Tornado Governance and Token Smart Contracts. Audit

Mikhail Vladimirov and Dmitry Khovratovich

23rd October 2020

This document describes the audit process of the Tornado Governance and Token smart contracts performed by ABDK Consulting.

1. Introduction

We've been asked to review the Tornado Governance and Token smart contracts given in separate files.

2. Governance

In this section we describe issues found in the Governance.sol.

2.1 Moderate Issues

This section lists moderate issues, which were found in the smart contract.

1. Line 190: the `delegatecall` allows target contract to make arbitrary modifications in Governance's storage, which could be dangerous. One way to address this issue is to move critical parts of Goverance's storage into a separate contract, that will allow Governance and only Governance to modify values.This separate contract could maintain modification counter. So, before doing a delegate call, Governance could remember the address of that storage contract (in case it is mutable) and the value of the modification counter. After the delegatecall, Governance could check that neither address to the storage contract nor modification counter changed, and revert in case they did.

Authors' comment: *This is an intended behavior. We avoid any calldata for proposals so it will be more readable for users*

2. Line 195, 197: the `data.length > 0` condition and `else` reverts the whole transaction, thus the proposal will not be marked as executed. Consider marking the proposal as failed instead of reverting the transaction.

Authors' comment: *This is intended behavior. If execution fails for some time the proposal will eventually become expired.*

2.2 Suboptimal Code

This section lists suboptimal code patterns, which were found in the smart contract.

1. Line 78, 89: it should be cheaper to have two separate events: one for vote and another for vote revocation instead of one `support`.
2. Line 91: setting torn to a dead address doesn't guarantee that the contract will be fully inoperable. There could be control paths to self destruct that still work even when torn refers to a dead address. Actually, there are no self destruct calls in the contract, so, probably, the protection is redundant.
3. Line 125: the `propose` function doesn't allow using delegated voting power to exceed the proposed threshold. Consider implementing such ability.
4. Line 191: there is no way to pass additional parameters to the called contracts, thus a new contract has to be deployed to almost every proposal. Consider implementing an ability to pass additional parameters along with delegate calls.
5. Line 208: there is no way to revoke current vote by the `_castVote` function without casting a new one. Consider implementing such ability.
6. Line 220: the current vote is reverted even if the new vote is the same as the current vote.
7. Line 244: the next code `_lockTokens(voter, proposal.endTime.add(EXECUTION_DELAY));` should be done only when a proposal was extended.
8. Line 245: the `VoteCast` event doesn't reflect the previous vote that was probably revoked by the call. Consider emitting a separate event in case reverted previous vote.

FIXED:

1. Line 78, 89: the next parameters should be indexed:
   - `proposer`
   - `voter`
   - `proposalId` (and should go first)
   - `id`

2.3 Other Issues

This section lists other minor issues which were found in the token smart contract.

1. Line 76, 89: events are usually named via nouns. For example `ProposalExecuted` could be `ProposalExecution` or `Execution`.
2. Line 264, 266: there should be probably `<` instead of `<=`.

FIXED:

Line 86: events are usually named via nouns. For example, `Vote` for the `VoteCast`, `NewProposal` or `ProposalCreation` for the `ProposalCreated`.

3. TORN

In this section we describe issues found in the torn.sol.

3.1 Fixed Moderate Flaws

This section lists moderate flaws, which were found in the smart contract.

Line 28: the next check `totalSupply() == 0` is unreliable as tokens are burnable. Thus it is possible to reinitialize TORN after burning all the tokens. More reliable check would be:

```
require (_governance != address (0));
require (governance == address (0));
```

3.2 Suboptimal Code

This section lists suboptimal code patterns, which were found in the smart contract.

1. Line 22: seems like the `Recipient` data structure is not used anymore.
2. Line 47: the function name `changeTransferability` looks cumbersome and uncommon. Two functions: `pause`/`unpause` or a single function the `setPaused` would look more conventional.
3. Line 58, 65: the next events:
   - `Allowed`
   - `Disallowed`

   is emitted even if the address wasn't in the allowed list.
4. Line 75: the next requirement
   `require(!paused() || allowedTransferee[from] || allowedTransferee[to], "TORN: paused")`
   could be simplified with the `whenNotPaused` modifier.
5. Line 85: the next check `_to != address(0)` is redundant. It is anyway possible to send tokens to the dead address.
6. Line 90, 95: the next formula `_balance == 0 ? totalBalance : Math.min(totalBalance, _balance)` looks like unnecessary complication. The contract becomes more predictable if ether sends as much as the user asked.

FIXED

1. Line 94: note that since `_token` is untrusted, `balanceOf` may not be a view function but can execute arbitrary code.

3.3 Other Issues

This section lists other minor issues which were found in the token smart contract.

1. Line 19: events are usually named via nouns. Here some examples:
   - `Permit` for the `Allowed`
   - `PermitRevocation` for the `Disallowed`

FIXED:

2. Line 14: the word `token` in a name of token is probably redundant.
3. Line 15, 81: there should be `IERC20` instead of `ERC20`.

4. Vesting

In this section we describe issues found in the vesting.sol.

4.1 Suboptimal Code

This section lists suboptimal code patterns, which were found in the smart contract.

1. Line 29: using months (or, strictly speaking 30-days intervals) as a time unit for cliff and durations reduces the contract's flexibility. Consider using seconds instead.
2. Line 51: the next check
   `require(_beneficiary != address(0), "Beneficiary cannot be empty")`
   looks redundant. It is anyway possible to specify a dead beneficiary address.

FIXED:

1. Line 69: the `released` variable is used without being initialized.

4.2 Fixed Other Issues

This section lists other minor issues which were found in the token smart contract.

Line 21: events are usually named via nouns. Here some examples:
- `Release` for the `Released`
- `Revocation` for the `Revoked`

5. ERC20Permit

In this section we describe issues found in the ERC20Permit.sol.

5.1 Suboptimal Code

This section lists suboptimal code patterns, which were found in the smart contract.

1. Line 31: the `_updateDomainSeparator` function is redundant. The domain separator will anyway be updated on first call to `permit(...)`.
2. Line 58-59: the next line

```
_nonces[owner]++;
_approve(owner, spender, amount);
```

could be done when calculating `hashStruct` to avoid reading `_nonces[owner]` from the storage for the second time.

3. Line 65: the `nonces` getter would not be necessary if the function would be public.
4. Line 72-80: it would be cheaper to concatenate everything and hash at once.

5.2 Other Issues

This section lists other minor issues which were found in the token smart contract.

Line 59: the returned value is ignored. Probably not an issue.

6. Signatures.sol

Several issues were found in the Signatures.sol but the contract was later deprecated.

7. Constants.sol

In this section we describe issues found in the Constants.sol.

7.1 Major Flaws

This section lists major flaws, which were found in the smart contract.

Line 47, 51: the next parameters: the `executionExpiration` and the `proposalThreshold` not range checked. Setting it above TORN total supply would lock up Governance. Consider adding an explicit check.

FIXED

Line 39, 43: the `executionDelay` parameter is not range checked. Setting it above `EXECUTION_EXPIRATION` would lock up Governance. Consider adding an explicit check.

7.2 Suboptimal Code

This section lists suboptimal code patterns, which were found in the smart contract.

1. Line 28: the `_initializeConstants` function is probably redundant, as `setXXX` functions could be used instead. Consider moving its logic into Governance.initialize.
2. Line 30: if TORN supply changes, the `EXECUTION_EXPIRATION` may stop being 0.5%. If the percentage is important, consider specifying it instead of the number of tokens.
3. Line 39: each `setXXX` function allows setting a single parameter, while there could be situations when one wants to set several parameters atomically. Consider implementing a setter function that sets all parameters at once.

FIXED

1. Line 4: despite its name `Constants`, the contract actually defines mutable storage variables rather than constants. Consider renaming to `Configuration` or something like this.

7.3 Other Issues

This section lists other minor issues which were found in the token smart contract.

1. Line 6: upper case is commonly used for real compile-time constants. Using it for mutable storage variables is confusing.

FIXED

1. Line 33: `75 seconds` instead of just `75` would be more readable.

8. ECDSA.sol

In this section we describe issues found in the ECDSA.sol.

8.1 Fixed Other Issues

This section lists other minor issues which were found in the token smart contract.

Line 45: `byte(0, mload(add(signature, 0x60` could be done as:

```
mload(add(signature, 0x41))
```

9. Delegates.sol

In this section we describe issues found in the Delegates.sol.

9.1 Fixed Major Flaws

This section lists major flaws, which were found in the smart contract.

1. Line 12: in case when the `msg.sender` has already delegated its voting power, this will undelegate the previous delegation without emitting an event. Consider either adding an explicit check that msg.sender is not currently delegating, or emitting an Undelegate event in case current delegation is overridden by the new one. Otherwise it would be hard to track current delegations by logged event. After fixing came another problem: the `delegate` and the `undelegate` share some functionality. It might be easier to have a single function with 0 argument meaning stop delegation.
   Also, in case when the `to` is zero address, the call will actually undelegate the current delegation. Consider adding an explicit check that the `to` is not zero.

9.2 Suboptimal Code

This section lists suboptimal code patterns, which were found in the smart contract.

1. Line 13: the `Delegate` event is emitted even if the new delegation is the same as the current one.
2. Line 24: the function `proposeByDelegate` doesn't allow the caller to combine its own voting power with voting powers of those who've delegated to

him. Consider implementing an ability to pass an array of the `from` addresses, and to count voting powers of the caller and all specified delegates together when checking the proposal threshold.

3. Line 39: the `castDelegatedVote` function always casts votes from the `msg.sender`. In case when the `msg.sender` has very many delegates and cannot cast votes from all of them in a single all due to block gas limit, he may want to split the set of delegates into chunks and call the `castDelegatedVote` for each chunk. This will do extra work of casting votes from the `msg.sender` for each chunk. Consider implementing some ability to the `_castVote` from a list of delegates but not from the `msg.sender`.

9.3 Fixed Other Issues

This section lists other minor issues which were found in the token smart contract.

1. Line 26: the meaning of the `target` parameter is unclear from the function name and signature. Consider adding the documentation comments and renaming the parameter.

10. Voucher

In this section we describe issues found in the Voucher.sol.

10.1 Fixed Critical Flaws

This section lists critical flaws, which were found in the smart contract.

Line 37: the `pause` function can be called by anyone and there is no way to unpause the contract.
FIX: function removed.

10.2 Suboptimal Code

This section lists suboptimal code patterns, which were found in the smart contract.

1. Line 22: the `blockTimestamp` is unpredictable. It would be better to just pass expiration time explicitly as a constructor argument.
2. Line 43: the next check `to == address(0)` looks redundant. It is anyway possible to transfer tokens to zero address. Also this check seems irrelevant to the error message.

10.3 Other Issues

This section lists other minor issues which were found in the token smart contract.

Line 9: the next comment `TornadoCash voucher for early adopters` supposed to be token name, rather than token description.

11. Core

In this section we describe issues found in the Core.sol.

11.1 Fixed Critical Flaws

This section lists critical flaws, which were found in the smart contract.

Line 13: the `amounts[i] =` will always throw, as the length of the `amounts` is zero. Add the following line before the loop:
`amounts = new uint256[] (accs.length);`

11.2 Fixed Other Issues

This section lists other minor issues which were found in the token smart contract.

1. Line 7: the `delegatedTo` mapping is not used in the contract. Consider moving it where it is used, or moving here the functions, that use this mapping.
2. Line 9: the `balances` name is confusing. It doesn't say that these are locked balances. Consider renaming to the `lockedBalances`.
3. Line 11: the `getBalances` function is probably redundant as `balances` function is already public. However it may save some gas when called for many addresses at once.

12. Summary

Based on our findings, we also recommend the following:

1. Fix the major issues.
2. Pay attention to moderate issues.
3. Refactor the code to remove suboptimal parts.
4. Fix other (minor) issues.