner);
```

3.51 CVF-51 Inefficient function

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

Description the `exp2` function is slightly more efficient.
Recommendation Consider using it instead of exp. This will require adjusting `poolWeight` value accordingly.
Client comment The gas reduction is too small, it's not worth making the code less readable for that.

Listing 51: Inefficient function

```
71  int128 exp = FloatMath.exp(pow);
```

3.52 CVF-52 Index return absence

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

Description there is no return for the `ITornadoTrees` function. It should probably return the index of the first inserted leaf.
Recommendation Consider renaming to something like `ITornadoRegistry`.
Client comment We think it will be confusing for the caller why he submitted array of items and got only a single number in return

Listing 52: Index return absence

```
14  function bulkInsert(bytes32[] calldata leaves) external
onlyOwner {
```

References

[1] *Solidity Documentation*
https://docs.soliditylang.org/en/v0.6.0/060-breaking-changes.html