TokenTable Audit Report

(Created by EthSign for Notcoin token airdrop)

TONTECH VERIFIED ON 30 JUL 2024 | Report revision 7 | FINAL

01 Summary

Total findings	Resolved	Acknowledged	Declined	Unresolved
21	20	1	0	0

The findings were classified by the severity levels as following:

Critical	Major	Medium	Minor	Informational
3	3	5	5	5

02 Severity levels

Critical

These are the most severe security flaws in a smart contract. They can lead to immediate and significant risks, allowing attackers to gain unauthorized access, bypass security mechanisms, or cause substantial financial losses. Critical vulnerabilities require immediate attention and remediation.

Major

These vulnerabilities are significant but may not pose an immediate threat or have a lower probability of exploitation compared to critical vulnerabilities. However, they still have the potential to result in significant damages or security breaches if left unaddressed. They should be resolved in a timely manner to ensure the overall security of the smart contract.

Medium

Medium vulnerabilities indicate security weaknesses that may not have a high likelihood of exploitation or immediate serious consequences. However, they can still impact the overall security posture of the smart contract and should be fixed to enhance its resilience against potential attacks or vulnerabilities.

Minor

These vulnerabilities are less critical in nature and generally have a limited impact on the security of the smart contract. While they may not pose an immediate threat, they should not be overlooked, as they can compound with other vulnerabilities or serve as a stepping stone for potential attackers. Addressing them is important to maintain a robust security posture.

Informational

Suggestions for code style, architecture, and optimization to improve quality and efficiency. Not security risks, but enhance resilience.

03 Codebase

Repository (`EthSign/tokentable-tvm`)

Actual code revision commit (`eaa674963ccff770582a235029789eaa973fb1f0`)
Original commit (`7e4b247b1082de21293c6e1f5c576ca2ea7ef284`)

External code library (`@ston-fi/funcbox v0.0.13`)

04 Scope

Source code files:

File paths	Rollup SHA-256 hash of the files
`contracts/*.fc` `contracts/*/*.fc`	`a31ba7764e802a556661d4313900c425 447755de9f97db4d4d259ede06220dbc`
`node_modules/@ston-fi/funcbox/*.fc`	`88ff7ccfc8013cc8c0f1612a4bb0f5dc 02319e6974261df70ad921937e5b8f3f`

File paths	Rollup SHA-256 hash of the files
`node_modules/@ston-fi/funcbox/*/*.fc`	9e2cc4870c0f5f8f747b8fc2ae454d93da14cd0184a809491471265d907735a6
`node_modules/@ston-fi/funcbox/*/*/*.fc`	20e9776d6027019e741cdb7e321e1d8fecb6175fb0f2e186c4d1d07c6ad2f804
Original commit `contracts/*.fc` `contracts/*/*.fc`	a5bae2f97e29f543cf15a7aeda95f660e60267457c33adcdc4f502ce00dcd858

Rollup is obtained by calling `sha256sum <paths> 2>/dev/null | sha256sum` For example, the result of `sha256sum contracts/*.fc contracts/*/*.fc 2>/dev/null | sha256sum` call is used to obtain the first hash.

Resulting hashes **after code updates and fixes** (compiled with `func 0.4.4`):

Smart contract	Compiled code cell hash
`merkle-token-distributor.fc`	939f98627ee129c6a3d0ee1f5b188a3e88df914d5268f631a98940fbf2e2bdef
`leaf.fc`	473f004aee1730342f1aa3affa06f39862af9e6d1eacff8cee6a3f525a6e5781

Resulting hashes of original (first) code revision (compiled with `func 0.4.4`):

Smart contract	Compiled code cell hash
`merkle-token-distributor.fc`	62b61a79f745442681742b6991ff7fa668404d3f4d2c31b4bc4a2d20591af493
`leaf.fc`	f566c61909fcd5f55a4b4105f4ff0e732056a0acb176a23e1c2e09ffb7a0b475

05 Issues

Notice about gas credit and gas usage constant

Category	Severity	Location	Status
Limitation	Informational	None	Acknowledged

Description

Be advised that due to implemented checks and because of `gas_consumption` constant not taking into account depth or size of the proof, it is advised to limit amount of dictionary elements per single contract.

Recommendation

The smart contracts were extensively tested with 1 million dictionary entries, therefore it is advised not to exceed this amount of items in a single contract, and use multiple contract instances, if necessary.

Acknowledgment

Developers and the customer correctly understand the safety limitations of the smart contract, and will use it with about 300 - 400 thousand items per a single contract, keeping it safe from maximum tested 1 million.

In leaf to distributor message used variable can be inlined

Category	Severity	Location	Status
Optimization issue	Informational	`leaf.fc` L100-102	Resolved

Description

A single bit `used` is stored in a separate cell, although it can be easily inlined into parent cell.

Recommendation

Move `used` variable into the parent cell and update corresponding deserialization code in Distributor.

Resolution

The suggested recommendations were implemented in the code.

Multiple issues with leaf deployment flow

Category	Severity	Location	Status
Logic issue	**Major**	`merkle-token-distributor.fc`	Resolved

Description

Right now Leaf is deployed by calling `deploy_leaf` operation on the Distributor. This deploys leaf with attaching incoming message value (mode 64) and 0.03 TON on top of that, which causes the Distributor SC to be drained of funds with Leaf deployments.

Moreover, there is a window of opportunity after the leaf is deployed and potential attacker now knows and observes leaf contract, and arrival of actual external message from the user, that allows attacker to freely spam external messages and prevent user from claiming reward.

Recommendation

It is advised to remove Leaf deployment operation on Distributor altogether, and, instead, directly send necessary value to uninitialized Leaf contract directly, and then attach StateInit (possible, automatically) when calling `claim` operation on the Leaf. In such scenario, window of opportunity is extremely narrow for an attacker, and requires them to have access to mempool for some chance of success.

Resolution

The issue was (mostly) resolved, now the leaf is prefilled (and deployed) by the user, and, afterward, external message is sent to the Leaf. For some reason, difficulties arisen when trying to deploy SC with external message.

Since in client logic external message is sent immediately after deploying the leaf, possible attack does not give any profit to the attacker, and is made as difficult as possible by the resolution of the next issue, attack probability is now considered to be negligible.

Leaf can be still blocked from claiming rewards (again)

Category	Severity	Location	Status
Logic issue	Medium	`leaf.fc`	Resolved

Description

Even with all proper checks it is still possible, in theory, to permanently block the leaf from receiving the reward by providing a specifically trimmed proof, that would consume a large chunk of Leaf to Distributor message value for forwarding fees, and, therefore, Distributor will not be able to send back unlock or bounce, because they will get out of gas error during processing.

Recommendation

If the amount of items in the tree is below 1 million (consequently, the dictionary depth in such case is 20, therefore maximum depth of the honest proof is 21 that includes 42 cells including pruned branches) the gas credit is right enough to calculate depth and data size of the proof. Therefore, to make sure, that the proof is not excessively bloated, it is recommended to compare depth against amount of cells in proof, and make sure that it is no more than 2 * depth. To allow for some lax in case of possible data overhead, it is acceptable to add 1 or 2 to max limit of cells.

Therefore, it is possible to implement the check by adding the following code before `accept_message()` (function `compute_data_size` will throw cell overflow exception (8) if specified amount of cells is exceeded), that also does some basic checks for proof correctness, that are possible to do in Leaf:

```
(slice cs, int exotic?) = proof.begin_parse_exotic();
throw_unless(error::is_proof_exotic, exotic?);
throw_unless(error::tree_exceeded, cs~load_uint(8) == 3);

proof.compute_data_size(2 * proof.cell_depth() + 2);
(_, int found?) = cs~load_ref().udict_get?(256, index);
throw_unless(error::not_found, found?);
```

The code was tested to fit into external message gas credit for 1M dictionary entries (depth 20). With such checks attack would be very difficult to carry out, and without any profit for the attacker can be considered of negligible probability.

Resolution

The recommended code with checks was fully added, gas usage was retested.

Value flow management issue

Category	Severity	Location	Status
Logic issue	Medium	`merkle-token-distributor.fc L82-L112`	Resolved

Description

Calculating remainder of message value to send by subtracting some values from `msg_value` is not a correct practice, because on one had that does not cover variable gas used by calculation, and on the other hand taking 0.05 TON for storage from each transaction is a little too extreme (1 TON will accumulate from each 20 claims). Analyzing that two constants are provided, min_tons_for_storage and master_contract_storage_fee, most likely it is expected for the contract to have 0.05 TON for storage at all times.

Recommendation

To make sure that the contract has 0.05 TON remaining after each transaction, it is advised to reserve these 0.05 TONs on the balance first, then send claim fee to the appropriate contract, and then send all remainder (mode 128) to jetton wallet for further processing.

In case of failure, no changes are required - just send back unlock message with mode 64 like in the current code.

Resolution

Now the logic precisely follows the recommendations.

Incorrect handling of contract balance value

Category	Severity	Location	Status
Logic issue	Medium	`merkle-token-distributor.fc` L284	Resolved

Description

Balance already contains the incoming message value, and, therefore, `total_bal` variable will contain `msg_value` accounted for twice.

This allows for user to supply less than required value, and remainder will be taken from the balance of the smart contract.

Recommendation

Remove the `total_bal` variable and just use `balance` instead.

Resolution

The erroneous variable was removed, and `balance` is now used directly.

Jetton forward amount may be excessive

Category	Severity	Location	Status
Logic issue	Medium	`merkle-token-distributor.fc` L109	Resolved

Description

When sending tokens, `forward_ton_amount` of 0.01 TON is used, that may be excessive in most cases.

Recommendation

Unless it is required to trigger other smart contracts with the claimed jetton, it is advised to decrease it to 1 (nano-ton unit).

Resolution

The value of `forward_ton_amount` was decreased to 1 nano-ton unit.

Fixed fee values are PUSHINT ASMs instead of constants

Category	Severity	Location	Status
Optimization issue	Informational	`utils.fc`	Resolved

Description

Fixed-value fee constants are written in code as PUSHINT ASM functions instead of integer constants. This prevents compiler from pre-evaluating const expressions and performing other optimizations in code, thus leading to less effective and more bloated TVM contract code.

`unlock_back_fee` also seems to be quite out of place (in `leaf.fc`, where other fee constants are in `utils.fc`).

Recommendation

It is advised to migrate constant values from `asm "... PUSHINT"` to `const int` to enjoy compiler optimizations.

Also, consider moving `unlock_back_fee` to `utils.fc` for less logic fragmentation.

Resolution

The suggested recommendations were fully implemented in the code.

06 Previous issues

Unrestricted external message processing on Distributor

Category	Severity	Location	Status
Logic issue	**Critical**	`merkle-token-distributor.fc` L328 - L343	Resolved

Description

It is possible to execute `op::claim` external message without any ownership or other restrictions (apart from simple seqno reply protection). Therefore, it is possible to very easily drain the TON balance of the contract by calling this function many times. Moreover, each successful invocation also sends a fixed (0.2 TON) amount to Leaf contract to invoke `op::deploy_leaf` (using an unlogical mode 64, since external messages do not carry a value). While, afterward, it should return to the Distributor contract, it still loses computation, forward and deployment costs.

Also, the contract will be practically unusable, because only one external message with specific seqno can be processed in a single block, therefore, huge concurrency on calling the `op::claim` external message will make it unusable.

Recommendation

It is advised to move external message trigger logic back to Leaf, therefore the user would send some TONs to Leaf, and calls `op::claim` external message along with deploying the Leaf itself (with logic similar to `op::deploy_leaf` internal message).

While the current implementation is the best solution if it would be possible to deploy Leaf and call `op::deploy_leaf` with an internal message from the user's wallet, however, if there is still issue with that, it is advised to not perform external messages on a shared common contract, and do it on Leaf instead.

Resolution

Claim message processing logic was moved back to the Leaf smart contract.

Leaf can still be blocked from claiming rewards

Category	Severity	Location	Status
Logic issue	**Major**	No specific location	Resolved

Description

Despite the carried out measures, it is still possible to permanently block Leaf from receiving rewards by deploying it and calling `op::deploy_leaf`. If the balance (and, consequently, amount of carried TONs in the message) is low enough, the message will be successfully sent to Distributor, but will fail processing with error code -13 (out of gas), and, in this special case, `try` will not help to deal with this problem.

Moreover, even if processing succeeds, transferred value may not be enough for claim fees and jetton processing costs.

Recommendation

Before sending a message to Distributor, make sure that the contract balance is high enough to, at least, successfully process the incoming message and send unlock reply, if necessary. This may be quite tricky because traversing dictionary gas depends on its depth, therefore, it may be reasonable to account for `cell_depth` or `CDEPTHI` of the proof.

As for the fees and jetton costs, it is advised to check that enough value was provided in Distributor before calling send_tokens (make sure it is enough for gas (estimate) + claim fee + jetton fees), and throw (inside `try`) if the value is not enough - see `int msg_value`.

Resolution

Relevant checks were added to both contracts. As for the depth, gas was tested for dictionaries with 1M entries, and it is expected that each instance of the contract will have dictionaries with no more than 350k entries because of related limitations not related to smart contracts.

Unused data in unlock message

Category	Severity	Location	Status
Logic issue	Minor	`merkle-token-distributor.fc` L309	Resolved

Description

The `proof` that is sent in the message is not used in actual unlock message processing on Leaf, but it can be quite heavy.

Recommendation

Remove the unnecessary `.store_ref(proof)`.

Resolution

The unnecessary store call was removed.

Different proofs can be supplied for same index

Category	Severity	Location	Status
Logic issue	**Critical**	No specific location	Resolved

Description

It is possible to assemble different merkle proofs for a specific dictionary entry index by keeping or pruning different branches not related to the target index entry. For example, if it is necessary to use left branch of a tree to reach the target index entry, and the right branch is pruned, it is possible to keep it, and prune its child cells instead.

This will result in different merkle proof that will have different hash, but still can be used to claim tokens since it is still possible to traverse such proof to reach the target dictionary entry.

Recommendation

It is advised to index Leaf elements only by index, and do not include proof hash in state init data of the contract. Instead, similarly to the next recommendation, lock the Leaf (`used = 1`) when sending proof to Distributor, and, if the provided proof is incorrect for any reason, reply with the *unlock* message to the Leaf instead of throwing.

It may be advisable to wrap work with proof exotic cell into `try { ... } catch` construct to catch possible deserialization errors, that may help with this and next issues.

Resolution

The issue was resolved by indexing Leaf only by Distributor address and index (without the proof hash). Also, `try` construct and unlock message support and flow were added.

Leaf can be blocked from claiming rewards

Category	Severity	Location	Status
Logic issue	**Major**	No specific location	Resolved

Description

If Leaf is deployed before `start_time` and `claimable_timestamp` conditions are met, then the message will fail on Distributor, and Leaf will be left with `used = 1` state unable to claim reward afterward. Most importantly, anyone can deploy and call a leaf for any recipient at any time, even before time conditions are met, effectively preventing recipient from claiming their reward forever.

Recommendation

If claiming is not possible due to start time conditions not satisfied, it is advised to send a special *unlock* message to the Leaf carrying all incoming balance (mode 64) that would set its `used` value back to 0.

This would not open possibility for double claiming because immediately, upon sending, `used` is set to 1, and it is returned back to 0 only after *unlock* message arrives. Therefore, the user would not be able to send any more messages during this period.

Resolution

The issue was resolved by wrapping Leaf message processing logic in Distributor with `try` construction and sending proper `op::unlock` message if any issue except for already used Leaf happens.

Sender address is read from the message body

Category	Severity	Location	Status
Logic issue	**Critical**	`merkle-token-distributor.fc` L135	Resolved

Description

The value of `sender_address` is read from `slice in_msg` (last parameter of `recv_internal`) that is the message body.

```
() recv_internal(int balance, int msg_value, cell in_msg_cell, slice in_msg) {
  ...
  slice sender_address = in_msg~load_msg_addr();
  ...
}
```

The problem is that `in_msg` can be completely manipulated by the sender, that is, any contract can send internal message with arbitrary value of `in_msg`, and, consequently, the value of `sender_address`.

The issue is critical, because this `sender_address` is used later in code to check for admin permissions by comparing it against the `admin_address` for various actions, including code modification and arbitrary withdrawals.

```
() recv_internal(int balance, int msg_value, cell in_msg_cell, slice in_msg) {
  ...
  throw_unless(error::unauthorized, equal_slices(sender_address,
admin_address));
  ...
}
```

Therefore, **any** user (contract) can call these functions by just putting `admin_address` at the correct position in the incoming message.

Recommendation

Properly retrieve message flags and sender by extracting them from `cell in_msg_cell` that contains the full message cell, including the message headers, from which flags and message sender **must** be extracted.

As an example, proper reading of flags and message sender can be implemented like the following:

```
() recv_internal(int balance, int msg_value, cell in_msg_full, slice in_msg) {
  if (in_msg.slice_empty?()) { ;; ignore all empty messages
    return ();
  }

  slice in_msg_full_slice = in_msg_full.begin_parse();
  int flags = in_msg_full_slice~load_uint(4);

  if (flags & 1) { ;; ignore all bounced messages
    return ();
  }

  slice sender_address = in_msg_full_slice~load_msg_addr();

  int op = in_msg~load_uint(32); ;; by convention, the first 32 bits of incoming
message is the op
  int query_id = in_msg~load_uint(64); ;; also by convention, the next 64 bits
contain the "query id", although this is not always the case
  ...
}
```

Please note, that it is recommended to rename `in_msg_cell` to `in_msg_full` for clarity.

Also pay attention that this change will require removal of unnecessary 4-bit *flags* and *message sender* from the structure of the internal message. Therefore, this recommended change will also fix a minor issue regarding the message structure that is outlined later in the document.

Resolution

The issue was resolved by reading fields from proper places with the code similar to the provided example.

End time is not enforced

Category	Severity	Location	Status
Logic issue	Medium	`merkle-token-distributor.fc`	Resolved

Description

While `end_time` variable is present in the contract, it is not enforced at all and is only returned in the getter.

The problem surfaces as no TONs will remain on balance of Leaf after the operation (because of send mode 128). Consequently, after the Leaf contract accrues over 1 TON of debt it will be deleted from the blockchain, and it will be possible to claim the drop again.

Recommendation

Add proper checks for `end_time` when performing the claim. For example:

```
throw_unless(error::time_inactive, now() <= end_time);
```

Then, with reasonable `end_time`, double claim scenario will not be possible, because 1 TON debt will take a very long time to accumulate.

Resolution

The issue was resolved by adding a proper check as in the example.

Start time is not enforced correctly

Category	Severity	Location	Status
Logic issue	Minor	`merkle-token-distributor.fc` L309	Resolved

Description

The check for `start_time` is not correct in its logic, and may result in incorrect behavior.

```
throw_unless(error::time_inactive, ((claimable_timestamp <= now()) |
(claimable_timestamp <= start_time)));
```

This check succeeds if current time is after `claimable_timestamp` of a specific entry of the user, OR if global `start_time` is after `claimable_timestamp` or equals to it. Therefore, entries, that have their `claimable_timestamp` less than `start_time`, will be claimable immediately even before the both timestamps.

Recommendation

Replace the check with proper logical condition or explain the strange code logic. For example:

```
throw_unless(error::time_inactive, ((claimable_timestamp <= now()) & (start_time
<= now())));
```

Resolution

The issue was resolved by adding the proper checks:

```
throw_unless(error::time_inactive, claimable_timestamp <= now());
throw_unless(error::time_inactive, start_time <= now());
```

Internal message bounce is checked incorrectly

Category	Severity	Location	Status
Logic issue	Minor	`merkle-token-distributor.fc` `L123 - L131`	Resolved

Description

The bounce check is implemented incorrectly by reading 4 bits from the *body* of the message. This issue is closely related to the critical issue `Sender address is read from the message body`, please see that issue for the complete code example.

Recommendation

Please see recommendation of the aforementioned issue that also fixes this problem.

Resolution

The check was fixed, the bits are now read from the correct cell (slice).

Internal message structure violates standards

Category	Severity	Location	Status
Standards violation	Minor	`merkle-token-distributor.fc` `L127 - L134`	Resolved

Description

The internal message contains unnecessary 4-bit *flags* prefix, but must immediately start with 32-bit opcode by TON standards. Therefore, internal messages directed to this contract will not be able to be correctly parsed by explorers or wallets because of these extra bits.

This issue is closely related to the critical issue `Sender address is read from the message body`, please see that issue for the complete code example.

Recommendation

Please see recommendation of the aforementioned issue that also fixes this problem.

Resolution

The offending prefix was removed, now internal messages immediately start with 32-bit opcode field.

Unused variable in logic: claim delegate

Category	Severity	Location	Status
Unused variable	Minor	merkle-token-distributor.fc	Resolved

Description

Variable `claim_delegate` is defined, can be retrieved with getter, can be modified with admin updates, but does not actually affect anything. Most likely some code logic was not implemented.

Resolution

The unused variable `claim_delegate` was removed from the code.

Recommendation

Implement the relevant missing logic or explain why that variable should not be used in actual contract logic.

Suspicious unused variables

Category	Severity	Location	Status
Logic issue	Minor	leaf.fc	Resolved

Description

For some reason `recv_external` contains many unnecessary elements in the message such as 4-bit *flags* and sender address (that is unused, therefore read result is discarded by compiler). While not as standardized as internal message layout, the variables do not carry and functional logic.

Empty message check is also unnecessary, since external message is not processed unless it is explicitly accepted.

```
() recv_external(slice in_msg) impure {
  if (in_msg.slice_empty?()) { ;; ignore all empty messages
    return ();
  }

  int flags = in_msg~load_uint(4);

  if (flags & 1) { ;; ignore all bounced messages
```

```
    return ();
  }

  int op = in_msg~load_uint(32); ;; by convention, the first 32 bits of incoming
message is the op
  int query_id = in_msg~load_uint(64); ;; also by convention, the next 64 bits
contain the "query id", although this is not always the case
  slice sender_address = in_msg~load_msg_addr();
  ...
}
```

Recommendation

Remove the unneeded variables. Note that this will require adjustment of external messages structure.

As an example:

```
() recv_external(slice in_msg) impure {
  int op = in_msg~load_uint(32); ;; by convention, the first 32 bits of incoming
message is the op
  int query_id = in_msg~load_uint(64); ;; also by convention, the next 64 bits
contain the "query id", although this is not always the case
  ...
}
```

Resolution

The code was updated as per recommendation, all unnecessary fields and variables were removed.

Additional value carried with mode 64

Category	Severity	Location	Status
Logic issue	Minor	`merkle-token-distributor.fc` L104	Resolved

Description

An additional value of 0.01 TON is attached to messages in addition to value of the inbound message as per mode 64.

```
() send_tokens(int query_id, slice recipient, int amount) impure {
    ...
    send_raw_message(begin_cell()
        .store_uint(0x18, 6)
        .store_slice(token)
        .store_coins(10000000)
        .store_uint(1, 1 + 4 + 4 + 64 + 32 + 1 + 1)
        .store_ref(begin_cell()
            .store_uint(op::jetton_transfer, 32)
            .store_uint(query_id, 64)
            .store_coins(amount)
            .store_slice(recipient)
            .store_slice(recipient)
            .store_uint(0, 1)
            .store_coins(10000000)
            .store_uint(0, 1)
            .end_cell())
        .end_cell(), 64);
}
```

Therefore, for each such message 0.01 TON will be taken from the SC's balance in addition to value of the message.

Recommendation

Make sure that these extra TONs are necessary, remove them or explain why it is needed, accordingly.

Resolution

Coins amount was changed to 0 in such situations (where message is sent with mode 64).

External code library reference is not specific

Category	Severity	Location	Status
Weak reference	Informational	`merkle-token-distributor.fc` L1	Resolved
		`leaf.fc` L1	
		`package.json` L10	

Description

A part of code is included from a Node.js library reference, but the reference itself is done using a caret range, that may cause inclusion of different library version.

```
merkle-token-distributor.fc
 1  #include "../../node_modules/@ston-fi/funcbox/autoload.fc";

leaf.fc
 1  #include "../../node_modules/@ston-fi/funcbox/autoload.fc";

package.json
 9      "devDependencies": {
10          "@ston-fi/funcbox": "^0.0.13",
```

Recommendation

While being reasonably specific when referencing a patch version, it may cause unintended changes to the code when the library version referenced is at least ^0.1.0, therefore it is advised to use exact `=` version reference in the future.

Resolution

The caret range was replaced with a specific version:

```
"@ston-fi/funcbox": "0.0.13",
```

Tests are not passing

Category	Severity	Location	Status
Tests problem	Informational		Resolved

Description

Some test cases are failing. (All tests cases related to claiming tokens). The developers team claims that functionality works on testnet, but the failing tests are still an issue that should not be forgotten about.

```
FAIL  tests/MerkleTokenDistributor.spec.ts (18.492 s)
 MerkleTokenDistributor
   ✓ should deploy (2375 ms)
   ✓ should set base params (2259 ms)
   ✓ should withdraw (2215 ms)
   × should claim one time (2336 ms)
   × should claim many times (2320 ms)
   × should not claim if already did (2319 ms)
   × should not claim with wrong index (2486 ms)
```

Recommendation

It is recommended to review the test cases or report the issue to TonTech if the issue is triaged to be originating from the Blueprint / Sandbox itself.

Resolution

The tests were fixed, and now complete successfully:

```
PASS  tests/MerkleTokenDistributor.spec.ts (30.495 s)
 MerkleTokenDistributor
   ✓ should deploy (2316 ms)
   ✓ should set base params (2173 ms)
   ✓ should withdraw (2145 ms)
   ✓ should withdraw tokens (2164 ms)
   ✓ should claim one time (2258 ms)
   ✓ should claim many times (12483 ms)
   ✓ should not claim if already did (2289 ms)
   ✓ should not claim with wrong index (2408 ms)
```