Quantstamp Security Assessment Certificate

February 18th 2021 — Quantstamp Verified

OriginTrail Starfleet Staking

This security assessment was prepared by Quantstamp, the leader in blockchain security

Executive Summary

Type	Staking Contract
Auditors	Sebastian Banescu, Senior Research Engineer Leonardo Passos, Senior Research Engineer Jose Ignacio Orlicki, Senior Engineer
Timeline	2021-02-04 through 2021-02-17
EVM	Muir Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	OT-RFC 10 Starfleet boarding (staking) smart contract specs
Documentation Quality	Medium
Test Quality	Medium

Source Code

Repository	Commit
starfleet-boarding-contract	595a752 (audit)
starfleet-boarding-contract	79012db (1st reaudit)
starfleet-boarding-contract	8f145a9 (2nd reaudit)

Total Issues	**16** (12 Resolved)
High Risk Issues	**1** (1 Resolved)
Medium Risk Issues	**4** (2 Resolved)
Low Risk Issues	**6** (5 Resolved)
Informational Risk Issues	**4** (3 Resolved)
Undetermined Risk Issues	**1** (1 Resolved)

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
Undetermined	The impact of the issue is uncertain.
Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
Acknowledged	The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).
Resolved	Adjusted program implementation, requirements or constraints to eliminate the risk.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

Quantstamp has performed a security audit of the OriginTrail `StarfleetStaking` contract and has identified 13 security issues ranging from High to Undetermined risk levels. Additionally, we have found 11 best practice issues, missing comments and a low branch coverage when executing the existing test suite. We recommend addressing all these issues before deploying the smart contract in production.

DISCLAIMER: The TRAC token and the StarTRAC token contracts were not part of this audit. Quantstamp has assumed that both of these tokens strictly adhere to the ERC20 token standard. If this assumption is broken, then other issues might arise. As a user of the `StarfleetStaking` contract we recommend you check if the 2 aforementioned tokens adhere to the ERC20 standard.

Update: Quantstamp has performed a reaudit of the code at commit `79012db` and has marked 9 issues Fixed, 3 issues Mitigated, 1 issue Acknowledged and has found 3 additional issues with IDs from QPS-14 to QSP-16. Additionally, 7 of 11 best practice issues have been fixed and 2 others have been partially fixed. Finally, 12 additional tests have been added to the test suite, which have increased the branch coverage from 65% to 73%.

ID	Description	Severity	Status
QSP-1	Functional Requirements Not Clear and/or Not Correctly Enforced	High	Mitigated
QSP-2	Missing Funds	Medium	Acknowledged
QSP-3	Integer Overflow / Underflow	Medium	Fixed
QSP-4	Unchecked Return Value	Medium	Fixed
QSP-5	Token Address Converted To ERC20 Contract Instead Of IERC20 Interface	Low	Fixed
QSP-6	Owner Could Renounce Ownership	Low	Fixed
QSP-7	`MinThresholdReached` Emitted After Every Deposit Once `MIN_THRESHOLD` Is Reached	Low	Fixed
QSP-8	After Withdrawal The Participant Is Not Removed	Low	Fixed
QSP-9	Contract May Receive ETH	Low	Fixed
QSP-10	Unlocked Pragma	Informational	Mitigated
QSP-11	`now` Is Deprecated	Informational	Mitigated
QSP-12	Block Timestamp Manipulation	Informational	Fixed
QSP-13	`tZero` Could Refer To The Past	Undetermined	Fixed
QSP-14	Gas costs for accounting could be prohibitive	Medium	Unresolved
QSP-15	Use of `msg.sender.transfer()`	Low	Unresolved
QSP-16	`accountStarTRAC` assumes distinct contributors	Informational	Unresolved

Quantstamp Audit Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.7.0

Steps taken to run the tools:

1. Installed the Slither tool: `pip install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Findings

QSP-1 Functional Requirements Not Clear and/or Not Correctly Enforced

Severity: *High Risk*

Status: Mitigated

File(s) affected: `StarfleetStake`

Description: The OT-RFT10 document describes something similar to a finite state machine (FSM) with at least 4 states, i.e. boarding period, starfleet launch, bridge launch, fallback period. However it is unclear what the transition conditions between these states are. For instance, the table on page 4 of the OT-RFT10 document indicates that in addition to the timestamp constraint of each of the periods, there is an additional condition for transitioning from the "boarding period" state to the "starfleet launch" state, namely the `MIN_THRESHOLD` must be reached. However, there is no functional requirement that indicates that the starfleet launch cannot start if the `MIN_THRESHOLD` is not reached. Therefore, the specification is not clear for the transition from the "boarding period" state to the "starfleet launch" state.

In the implementation this unclarity translates to a violation of FR3 which states that:

> "FR3 Token holders SHALL NOT be able to withdraw TRAC tokens during the Starfleet launch window and Bridge launch window phases."

This leads to the possibility of users withdrawing their staked deposits within the starfleet launch and bridge launch windows in case the `MIN_THRESHOLD` is not reached.

Exploit Scenario: The following steps illustrate this issue:

1. Alice deposits a certain amount of tokens during the boarding period

2. The boarding period passes but the `MIN_THRESHOLD` is not reached.
3. Given that the the boarding period has ended according to the current timestamp, the contract is in the starfleet launch window.
4. Alice can still call `withdrawTokens()` to withdraw her entire deposit.

Recommendation: Either clarify the transition conditions from one state to another or enforce the functional requirements in the code. Alternatively, adjust the functional requirements from the specification to reflect the implementation more precisely.

Update: While the specification now contains a table with the possible state transitions and associated conditions, for clarification purposes, we suggest rephrasing FR2 as follows (see bold text to be added):

> "At any point in time, token holders SHALL be able to withdraw their deposited TRAC if the minimum staking threshold (MIN_THRESHOLD_REACHED) has not been reached"

Additionally, we recommend adding the following condition `require(min_threshold_reached, "Minimum threshold not reached");` to each of the functions below:

- `fallbackWithdrawTokens`
- `accountStarTRAC`
- `transferTokens`

With the suggested change, there will be a better alignment between the code and the specification.

QSP-2 Missing Funds

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `StarfleetStake`

Description: As hinted in FR2, contributors will not be able to withdraw their tokens after the minimum threshold is reached, that is once `min_threshold_reached == true`, because of the `require` statement on L103 in the `withdrawTokens()` function.

According to FR4, the owner of the contract has the privilege to transfer its full balance to a custodian during the BRIDGE_PERIOD. Additionally, as per FR5, the owner can perform accounting (i.e. specify who the contributors are and how much they contributed) in the fallback period. This accounting will allow contributors to withdraw the indicated amount of tokens. These 2 operations are of high risk and high impact therefore it is necessary to satisfy:

> "NFR2 The Starfleet staking smart contract manager SHALL utilize a secure multisignature wallet for contract management."

However, it is unclear why such an accounting function is needed, since the information regarding the contributors and their deposited amounts are already stored in the list of the `stake` state variable. Calling the `accountStarTRAC` is error prone (e.g., participants could be missed or amounts could be incorrect) and costly (i.e. gas costs) since the number of participants could be very high. If the number of participants is high enough, the `accountStarTRAC` function may fail due to an out of gas error, which would mean that the input array of `contributors` and `amounts` needs to be partitioned and the functions needs to be called multiple times for each partition, which would make the process even more error prone.

Moreover, there is no guarantee that the transferred tokens will be returned from the custodian in full and on time, i.e. when the bridge period is over. The contributors have to trust the owner that this will be the case. During the Bridge Launch Period the `custodian` address has no specific interface to comply with and can also be a non-contract. As per FR4, the contract owner can withdraw all the tokens to an arbitrary EOA.

Recommendation: The following 3 steps are all recommended:

1. Remove the `accountStarTRAC()` function to reduce human error and gas costs. This also means removing `fallbackWithdrawTokens()`, whose logic and purpose can be incorporated in `withdrawTokens()`.
2. Remove the first `require(now >= tZero)` statement from `withdrawTokens()`, because deposits are not allowed before `tZero`, which means the withdraw function will fail on the `require(stake[msg.sender] > 0)` anyway.
3. Change the second require statement in `withdrawTokens()` from: `require(!min_threshold_reached)` to `require(!min_threshold_reached || now > tZero + BOARDING_PERIOD_LENGTH + LOCK_PERIOD_LENGTH + BRIDGE_PERIOD_LENGTH)`.
4. Include checks to make sure that the `custodian` is a contract and also satisfy an specific contract interface, for example `IBridgeCustodian` interface including a flag `isBridgeCustodian() returns (bool)`. Check the flag in `transferTokens()`.

   Update from the dev team: The intention behind the `accountStarTRAC` function is to enable the contract manager to copy a snapshot of the Starfleet blockchain ledger state on Ethereum, should the Fallback period be initiated (and Starfleet bridge launch fail). Since the token amounts for each participant address of staked TRAC in the smart contract at the end of the boarding phase, and acquired StarTRAC on Starfleet blockchain at the end of the Starfleet launch and Bridge phases can (and most likely will) differ due to StarTRAC being actively transacted with on the Starfleet blockchain for a longer period of time, it is necessary to use `accountStarTRAC` to create a snapshot of the Starfleet ledger state at possible end-of-life of Starfleet blockchain, and fallbackWithdrawTokens then to be utilized by token holders to return their TRAC in the amount of StarTRAC that the snapshot associates with their account. These features are however only to be utilized in the case of Starfleet project failure, and the functions are deliberately implemented in such a way that they do not alter the stake list state to minimize complexity and potential problems. Recommendations 2 and 4 from QSP-2 have been included in the smart contract implementation as they contribute to the security and efficiency of the smart contract.

QSP-3 Integer Overflow / Underflow

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `StarfleetStake`

Related Issue(s): SWC-101

Description: Integer overflow/underflow occur when an integer hits its bit-size limit. Every integer has a set range; when that range is passed, the value loops back around. A clock is a good analogy: at 11:59, the minute hand goes to 0, not 60, because 59 is the largest possible minute. The following instance were detected in the contracts:

1. The `require` statement on L66 uses the primitive addition operator to add a user provided input `amount` to the balance of the contract, which could result in an integer overflow.
2. There are several locations where `tZero` is added to one of the 3 period length constants defined in the contract using the `+` operator:
   . `depositTokens` on L65
   . `fallbackWithdrawTokens` on L114
   . `accountStarTRAC` on L128

. `transferTokens` on L142

Recommendation: Use the `add` function in the OpenZeppelin `SafeMath` library.

QSP-4 Unchecked Return Value

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `StarfleetStake`

Related Issue(s): SWC-104

Description: Most functions will return a `true` or `false` value upon success. Some functions, like `send()`, are more crucial to check than others. According to the ERC20 standard the `transfer` and `transferFrom` functions return `true` if the transfer was successful and `false` otherwise. These functions do not need to revert if the transfer is not successful. However, there are several places in the contract where these return values are ignored:

1. `depositTokens` ignores return value by `token.transferFrom(msg.sender,address(this),amount)` on L70
2. `withdrawTokens` ignores return value by `token.transfer(msg.sender,amount)` on L107
3. `fallbackWithdrawTokens()` ignores return value by `token.transfer(msg.sender,amount)` L120
4. `transferTokens` ignores return value by `token.transfer(custodian,balanceTransferred)` on L148

If the `token` contract signals an error by returning `false` instead of reverting, the functions above do not detect any error and proceed with their execution.

Recommendation: It's important to ensure that every necessary function is checked. Require the return values of the `transfer` and `transferFrom` functions be `true`.

QSP-5 Token Address Converted To ERC20 Contract Instead Of IERC20 Interface

Severity: *Low Risk*

Status: Fixed

File(s) affected: `StarfleetStake`

Description: On L52 and 55, the input token address is converted to `ERC20`, which may be an unsafe operation if the target contract does not inherit from OpenZeppelin's implementation.

Recommendation: Cast the input address to the `IERC20` interface. This makes the implementation agnostic to whether the target contract is indeed inheriting from OpenZeppelin.

QSP-6 Owner Could Renounce Ownership

Severity: *Low Risk*

Status: Fixed

File(s) affected: `StarfleetStake`

Description: Presently, the contract owner could renounce his ownership, leaving the contract without any owner. If that happens, `onlyOwner` operations will not longer be possible to be executed, which could cause the contract to be partially inoperable.

Recommendation: Clarify if the issue is indeed problematic; if so, make sure that the contract maintains an owner at all times by overriding the `renounceOwnership` function inherited from the `Ownable` contract. If not an issue, state why it is not with proper documentation in the code.

QSP-7 `MinThresholdReached` Emitted After Every Deposit Once `MIN_THRESHOLD` Is Reached

Severity: *Low Risk*

Status: Fixed

File(s) affected: `StarfleetStake`

Description: Once the total token balance of the contract is higher or equal to `MIN_THRESHOLD`, the `depositTokens` function will emit the `MinThresholdReached` event on each subsequent deposit. This might be confusing for any listener of this event, in case they are expecting this event to be emitted only once.

Recommendation: Only emit the event if `min_threshold_reached == false` in addition to the existing `if`-condition on L78.

QSP-8 After Withdrawal The Participant Is Not Removed

Severity: *Low Risk*

Status: Fixed

File(s) affected: `StarfleetStake`

Description: The list of `participants` tracks addresses with positive balance of `token` deposited in the contract. If the contributor withdraws their balance by calling the `withdrawTokens()` function, that contributor is not removed from the list of `participants`.

Recommendation: Remove the contributor from the `participants` list if they call `withdrawTokens()`

Update: Although this issue is fixed, it would be recommended that in addition to removing an account address from the `participants` array, also removing the account index position in the `participant_indexes` mapping.

QSP-9 Contract May Receive ETH

Severity: *Low Risk*

Status: Fixed

File(s) affected: `StarfleetStake`

Description: As required by functional requirement FR8 the staking contract should not be allowed to receive any ETH transfers. The default behaviour (if no fallback function is explicitly given) is to throw an exception. For unwilling EOA transfers this is sufficient. However, according to the official Solidity documentation (https://docs.soliditylang.org/en/v0.6.0/contracts.html#receive-ether-function):

> "A contract without a receive Ether function can receive Ether as a recipient of a **coinbase** transaction (aka miner block reward) or as a destination of a `selfdestruct`."

Recommendation: Include a withdrawal function for Ether and other ERC20s (except for the token corresponding to the `token` state variable), that can be called only by the owner of the stake contract. This will avoid frozen funds in this contract.

QSP-10 Unlocked Pragma

Severity: *Informational*

Status: Mitigated

File(s) affected: `StarfleetStaking.sol`

Related Issue(s): SWC-103

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.6.*` or `pragma solidity >=0.6.0 <=0.8.0`. The caret (^) or the greater/less than or equal (>=, <=) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked". Given that only certain versions of the Solidity compiler are recommended for production use it is dangerous to use this wide range of versions in the pragma. Moreover, the compiled contracts with one version of the compiler might behave slightly differently than when compiled with a different version of the compiler.

Recommendation: For consistency and to prevent unexpected behavior in the future, it is recommended to remove the caret, greater/less than or equal signs and to lock the file onto a specific Solidity version. Some versions that fall in the existing range and are recommended for production are: 0.6.8, 0.610, 0.6.11.

Update: The `StarfleetStaking` contract has been fixed. However, the `IBridgeCustodian` and `MultiSigWallet` have unlocked pragmas. We recommend also locking those pragmas.

QSP-11 `now` Is Deprecated

Severity: *Informational*

Status: Mitigated

File(s) affected: `StarfleetStake`

Description: Solidity 0.7.0 deprecated the `now` keyword. For contracts ^0.7.0, you must use `block.timestamp`. Since the pragma of the contract is unlocked and allows versions between 0.6.0 and 0.8.0 this could be an important issue.

Recommendation: If you decide to use a version of Solidity that is higher or equal to 0.7.0, then use `block.timstamp` instead of `now`.

Update: This issue was mitigated by by using solidity `0.6.10`. However, if the Solidity version will be changed in the future, this issue could arise again.

QSP-12 Block Timestamp Manipulation

Severity: *Informational*

Status: Fixed

File(s) affected: `StarfleetStake`

Related Issue(s): SWC-116

Description: Projects may rely on block timestamps for various purposes. However, it's important to realize that miners individually set the timestamp of a block, and attackers may be able to manipulate timestamps for their own purposes by up to 900 seconds. The smart contract relies the value of the block timestamp for almost every function and is therefore subject to this attack.

Recommendation: Either implement tests to show that a 900 second delay does not impact end-users, or clarify to end users (via publicly facing documentation) that such a 900 second time error is possible and they should not wait for the last minutes of a given period to perform an action.

QSP-13 `tZero` Could Refer To The Past

Severity: *Undetermined*

Status: Fixed

File(s) affected: `StarfleetStake`

Description: In the constructor, `tZero` could be set to the past, i.e., any timestamp lower than `now` if `startTime > 0`. It is unclear whether this is a desired behavior or if it is a bug.

Recommendation: Double check if the issue is indeed a design choice; if not, make sure to require `startTime` to be greater than (or equal to) `now`.

QSP-14 Gas costs for accounting could be prohibitive

Severity: *Medium Risk*

Status: Unresolved

File(s) affected: `StarfleetStake.sol`

Related Issue(s): SWC-128

Description: A Denial-of-Service (DoS) attack is a situation which an attacker renders a smart contract unusable and/or when the contract owner is forced to call the contract too many times, e.g.:

1. If the number of participants is high enough, the `accountStarTRAC` function may fail due to an out of gas error, which would mean that the input array of `contributors` and `amounts` needs to be partitioned and the functions needs to be called multiple times for each partition
2. If one has enough funds to make many small deposits, accounting as performed by `accountStarTRAC` could require more gas than the limit set within a block. In the latter case, accounting will require breaking the `contributors` array into smaller chunks, and one will have to send one transaction per chunk, which could be very costly and time consuming.

Note that this is related to the 3rd paragraph of QSP-2. Since recommendations 1 and 3 were not implemented, the issue described here is possible.

Recommendation: The following steps are recommended:

1. Add a limit to the number of contributors that can be registered per call to `accountStarTRAC()`. This will prevent out-of-gas errors.
2. Add a minimum deposit amount to make the issue where multiple small deposits are made financially infeasible.

QSP-15 Use of `msg.sender.transfer()`

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `StarfleetStake.sol`

Description: `.transfer()` forwards exactly 2,300 gas to the recipient. The goal of this hardcoded gas stipend was to prevent reentrancy vulnerabilities, but this only makes sense under the assumption that gas costs are constant. Recently EIP 1884 was included in the Istanbul hard fork. One of the changes included in EIP 1884 is an increase to the gas cost of the SLOAD operation, causing a contract's fallback function to cost more than 2300 gas.

Recommendation: Stop using `.transfer()` and instead use `.call()`.

QSP-16 `accountStarTRAC` assumes distinct contributors

Severity: *Informational*

Status: Unresolved

File(s) affected: `StarfleetStake.sol`

Description: The `accountStarTRAC` function assumes that all entries in the `contributors` array are distinct. While that is a fair assumption, it should be documented and enforced in the code.

Recommendation: The following steps are recommended:

1. Add a code comment stating that the function assumes all entries in the `contributors` array to be distinct.
2. Add a flag called `allowOverwriting` to this function which should be set to `true` if the function should perform corrective accounting actions that need to update amount values performed by a prior call to this function. This flag should be set to `false` when the function is called to initialize amount values for the first time.
3. If the `allowOverwriting` flag is set to `false`, then check that the value of `starTRAC_snapshot[contributors[i]]` is equal to `0` before assigning it `amounts[i]`. If it is not it means that `contributors[i]` has already appeared in the list of contributors or was initialized by a prior call to this function.

Automated Analyses

Slither

Slither has detected 52 results out of which most have been filtered out as false positives. The true positives have been incorporated in the findings above.

Adherence to Specification

We have found places where the implementation does not completely adhere to the specification or where the specification is not sufficiently clear. These findings are all included above.

Code Documentation

Each function should at least have a brief description of its purpose and a description of each input and output parameter. This is not the case for any of the functions in the code base.

Adherence to Best Practices

1. **[Fixed]** The `README.md` file does not describe how to setup the project and how to run the tests to check if they pass. The typical setup for an NPM project consists of 2 simple commands: `npm install` and `npm test`.
2. **[Fixed]** There is no script to check the code coverage of the test suite. This should normally be included in `package.json` and be executable using `npm run coverage`.
3. Since the `tZero` state variable is only initialized in the constructor and never changed, it should be declared as `immutable`.
4. **[Fixed]** Avoid computing the same arithmetic summations using the same constants in multiple functions in order to save gas costs. These operations could be precomputed at compile time, e.g. instead of defining the period length you could define the period end in the constructor like so: `boardingPeriodEnd = tZero.add(BOARDING_PERIOD_LENGTH);`, where `boardingPeriodEnd` would be declared as an `immutable` state variable. The same should be done for the other 2 constants referring to period lengths.
5. The contents of the `participants` state variable are not used in the contract. Only the length of the `participants` is used. This means that storage space is wasted, which costs gas. We recommend simply storing the number of participants in an integer state variable instead.
6. **[Fixed]** Several `require` statements are missing an error message as the 2nd argument. Such error messages are helpful to end-users to indicate why the transaction failed and also serve as documentation for code maintenance.
7. **[Fixed]** On L55, refactor the address into a contract constant.
8. **[Fixed]** Variable naming style appear to be inconsistent throughout the code. For instance: `min_threshold_reached` (snake case) versus `StarTRAC_snapshot` (a mix of

upper camel case + snake case) versus `startTime` (camel case). Adopt one naming style and stick to it consistently.

9. **[Fixed]** Events are not indexed. We suggest indexing event parameters to aid querying.

10. **[Partially Fixed]** The `contracts/` folder should only contain production code. However, the `TracToken.sol` file seems to be only used for testing purposes. Testing contracts should be moved into a dedicated subfolder called something like `contracts/mocks/` to distinguish them from production contracts.

11. **[Partially Fixed]** Make sure to indent code properly. For instance, a code snippet like:

```
if(startTime!=0){
  tZero = startTime;
}else{
  tZero = now;
}
```

could be better indented as in:

```
if(startTime != 0){ <=== space separating operator from operands
  tZero = startTime;
} else { <=== `{` has surrounding spaces
  tZero = now;
}
```

Many other parts can also be improved. We advise using an automated indenting tool.

Test Results

Test Suite Results

We confirm that all the 20 existing test cases are passing.

Update: The test suite has been extended with 12 additional test cases and we confirm that all tests are passing.

```
Contract: StarfleetStake
  Token holder deposit functionality checks
    StarfleetStake & Token basic checks
       ✓ Sanity check
       ✓ Account 0 (Contract manager) should be owner (3036ms)
       ✓ Account 1 should not be owner
       ✓ Contract manager cannot renounce ownership (68ms)
       ✓ Contract manager can change ownership (79ms)
       ✓ Non-managers cannot change ownership (41ms)
       ✓ Staking contract should have 0 tokens at deployment
       ✓ Account 0 has totaly supply (for testing purposes)
       ✓ Staking contract should not accept ETH (57ms)
    TH1 - Token holders must be able to deposit TRAC during the boarding period
       ✓ Token holders cannot try to deposit a zero amount (107ms)
       ✓ Token holder can deposit 1000 tokens before boarding period has expired (439ms)
       ✓ A token holder cannot deposit more than MAX_THRESHOLD tokens (308ms)
       ✓ Cannot deposit tokens before boarding period has started (203ms)
       ✓ Cannot deposit tokens after boarding period has expired (235ms)

Contract: StarfleetStake
  TH2 Must be able withdraw TRAC before BOARDING_PERIOD_END and MIN_THRESHOLD not reached
     ✓ Can withdraw deposited TRAC before BOARDING_PERIOD_END, when MIN_THRESHOLD NOT reached (663ms)
     ✓ Cannot withdraw TRAC if there is no TRAC deposited (92ms)
     ✓ Withdrawing tokens updates the participant array correctly (1781ms)
     ✓ Cannot withdraw deposited TRAC when MIN_THRESHOLD reached (775ms)
     ✓ Contract manager cannot transfer funds before bridge launch window (329ms)
     ✓ Contract manager cannot transfer transfer funds during bridge launch window if the custodian is not a contract (121ms)
     ✓ Contract manager cannot transfer transfer funds during bridge launch window if the custodian does not have getOwners function (140ms)
     ✓ Contract manager cannot transfer transfer funds during bridge launch window if the custodian does not have owners (320ms)
     ✓ Contract manager can transfer funds during bridge launch window (438ms)

Contract: StarfleetStake
  StarfleetStake & Token basic checks
     ✓ TH5 Must be able withdraw TRAC after BOARDING_PERIOD_LENGTH in case of MIN_THRESHOLD not reached (862ms)

Contract: StarfleetStake
   ✓ Contract manager cannot transfer funds after bridge launch window (555ms)
   ✓ Contract manager can account StarTRAC after bridge period (251ms)
   ✓ Reverts when arrays not the same length (128ms)
   ✓ Token holder can claim StarTRAC when StarTRAC snapshot is available (217ms)
   ✓ Token holder cannot claim StarTRAC twice (109ms)
   ✓ Cannot withdraw TRAC with withdrawMisplacedTokens (90ms)
   ✓ Can withdraw non-TRAC tokens with withdrawMisplacedTokens (571ms)
   ✓ In case of accidental Ether through selfDestruct, the withdrawMisplacedEther should be able to send to owner (212ms)


32 passing (20s)
```

Code Coverage

The branch coverage of the `StarfleetStake` contract is too low. We recommend bringing its value close to 100% to ensure that all the functionality supported by the contract has been properly tested.

Update: The branch coverage has been increased from 65% to 73%.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	94.94	73.44	92.86	95	
StarfleetStake.sol	94.94	73.44	92.86	95	65,198,199,200
contracts/mocks/	100	100	100	100	
IBridgeCustodian.sol	100	100	100	100	
MultiSigWallet.sol	100	100	100	100	
Suicidal.sol	100	100	100	100	
TracToken.sol	100	100	100	100	
All files	**95.24**	**73.44**	**94.74**	**95.29**	

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

```
6635edd391c42b988d4c3c122a75a0ddd139661801ca72571c802bc26f301b14 ./contracts/StarfleetStake.sol
fda4449552d0b0494df377ac27307dae2a253c6e145f5a1669685ac3afbf4bf6 ./contracts/Migrations.sol
fec46d2bccca7f7e5851be824e347d4b17e08b516012c66e957003fffbf47c66 ./contracts/mocks/IBridgeCustodian.sol
8f824769972dd78be9e04111833956bf08e530064a31b0db1bf8fd7137056192 ./contracts/mocks/Suicidal.sol
5dcbc39369ccba60f77ca6dae5ad72b9a000a0c3d8a4e0d659c109169b4206a0 ./contracts/mocks/MultiSigWallet.sol
3a037d2bc81b12d6f27da9dcf178a5ef93e540da5532d8a6eddf31376f2a529d ./contracts/mocks/TracToken.sol
```

Tests

```
f1352de1a15bb19a8d235fc88fb672f2c9c64d625db4f1ce2c6c65f2190579d5 ./test/contract_tests.js
```

Changelog

- 2021-02-10 - Initial report based on commit `595a752`
- 2021-02-16 - Updated report based on commit `79012db`
- 2021-02-17 - Updated report based on commit `8f145a9`

About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure blockchain platforms at scale using computer-aided reasoning tools, with a mission to help boost the adoption of this exponentially growing technology.

With over 1000 Google scholar citations and numerous published papers, Quantstamp's team has decades of combined experience in formal verification, static analysis, and software verification. Quantstamp has also developed a protocol to help smart contract developers and projects worldwide to perform cost-effective smart contract security scans.

To date, Quantstamp has protected $5B in digital asset risk from hackers and assisted dozens of blockchain projects globally through its white glove security assessment services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Quantstamp's collaborations with leading academic institutions such as the National University of Singapore and MIT (Massachusetts Institute of Technology) reflect our commitment to research, development, and enabling world-class blockchain security.