# Tornado.Cash Security Analysis

by Pessimistic

This report is public.

Published: November 10, 2020

## Abstract

In this report, we consider the security of the smart contracts of Tornado.Cash project. Our task is to find and describe security issues in smart contracts of the platform.

## Disclaimer

## Summary

In this report, we considered the security of the smart contracts of Tornado.Cash project. We performed our audit according to the procedure described below.

The initial analysis showed an issue that can prevent a user from voting for a limited number of times by front-running attack. Also, a number of low severity issues were found, including a bug, gas consumption, code quality, code logic, and project configuration.

After the initial audit, the developers updated one of two repositories. Most of these issues were fixed. The issues that were not fixed do not endanger project security.

## General recommendations

We recommend fixing the issue with the front-running attack, adding documentation to the project and using static analyzers and other automated tools for further development to improve overall code quality.

## Procedure

In our audit, we consider the following crucial features of the code:

1. Whether code logic corresponds to the specification.
2. Whether the code is secure.
3. Whether the code meets best practices.

We perform our audit according to the following procedure:

- Automated analysis
    - We scan project's code base with automated tools: Crytic, MythX, and SmartCheck.
    - We manually verify (reject or confirm) all the issues found by tools.
- Manual audit
    - We inspect the specification and check whether the logic of smart contracts is consistent with it.
    - We manually analyze code base for security vulnerabilities.
    - We assess overall project structure and quality.
- Report
    - We reflect all the gathered information in the report.

# Project overview

## Main security requirements

- Tokens can be withdrawn only by the owner (the user who deposited them)
- Only the tokens locked in the Governance contract can vote for or against a proposal
- Each token is counted either zero times or once for each proposal
- Users can only vote with their own tokens and the tokens that have been delegated to them
- Tokens that are used for voting stay locked until expiration of `EXECUTION_DELAY` period
- Only the user who has `PROPOSAL_THRESHOLD` tokens or is delegated `PROPOSAL_THRESHOLD` by one address can create a proposal
- A proposal can be executed only if more than a half of all cast votes are for the proposal and there are at least `QUORUM_VOTES` votes

All the requirements are satisfied.

## Project description

For the analysis, we provided with two GitHub repositories of Tornado.Cash project:

- Governance contracts, commit 36f2b1b1b5df57a255b908fe6b22dc10b5865f24
- Token contracts, commit 8343c4f89267a25aeae71afaf49b1d8120196235

The project has no documentation.

The total LOC of audited sources is 701.

# Manual analysis

The contracts were completely manually analyzed, their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

## Critical issues

Critical issues seriously endanger smart contracts security. We highly recommend fixing them.

**The audit showed no critical issues.**

## Medium severity issues

Medium issues can influence project operation in current implementation. We highly recommend addressing them.

### Front-running attack

`castDelegatedVote()` function of **Delegation** contract at line 52 is susceptible to a front-running attack.

Let a user (A) delegated tokens to another user (B). User B calls `castDelegatedVote()` function, but user A decided to undelegate these tokens. If `undelegate()` transaction of user A is mined before `castDelegatedVote()` transaction of user B, `castDelegatedVote()` transaction will fail due to `require()` check at line 52.

This can be used by an attacker who can frontrun the caller of `castDelegatedVote()` function, and thus prevent the user from voting.

We recommend replacing `require()` with the check that will skip undelegated tokens.

## Low severity issues

Low severity issues can influence project operation in future versions of code. We recommend taking them into account.

### Bug

`setVoteExtendTime()` function of **Configuration** contract at line 69 should check if passed value is correct, not the stored one.

*This issue has been fixed in commit 6a6c4e0a3f198f3c5c0c5ac9a801e695defe8ed1.*

### Code quality

**Governance** contract has some issues with code quality:

- `_transferTokens()` function violates checks-effects-interactions pattern
- `propose()` function should be `external`

  *This issue has been fixed in commit b0ac86d19cc2e7b90df3a430790a90dcefc61a73.*
- `execute()` function is declared as `virtual` for testing purposes. This approach obstructs code readability.
- if `execute()` function is called for the address with no code (EoA or a self-destructed contract), `delegatecall()` at line 180 will return `(true, 0x)`, and the function will emit `ProposalExecuted` event.
- To process this edge case correctly, `delegatecall()` is often protected by `extcodesize` check.

  *This issue has been fixed in commit 572482c2e79ede5b5ecf0cfd5ec4ce7aef863597.*

**Token contracts** have the following issues:

- `blockTimestamp()` function of **ERC20permit** contract is used to manipulate time for testing purposes. This approach obstructs code readability.
- `rescueTokens()` function of **TORN** contract uses `transfer()` to send ether and thus can fail due `Out of gas` at line 101.
- natspec for the constructor of **Vesting** contract misses `_token` and `_startTimestamp` parameters.

### Code logic

If `_propose()` function of **Governance** contract if called from `proposeByDelegate()` function, `ProposalCreated` event should include the delegator as `proposer`.

*This issue has been fixed in commit dce958ea348c12b59d1c373f12e30fefbf5fa99c.*

## Gas consumption

- `Proposal` and `Receipt` structs of **Governance** contract could utilize smaller `uint` types, so the compiler optimizes storage consumption. E.g. `block.timestamp` fits into `uint32`, and the whole TORN balance fits into `uint88`.

  `Proposal` can be reduced to three storage slots and `Receipt` to one.

- Optimizer is disabled in **truffle-config.js** at line 51.

  *This issue has been fixed in commit 38fd36b7f9e7daae52bf5f8689d133579064246b.*

## Project configuration

`truffle-plugin-verify` is mentioned in **truffle-config.js** but is not managed by **package.json**. This results in `yarn coverage` command failure.

*This issue has been fixed in commit 38fd36b7f9e7daae52bf5f8689d133579064246b.*

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer

Boris Nikashin, Analyst

Alexander Seleznev, Founder

November 10, 2020