CERTIK

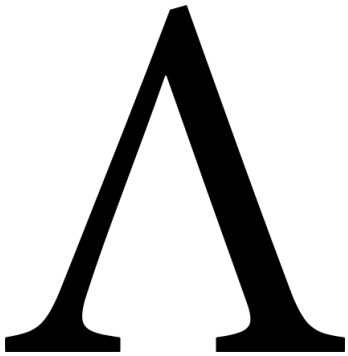
Audit Report

Produced by CertiK

for Λ

28th Feb, 2020

CertiK Audit Report For Ampleforth



Request Date: 2019-11-22
Revision Date: 2020-02-28
Platform Name: Ethereum

CERTIK

Contents

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Verification Services Agreement between CertiK and Ampleforth(the "Company"), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, the "Agreement"). This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

About CertiK

CertiK is a technology-led blockchain security company founded by Computer Science professors from Yale University and Columbia University built to prove the security and correctness of smart contracts and blockchain protocols.

CertiK, in partnership with grants from IBM and the Ethereum Foundation, has developed a proprietary Formal Verification technology to apply rigorous and complete mathematical reasoning against code. This process ensures algorithms, protocols, and business functionalities are secured and working as intended across all platforms.

CertiK differs from traditional testing approaches by employing Formal Verification to mathematically prove blockchain ecosystem and smart contracts are hacker-resistant and bug-free. CertiK uses this industry-leading technology together with standardized test suites, static analysis, and expert manual review to create a full-stack solution for our partners across the blockchain world to secure 6.2B in assets.

For more information: https://certik.org/

Executive Summary

This report has been prepared for Ampleforth to discover issues and vulnerabilities in the source code of their smart contracts. A comprehensive examination has been performed, utilizing CertiK's Formal Verification Platform, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Vulnerability Classification

CertiK categorizes issues into three buckets based on overall risk levels:

Critical

Code implementation does not match specification, which could result in the loss of funds for contract owner or users.

Medium

Code implementation does not match the specification under certain conditions, which could affect the security standard by loss of access control.

Low

Code implementation does not follow best practices, or uses suboptimal design patterns, which could lead to security vulnerabilities further down the line.

Testing Summary

PASS

CERTIK *believes this smart contract passes security qualifications to be listed on digital asset exchanges.*

Feb 28, 2020

Type of Issues

CertiK's smart label engine applied 100% formal verification coverage on the source code. Our team of engineers has scanned the source code using proprietary static analysis tools and code-review methodologies. The following technical issues were found:

Title	Description	Issues	SWC ID
Integer Overflow/ Underflow	An overflow/underflow occurs when an arithmetic operation reaches the maximum or minimum size of a type.	0	SWC-101
Function Incorrectness	Function implementation does not meet specification, leading to intentional or unintentional vulnerabilities.	0	
Buffer Overflow	An attacker can write to arbitrary storage locations of a contract if array of out bound happens	0	SWC-124
Reentrancy	A malicious contract can call back into the calling contract before the first invocation of the function is finished.	0	SWC-107
Transaction Order Dependence	A race condition vulnerability occurs when code depends on the order of the transactions submitted to it.	0	SWC-114
Timestamp Dependence	Timestamp can be influenced by miners to some degree.	1	SWC-116
Insecure Compiler Version	Using a fixed outdated compiler version or floating pragma can be problematic if there are publicly disclosed bugs and issues that affect the current compiler version used.	1	SWC-102 SWC-103
Insecure Randomness	Using block attributes to generate random numbers is unreliable, as they can be influenced by miners to some degree.	0	SWC-120
"tx.origin" for Authorization	tx.origin should not be used for authorization. Use msg.sender instead.	0	SWC-115

Title	Description	Issues	SWC ID
Delegatecall to Untrusted Callee	Calling untrusted contracts is very dangerous, so the target and arguments provided must be sanitized.	0	SWC-112
State Variable Default Visibility	Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.	0	SWC-108
Function Default Visibility	Functions are public by default, meaning a malicious user can make unauthorized or unintended state changes if a developer forgot to set the visibility.	0	SWC-100
Uninitialized Variables	Uninitialized local storage variables can point to other unexpected storage variables in the contract.	0	SWC-109
Assertion Failure	The assert() function is meant to assert invariants. Properly functioning code should never reach a failing assert statement.	0	SWC-110
Deprecated Solidity Features	Several functions and operators in Solidity are deprecated and should not be used.	0	SWC-111
Unused Variables	Unused variables reduce code quality	0	SWC-131

Vulnerability Details

Critical

No issue found.

Medium

No issue found.

Low

No issue found.

Manual Review Notes

Source Code SHA-256 Checksum

- **IStaking.sol**
  `c1e075730ae3ee33367c3a517c942ca4db33f987abf7eebf41754c6d9c3e978d`
- **TokenGeyser.sol**
  `a1530086b605c2723980154dad9dde16ee1900c899749b4302e309e76f314ec6`
- **TokenPool.sol**
  `50304745cbe4aaf7e80651e8a1336b334fa1ca0fb675cb608c5c540a697a04ab`

Summary

CertiK worked closely with Ampleforth to audit the design and implementation of its soon-to-be released smart contract. To ensure comprehensive protection, the source code was analyzed by the proprietary CertiK formal verification engine and manually reviewed by our smart contract experts and engineers. That end-to-end process ensures proof of stability as well as a hands-on, engineering-focused process to close potential loopholes and recommend design changes in accordance with best practices.

Overall, we found Ampleforth's smart contracts to follow good practices. With the final update of source code and delivery of the audit report, we conclude that the contract is structurally sound and not vulnerable to any classically known anti-patterns or security issues. The audit report itself is not necessarily a guarantee of correctness or trustworthiness, and we always recommend to seek multiple opinions, continually improve the codebase, and perform additioinal tests before the mainnet release.

Recommendations

Items in this section are not critical to the overall functionality of Ampleforth's smart contracts; however, we leave it to the client's discretion to decide whether to address them before the final deployment of source codes. Recommendations are labeled CRITICAL, MAJOR, MINOR, INFO, and DISCUSSION in decreasing significance level.

General

- INFO Compiler version: Recommend updating compiler to the latest version.
  - (Ampleforth - updated): Updated in commit $4088abcbf5923a613586b77790d9eaf6f12e55af$.

TokenGeyser.sol commit $a52136642fa95b5ae407cbe0a9bb559d2c7c360e$, previous

- CRITICAL `_unstake()`: `lastStake` is a copy, rather than a reference, of `accountStakes[accountStakes.length - 1]`. When `lastStake.stakingShares > sharesLeftToBurn`, `accountStakes[accountStakes.length - 1]` will not be updated when `lastStake` is updated. Recommend changing `Stake memory lastStake = accountStakes[accountStakes.length - 1]` to `Stake storage lastStake = accountStakes[accountStakes.length - 1]`.

  - (Ampleforth - updated): Fixed in commit 37b23ba279aab4e22e818465fcc20c092b52482a.

- MAJOR `.div()` is the division between integers rather than floating numbers. Frequent call of `.div()`, especially when the numerator is small or the denominator is large, will lead to large errors in results. For example, if `unlockScheduleShares()` is frequently called, the return could always be zero. Recommend adding a branch condition that allows user to withdraw everything remained in `initialLockedShares` once passed `endAtSec`.
  - (Ampleforth - updated): Fixed in commit 6a21a3eee985e4a5166a9926783f25f2bdc3d174.

- INFO `unlockScheduleShares()`: Considering gas saving, recommend adding an if statement so that if `now >= schedule.endAtSec` the function will return zero directly without other calculations.
  - (Ampleforth - updated): Updated in commit 6a21a3eee985e4a5166a9926783f25f2bdc3d174.

- INFO `require()`: Recommend adding error messages when calling `require()`.
  - (Ampleforth - updated): Updated in commit 84f97d11ce1b0a1e9707af1b50d30bedc020c0cc.
  - (CertiK - updated): Recommend adding error messages when calling `require()` in `constructor()`.

- INFO `constructor()`: Recommend adding a check to make sure `bonusPeriodSec_ != 0`.
  - (Ampleforth - updated): Updated in commit 996798113298 93b308898a8a6846974b5f1d6110.

- INFO `_lastAccountingTimestampSec`: Recommend initializing this variable to `now`.
  - (Ampleforth - updated): Updated in commit 9ccf49ba16fd9abe1aa849be47a746a374d3e28a.

- DISCUSSION `_unstake()`: Is there any reason to create a copy of `_userTotals[msg.sender]` for `totals` by storing it in `memory` instead of `storage`? A reference in `storage` would be better considering gas saving. Same question in `updateAccounting()`.
  - (Ampleforth - updated): These changes did save gas, as measured in unit tests. Updated in commit 0b30dc0f756b944438a2bacac61f0441724b5b5c.

- INFO `computeNewReward()`: Recommend using `oneHundredPct` instead of `10**BONUS_DECIMALS` in the calculation for `bonusedReward`.
  - (Ampleforth - updated): Updated in commit 7eb6e35f94dc3a867fa93048e373ef5a51a9b05a.

- INFO `lockTokens()` and `unlockTokens()`: Recommend saving the result of `totalLocked()` to a local variable to use considering gas saving.
  - (Ampleforth - updated): Updated in commit a7444c97aeca668c0d3e5bad861c30b8d7af58c4.

TokenGeyser.sol **commit** 84f97d11ce1b0a1e9707af1b50d30bedc020c0cc, **previous**

- Usage of `.div()`:
  - CRITICAL `_unstake()`: If we understand the Ampleforth Red Book correctly, `totalStakingShares` could be smaller than `totalStaked()`. Let's say `totalStakingShares < totalStaked()`, then `stakingSharesToBurn = 0` when `amount` is small enough (for example, `amount = 1`). There's no share burnt, but `msg.sender` can still get tokens from `_stakingPool`. We recommend checking `stakingSharesToBurn` after calculation and ending the function call if it is zero.
    - (Ampleforth - updated): Fixed in commit 1df70d4651a18d0b7aa774f26846466f58c29d32.
  - CRITICAL `_stakeFor()`: A similar problem exists in `_stakeFor()`. Users might not be able to get shares when they transfer a small amount to `_stakingPool`. We recommend checking `mintedStakingShares` after calculation and ending the function call if it is zero.
    - (Ampleforth - updated): Fixed in commit 1df70d4651a18d0b7aa774f26846466f58c29d32.
  - MAJOR `_lockTokens()`: Since it is possible that `totalLockedShares == 0` when `totalLocked()> 0`, we recommend calling `unlockTokens()` before calculating `mintedLockedShares` to set `totalLocked()` to zero. Otherwise all the tokens in `_lockedPool` will be sent to `_unlockedPool` once `unlockTokens()` is called, because `totalLockedShares` remains zero after tokens are sent to `_lockedPool`.
    - (Ampleforth - updated): Fixed in commit f7416a225c3c50d866e1ad4dc58b16033cf3d0c9.
  - DISCUSSION Divisions between integers brings losses of precision. It is not a big problem only when the numerator is much larger then the denominator.
    - (Ampleforth - updated): Updated in commit 18b54718934107675e93c7684163e6959352f844.
- DISCUSSION `_stakeFor()`: According to your descriptions, it seems `totalStaked()` can be changed by methods other than calling `_stakeFor()` or `_unstake()`. Then is it possible that `totalStaked()!= 0` while `totalStakingShares = 0`? If so, users will not be able to get shares by calling `_stakeFor()` when `totalStaked()!= 0 \&\& totalStakingShares = 0`. We recommend adding a check to make sure `totalStakingShares != 0` when `totalStaked()!= 0`.
  - (Ampleforth - updated): Fixed in commit 50418ebdf9a10392ecf98f2cb68af294134b9e97.

Static Analysis Results

INSECURE_COMPILER_VERSION

Line 1 in File TokenPool.sol

```
1 pragma solidity 0.5.0;
```

Version to compile has the following bug: 0.5.0: SignedArrayStorageCopy, ABIEncoderV2StorageArrayV DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor, IncorrectEventSignatureInLibraries, ABIEncoderV2PackedStorage

INSECURE_COMPILER_VERSION

Line 1 in File IStaking.sol

```
1 pragma solidity 0.5.0;
```

Version to compile has the following bug: 0.5.0: SignedArrayStorageCopy, ABIEncoderV2StorageArrayV DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor, IncorrectEventSignatureInLibraries, ABIEncoderV2PackedStorage

INSECURE_COMPILER_VERSION

Line 1 in File TokenGeyser.sol

```
1 pragma solidity 0.5.0;
```

Version to compile has the following bug: 0.5.0: SignedArrayStorageCopy, ABIEncoderV2StorageArrayV DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor, IncorrectEventSignatureInLibraries, ABIEncoderV2PackedStorage

TIMESTAMP_DEPENDENCY

Line 232 in File TokenGeyser.sol

```
232     totals.lastAccountingTimestampSec = now;
```

"now" can be influenced by miners to some degree

TIMESTAMP_DEPENDENCY

Line 234 in File TokenGeyser.sol

```
234     Stake memory newStake = Stake(mintedStakingShares, now);
```

"now" can be influenced by miners to some degree

TIMESTAMP_DEPENDENCY

Line 469 in File TokenGeyser.sol

```
469     _lastAccountingTimestampSec = now;
```

"now" can be influenced by miners to some degree

TIMESTAMP_DEPENDENCY

Line 480 in File TokenGeyser.sol

```
480        totals.lastAccountingTimestampSec = now;
```

! ”now” can be influenced by miners to some degree

TIMESTAMP_DEPENDENCY

Line 492 in File TokenGeyser.sol

```
492            now
```

! ”now” can be influenced by miners to some degree

TIMESTAMP_DEPENDENCY

Line 552 in File TokenGeyser.sol

```
552        schedule.lastUnlockTimestampSec = now;
```

! ”now” can be influenced by miners to some degree

TIMESTAMP_DEPENDENCY

Line 648 in File TokenGeyser.sol

```
648        if(now >= schedule.endAtSec){
```

! ”now” can be influenced by miners to some degree

TIMESTAMP_DEPENDENCY

Line 655 in File TokenGeyser.sol

```
655            schedule.lastUnlockTimestampSec = now;
```

! ”now” can be influenced by miners to some degree

Formal Verification Results

How to read

Detail for Request 1

transferFrom to same address

Verification date	20, Oct 2018
Verification timespan	395.38 ms

CERTIK *label location*	Line 30-34 in File howtoread.sol

CERTIK *label*

```
30    /*@CTK FAIL "transferFrom to same address"
31        @tag assume_completion
32        @pre from == to
33        @post __post.allowed[from][msg.sender] ==
34    */
```

Raw code location Line 35-41 in File howtoread.sol

Raw code

```
35    function transferFrom(address from, address to
          ) {
36        balances[from] = balances[from].sub(tokens
37        allowed[from][msg.sender] = allowed[from][
38        balances[to] = balances[to].add(tokens);
39        emit Transfer(from, to, tokens);
40        return true;
41    }
```

Counterexample This code violates the specification

Initial environment

```
1  Counter Example:
2  Before Execution:
3      Input = {
4          from = 0x0
5          to = 0x0
6          tokens = 0x6c
7      }
8      This = 0
53             balance: 0x0
54         }
55     }
56
```

Post environment

```
57 After Execution:
58     Input = {
59         from = 0x0
60         to = 0x0
61         tokens = 0x6c
```

Formal Verification Request 1

TokenPool

28, Feb 2020
4.24 ms

Line 14-16 in File TokenPool.sol

```
14  /*@CTK "TokenPool"
15    @post __post.token == _token
16   */
```

Line 17-19 in File TokenPool.sol

```
17  constructor(UFragments _token) public {
18      token = _token;
19  }
```

The code meets the specification.

Formal Verification Request 2

balance

28, Feb 2020
122.8 ms

Line 21-24 in File TokenPool.sol

```
21  /*@CTK "balance"
22    @tag assume_completion
23    @post __return == token._gonBalances[address(this)] / token._gonsPerFragment
24   */
```

Line 25-27 in File TokenPool.sol

```
25  function balance() public view returns (uint256) {
26      return token.balanceOf(address(this));
27  }
```

The code meets the specification.

Formal Verification Request 3

transfer

28, Feb 2020
2054.2 ms

Line 29-37 in File TokenPool.sol

```
29  /*@CTK "transfer"
30    @tag assume_completion
31    @pre msg.sender == _owner
32    @pre msg.sender != address(0)
33    @pre to != address(0)
```

```
    @post msg.sender != to -> __post.token._gonBalances[to] == token._gonBalances[to] +
        value * token._gonsPerFragment
    @post msg.sender != to -> __post.token._gonBalances[msg.sender] == token._gonBalances[
        msg.sender] - value * token._gonsPerFragment
    @post msg.sender == to -> __post.token._gonBalances[msg.sender] == token._gonBalances[
        msg.sender]
   */
```

Line 38-40 in File TokenPool.sol

```
  function transfer(address to, uint256 value) external onlyOwner returns (bool) {
     return token.transfer(to, value);
  }
```

The code meets the specification.

Formal Verification Request 4

supportsHistory

28, Feb 2020
3.82 ms

Line 21-23 in File IStaking.sol

```
  /*@CTK "supportsHistory"
    @post __return == false
   */
```

Line 24-26 in File IStaking.sol

```
  function supportsHistory() external pure returns (bool) {
     return false;
  }
```

The code meets the specification.

Formal Verification Request 5

TokenGeyser_require

28, Feb 2020
289.56 ms

Line 103-108 in File TokenGeyser.sol

```
  /*@CTK "TokenGeyser_require"
    @tag assume_completion
    @post startBonus_ <= 10**BONUS_DECIMALS
    @post bonusPeriodSec_ != 0
    @post initialSharesPerToken > 0
   */
```

Line 122-138 in File TokenGeyser.sol

```
122    constructor(UFragments stakingToken, UFragments distributionToken, uint256
           maxUnlockSchedules,
123                uint256 startBonus_, uint256 bonusPeriodSec_, uint256 initialSharesPerToken)
                       public {
124        // The start bonus must be some fraction of the max. (i.e. <= 100%)
125        require(startBonus_ <= 10**BONUS_DECIMALS, 'TokenGeyser: start bonus too high');
126        // If no period is desired, instead set startBonus = 100%
127        // and bonusPeriod to a small value like 1sec.
128        require(bonusPeriodSec_ != 0, 'TokenGeyser: bonus period is zero');
129        require(initialSharesPerToken > 0);
130
131        _stakingPool = new TokenPool(stakingToken);
132        _unlockedPool = new TokenPool(distributionToken);
133        _lockedPool = new TokenPool(distributionToken);
134        startBonus = startBonus_;
135        bonusPeriodSec = bonusPeriodSec_;
136        _maxUnlockSchedules = maxUnlockSchedules;
137        _initialSharesPerToken = initialSharesPerToken;
138    }
```

✅ The code meets the specification.

Formal Verification Request 6

TokenGeyser

📅 28, Feb 2020
⏱ 152.77 ms

Line 109-121 in File TokenGeyser.sol

```
109    /*@CTK "TokenGeyser"
110      @tag assume_completion
111      @pre startBonus_ <= 10**BONUS_DECIMALS
112      @pre bonusPeriodSec_ != 0
113      @pre initialSharesPerToken > 0
114      @post __post._stakingPool.token == stakingToken
115      @post __post._unlockedPool.token == distributionToken
116      @post __post._lockedPool.token == distributionToken
117      @post __post.startBonus == startBonus_
118      @post __post.bonusPeriodSec == bonusPeriodSec_
119      @post __post._maxUnlockSchedules == maxUnlockSchedules
120      @post __post._initialSharesPerToken == initialSharesPerToken
121     */
```

Line 122-138 in File TokenGeyser.sol

```
122    constructor(UFragments stakingToken, UFragments distributionToken, uint256
           maxUnlockSchedules,
123                uint256 startBonus_, uint256 bonusPeriodSec_, uint256 initialSharesPerToken)
                       public {
124        // The start bonus must be some fraction of the max. (i.e. <= 100%)
125        require(startBonus_ <= 10**BONUS_DECIMALS, 'TokenGeyser: start bonus too high');
126        // If no period is desired, instead set startBonus = 100%
127        // and bonusPeriod to a small value like 1sec.
128        require(bonusPeriodSec_ != 0, 'TokenGeyser: bonus period is zero');
129        require(initialSharesPerToken > 0);
130
```

```
        _stakingPool = new TokenPool(stakingToken);
        _unlockedPool = new TokenPool(distributionToken);
        _lockedPool = new TokenPool(distributionToken);
        startBonus = startBonus_;
        bonusPeriodSec = bonusPeriodSec_;
        _maxUnlockSchedules = maxUnlockSchedules;
        _initialSharesPerToken = initialSharesPerToken;
    }
```

The code meets the specification.

Formal Verification Request 7

getStakingToken

28, Feb 2020
3.78 ms

Line 143-145 in File TokenGeyser.sol

```
    /*@CTK "getStakingToken"
      @post __return == _stakingPool.token
     */
```

Line 146-148 in File TokenGeyser.sol

```
    function getStakingToken() public view returns (UFragments) {
        return _stakingPool.token;
    }
```

The code meets the specification.

Formal Verification Request 8

getDistributionToken

28, Feb 2020
11.31 ms

Line 153-157 in File TokenGeyser.sol

```
    /*@CTK "getDistributionToken"
      @tag assume_completion
      @pre _unlockedPool.token == _lockedPool.token
      @post __return == _unlockedPool.token
     */
```

Line 158-161 in File TokenGeyser.sol

```
    function getDistributionToken() public view returns (UFragments) {
        assert(_unlockedPool.token == _lockedPool.token);
        return _unlockedPool.token;
    }
```

The code meets the specification.

Formal Verification Request 9

Buffer overflow / array index out of bound would never happen.

28, Feb 2020
7025.11 ms

Line 168 in File TokenGeyser.sol

```
168    //@CTK NO_BUF_OVERFLOW
```

Line 170-172 in File TokenGeyser.sol

```
170    function stake(uint256 amount, bytes calldata data) external {
171        _stakeFor(msg.sender, msg.sender, amount);
172    }
```

The code meets the specification.

Formal Verification Request 10

Method will not encounter an assertion failure.

28, Feb 2020
2655.64 ms

Line 169 in File TokenGeyser.sol

```
169    //@CTK NO_ASF
```

Line 170-172 in File TokenGeyser.sol

```
170    function stake(uint256 amount, bytes calldata data) external {
171        _stakeFor(msg.sender, msg.sender, amount);
172    }
```

The code meets the specification.

Formal Verification Request 11

Buffer overflow / array index out of bound would never happen.

28, Feb 2020
2994.03 ms

Line 180 in File TokenGeyser.sol

```
180    //@CTK NO_BUF_OVERFLOW
```

Line 182-184 in File TokenGeyser.sol

```
182    function stakeFor(address user, uint256 amount, bytes calldata data) external {
183        _stakeFor(msg.sender, user, amount);
184    }
```

The code meets the specification.

Formal Verification Request 12

Method will not encounter an assertion failure.

28, Feb 2020
1731.66 ms

Line 181 in File TokenGeyser.sol

```
181    //@CTK NO_ASF
```

Line 182-184 in File TokenGeyser.sol

```
182    function stakeFor(address user, uint256 amount, bytes calldata data) external {
183        _stakeFor(msg.sender, user, amount);
184    }
```

The code meets the specification.

Formal Verification Request 13

Buffer overflow / array index out of bound would never happen.

28, Feb 2020
3653.3 ms

Line 192 in File TokenGeyser.sol

```
192    //@CTK NO_BUF_OVERFLOW
```

Line 216-247 in File TokenGeyser.sol

```
216    function _stakeFor(address staker, address beneficiary, uint256 amount) private {
217        require(amount > 0, 'TokenGeyser: stake amount is zero');
218        require(beneficiary != address(0), 'TokenGeyser: beneficiary is zero address');
219        require(totalStakingShares == 0 || totalStaked() > 0,
220                'TokenGeyser: Invalid state. Staking shares exist, but no staking tokens do');
221
222        uint256 mintedStakingShares = (totalStakingShares > 0)
223            ? totalStakingShares.mul(amount).div(totalStaked())
224            : amount.mul(_initialSharesPerToken);
225        require(mintedStakingShares > 0, 'TokenGeyser: Stake amount is too small');
226
227        updateAccounting();
228
229        // 1. User Accounting
230        UserTotals storage totals = _userTotals[beneficiary];
231        totals.stakingShares = totals.stakingShares.add(mintedStakingShares);
232        totals.lastAccountingTimestampSec = now;
233
234        Stake memory newStake = Stake(mintedStakingShares, now);
235        _userStakes[beneficiary].push(newStake);
236
237        // 2. Global Accounting
238        totalStakingShares = totalStakingShares.add(mintedStakingShares);
239        // Already set in updateAccounting()
240        // _lastAccountingTimestampSec = now;
241
242        // interactions
```

```
        require(_stakingPool.token.transferFrom(staker, address(_stakingPool), amount),
            'TokenGeyser: transfer into staking pool failed');

        emit Staked(beneficiary, amount, totalStakedFor(beneficiary), "");
    }
```

The code meets the specification.

Formal Verification Request 14

Method will not encounter an assertion failure.

28, Feb 2020
1824.4 ms

Line 193 in File TokenGeyser.sol

```
193     //@CTK NO_ASF
```

Line 216-247 in File TokenGeyser.sol

```
216     function _stakeFor(address staker, address beneficiary, uint256 amount) private {
217         require(amount > 0, 'TokenGeyser: stake amount is zero');
218         require(beneficiary != address(0), 'TokenGeyser: beneficiary is zero address');
219         require(totalStakingShares == 0 || totalStaked() > 0,
220                 'TokenGeyser: Invalid state. Staking shares exist, but no staking tokens do');
221
222         uint256 mintedStakingShares = (totalStakingShares > 0)
223             ? totalStakingShares.mul(amount).div(totalStaked())
224             : amount.mul(_initialSharesPerToken);
225         require(mintedStakingShares > 0, 'TokenGeyser: Stake amount is too small');
226
227         updateAccounting();
228
229         // 1. User Accounting
230         UserTotals storage totals = _userTotals[beneficiary];
231         totals.stakingShares = totals.stakingShares.add(mintedStakingShares);
232         totals.lastAccountingTimestampSec = now;
233
234         Stake memory newStake = Stake(mintedStakingShares, now);
235         _userStakes[beneficiary].push(newStake);
236
237         // 2. Global Accounting
238         totalStakingShares = totalStakingShares.add(mintedStakingShares);
239         // Already set in updateAccounting()
240         // _lastAccountingTimestampSec = now;
241
242         // interactions
243         require(_stakingPool.token.transferFrom(staker, address(_stakingPool), amount),
244             'TokenGeyser: transfer into staking pool failed');
245
246         emit Staked(beneficiary, amount, totalStakedFor(beneficiary), "");
247     }
```

The code meets the specification.

Formal Verification Request 15

_stakeFor require

28, Feb 2020
1692.6 ms

Line 194-200 in File TokenGeyser.sol

```
194   /*@CTK "_stakeFor require"
195     @tag assume_completion
196     @let uint totalStaked = _stakingPool.token._gonBalances[_stakingPool] / _stakingPool.
          token._gonsPerFragment
197     @post amount > 0
198     @post beneficiary != address(0)
199     @post totalStakingShares > 0 -> totalStaked > 0
200    */
```

Line 216-247 in File TokenGeyser.sol

```
216   function _stakeFor(address staker, address beneficiary, uint256 amount) private {
217       require(amount > 0, 'TokenGeyser: stake amount is zero');
218       require(beneficiary != address(0), 'TokenGeyser: beneficiary is zero address');
219       require(totalStakingShares == 0 || totalStaked() > 0,
220               'TokenGeyser: Invalid state. Staking shares exist, but no staking tokens do');
221
222       uint256 mintedStakingShares = (totalStakingShares > 0)
223           ? totalStakingShares.mul(amount).div(totalStaked())
224           : amount.mul(_initialSharesPerToken);
225       require(mintedStakingShares > 0, 'TokenGeyser: Stake amount is too small');
226
227       updateAccounting();
228
229       // 1. User Accounting
230       UserTotals storage totals = _userTotals[beneficiary];
231       totals.stakingShares = totals.stakingShares.add(mintedStakingShares);
232       totals.lastAccountingTimestampSec = now;
233
234       Stake memory newStake = Stake(mintedStakingShares, now);
235       _userStakes[beneficiary].push(newStake);
236
237       // 2. Global Accounting
238       totalStakingShares = totalStakingShares.add(mintedStakingShares);
239       // Already set in updateAccounting()
240       // _lastAccountingTimestampSec = now;
241
242       // interactions
243       require(_stakingPool.token.transferFrom(staker, address(_stakingPool), amount),
244           'TokenGeyser: transfer into staking pool failed');
245
246       emit Staked(beneficiary, amount, totalStakedFor(beneficiary), "");
247   }
```

The code meets the specification.

Formal Verification Request 16

_stakeFor case 1

28, Feb 2020
25251.42 ms

Line 201-208 in File TokenGeyser.sol

```
/*@CTK "_stakeFor case 1"
  @tag assume_completion
  @pre totalStakingShares > 0
  @let uint totalStaked = _stakingPool.token._gonBalances[_stakingPool] / _stakingPool.
      token._gonsPerFragment
  @let uint mintedStakingShares = totalStakingShares * amount / totalStaked
  @post __post._userTotals[beneficiary].stakingShares == _userTotals[beneficiary].
      stakingShares + mintedStakingShares
  @post __post.totalStakingShares == totalStakingShares + mintedStakingShares
 */
```

Line 216-247 in File TokenGeyser.sol

```
function _stakeFor(address staker, address beneficiary, uint256 amount) private {
    require(amount > 0, 'TokenGeyser: stake amount is zero');
    require(beneficiary != address(0), 'TokenGeyser: beneficiary is zero address');
    require(totalStakingShares == 0 || totalStaked() > 0,
            'TokenGeyser: Invalid state. Staking shares exist, but no staking tokens do');

    uint256 mintedStakingShares = (totalStakingShares > 0)
        ? totalStakingShares.mul(amount).div(totalStaked())
        : amount.mul(_initialSharesPerToken);
    require(mintedStakingShares > 0, 'TokenGeyser: Stake amount is too small');

    updateAccounting();

    // 1. User Accounting
    UserTotals storage totals = _userTotals[beneficiary];
    totals.stakingShares = totals.stakingShares.add(mintedStakingShares);
    totals.lastAccountingTimestampSec = now;

    Stake memory newStake = Stake(mintedStakingShares, now);
    _userStakes[beneficiary].push(newStake);

    // 2. Global Accounting
    totalStakingShares = totalStakingShares.add(mintedStakingShares);
    // Already set in updateAccounting()
    // _lastAccountingTimestampSec = now;

    // interactions
    require(_stakingPool.token.transferFrom(staker, address(_stakingPool), amount),
        'TokenGeyser: transfer into staking pool failed');

    emit Staked(beneficiary, amount, totalStakedFor(beneficiary), "");
}
```

The code meets the specification.

Formal Verification Request 17

_stakeFor case 2

28, Feb 2020
14225.36 ms

Line 209-215 in File TokenGeyser.sol

```
209   /*@CTK "_stakeFor case 2"
210     @tag assume_completion
211     @pre totalStakingShares == 0
212     @let uint mintedStakingShares = amount * _initialSharesPerToken
213     @post __post._userTotals[beneficiary].stakingShares == _userTotals[beneficiary].
          stakingShares + mintedStakingShares
214     @post __post.totalStakingShares == totalStakingShares + mintedStakingShares
215    */
```

Line 216-247 in File TokenGeyser.sol

```
216   function _stakeFor(address staker, address beneficiary, uint256 amount) private {
217      require(amount > 0, 'TokenGeyser: stake amount is zero');
218      require(beneficiary != address(0), 'TokenGeyser: beneficiary is zero address');
219      require(totalStakingShares == 0 || totalStaked() > 0,
220             'TokenGeyser: Invalid state. Staking shares exist, but no staking tokens do');
221
222      uint256 mintedStakingShares = (totalStakingShares > 0)
223          ? totalStakingShares.mul(amount).div(totalStaked())
224          : amount.mul(_initialSharesPerToken);
225      require(mintedStakingShares > 0, 'TokenGeyser: Stake amount is too small');
226
227      updateAccounting();
228
229      // 1. User Accounting
230      UserTotals storage totals = _userTotals[beneficiary];
231      totals.stakingShares = totals.stakingShares.add(mintedStakingShares);
232      totals.lastAccountingTimestampSec = now;
233
234      Stake memory newStake = Stake(mintedStakingShares, now);
235      _userStakes[beneficiary].push(newStake);
236
237      // 2. Global Accounting
238      totalStakingShares = totalStakingShares.add(mintedStakingShares);
239      // Already set in updateAccounting()
240      // _lastAccountingTimestampSec = now;
241
242      // interactions
243      require(_stakingPool.token.transferFrom(staker, address(_stakingPool), amount),
244          'TokenGeyser: transfer into staking pool failed');
245
246      emit Staked(beneficiary, amount, totalStakedFor(beneficiary), "");
247   }
```

The code meets the specification.

Formal Verification Request 18

computeNewReward case 0

28, Feb 2020
379.4 ms

Line 362-368 in File TokenGeyser.sol

```
362     /*@CTK "computeNewReward case 0"
363       @tag assume_completion
364       @pre stakeTimeSec >= bonusPeriodSec
365       @let uint totalUnlocked = _unlockedPool.token._gonBalances[_unlockedPool] /
              _unlockedPool.token._gonsPerFragment
366       @let uint newRewardTokens = totalUnlocked * stakingShareSeconds /
              _totalStakingShareSeconds
367       @post __return == currentRewardTokens + newRewardTokens
368      */
```

Line 378-398 in File TokenGeyser.sol

```
378     function computeNewReward(uint256 currentRewardTokens,
379                               uint256 stakingShareSeconds,
380                               uint256 stakeTimeSec) private view returns (uint256) {
381
382         uint256 newRewardTokens =
383             totalUnlocked()
384             .mul(stakingShareSeconds)
385             .div(_totalStakingShareSeconds);
386
387         if (stakeTimeSec >= bonusPeriodSec) {
388             return currentRewardTokens.add(newRewardTokens);
389         }
390
391         uint256 oneHundredPct = 10**BONUS_DECIMALS;
392         uint256 bonusedReward =
393             startBonus
394             .add(oneHundredPct.sub(startBonus).mul(stakeTimeSec).div(bonusPeriodSec))
395             .mul(newRewardTokens)
396             .div(oneHundredPct);
397         return currentRewardTokens.add(bonusedReward);
398     }
```

The code meets the specification.

Formal Verification Request 19

computeNewReward case 1

28, Feb 2020
163.97 ms

Line 369-377 in File TokenGeyser.sol

```
369     /*@CTK "computeNewReward case 1"
370       @tag assume_completion
371       @pre stakeTimeSec < bonusPeriodSec
```

```
    @let uint totalUnlocked = _unlockedPool.token._gonBalances[_unlockedPool] /
        _unlockedPool.token._gonsPerFragment
    @let uint newRewardTokens = totalUnlocked * stakingShareSeconds /
        _totalStakingShareSeconds
    @let uint oneHundredPct = 10**BONUS_DECIMALS
    @let uint bonusedReward = (startBonus + (oneHundredPct - startBonus) * stakeTimeSec /
        bonusPeriodSec) * newRewardTokens / oneHundredPct
    @post __return == currentRewardTokens + bonusedReward
   */
```

Line 378-398 in File TokenGeyser.sol

```
  function computeNewReward(uint256 currentRewardTokens,
                            uint256 stakingShareSeconds,
                            uint256 stakeTimeSec) private view returns (uint256) {

    uint256 newRewardTokens =
        totalUnlocked()
        .mul(stakingShareSeconds)
        .div(_totalStakingShareSeconds);

    if (stakeTimeSec >= bonusPeriodSec) {
        return currentRewardTokens.add(newRewardTokens);
    }

    uint256 oneHundredPct = 10**BONUS_DECIMALS;
    uint256 bonusedReward =
        startBonus
        .add(oneHundredPct.sub(startBonus).mul(stakeTimeSec).div(bonusPeriodSec))
        .mul(newRewardTokens)
        .div(oneHundredPct);
    return currentRewardTokens.add(bonusedReward);
  }
```

The code meets the specification.

Formal Verification Request 20

totalStakedFor

28, Feb 2020
27899.06 ms

Line 404-409 in File TokenGeyser.sol

```
  /*@CTK "totalStakedFor"
    @tag assume_completion
    @let uint totalStaked = _stakingPool.token._gonBalances[_stakingPool] / _stakingPool.
        token._gonsPerFragment
    @post totalStakingShares == 0 -> __return == 0
    @post totalStakingShares > 0 -> __return == totalStaked * _userTotals[addr].
        stakingShares / totalStakingShares
   */
```

Line 410-413 in File TokenGeyser.sol

```
  function totalStakedFor(address addr) public view returns (uint256) {
    return totalStakingShares > 0 ?
```

```
412             totalStaked().mul(_userTotals[addr].stakingShares).div(totalStakingShares) : 0;
413     }
```

The code meets the specification.

Formal Verification Request 21

totalStaked

28, Feb 2020
7.52 ms

Line 418-421 in File TokenGeyser.sol

```
418     /*@CTK "totalStaked"
419       @tag assume_completion
420       @post __return == _stakingPool.token._gonBalances[_stakingPool] / _stakingPool.token.
              _gonsPerFragment
421      */
```

Line 422-424 in File TokenGeyser.sol

```
422     function totalStaked() public view returns (uint256) {
423         return _stakingPool.balance();
424     }
```

The code meets the specification.

Formal Verification Request 22

token

28, Feb 2020
16.34 ms

Line 431-433 in File TokenGeyser.sol

```
431     /*@CTK "token"
432       @post __return == _stakingPool.token
433      */
```

Line 434-436 in File TokenGeyser.sol

```
434     function token() external view returns (address) {
435         return address(getStakingToken());
436     }
```

The code meets the specification.

Formal Verification Request 23

If method completes, integer overflow would not happen.

28, Feb 2020
2370.09 ms

Line 448 in File TokenGeyser.sol

```
448    //@CTK NO_OVERFLOW
```

Line 458-494 in File TokenGeyser.sol

```
458    function updateAccounting() public returns (
459        uint256, uint256, uint256, uint256, uint256, uint256) {
460
461        unlockTokens();
462
463        // Global accounting
464        uint256 newStakingShareSeconds =
465            now
466            .sub(_lastAccountingTimestampSec)
467            .mul(totalStakingShares);
468        _totalStakingShareSeconds = _totalStakingShareSeconds.add(newStakingShareSeconds);
469        _lastAccountingTimestampSec = now;
470
471        // User Accounting
472        UserTotals storage totals = _userTotals[msg.sender];
473        uint256 newUserStakingShareSeconds =
474            now
475            .sub(totals.lastAccountingTimestampSec)
476            .mul(totals.stakingShares);
477        totals.stakingShareSeconds =
478            totals.stakingShareSeconds
479            .add(newUserStakingShareSeconds);
480        totals.lastAccountingTimestampSec = now;
481
482        uint256 totalUserRewards = (_totalStakingShareSeconds > 0)
483            ? totalUnlocked().mul(totals.stakingShareSeconds).div(_totalStakingShareSeconds)
484            : 0;
485
486        return (
487            totalLocked(),
488            totalUnlocked(),
489            totals.stakingShareSeconds,
490            _totalStakingShareSeconds,
491            totalUserRewards,
492            now
493        );
494    }
```

✅ The code meets the specification.

Formal Verification Request 24

Buffer overflow / array index out of bound would never happen.

📅 28, Feb 2020
⏱ 614.9 ms

Line 449 in File TokenGeyser.sol

```
449    //@CTK NO_BUF_OVERFLOW
```

Line 458-494 in File TokenGeyser.sol

```
458    function updateAccounting() public returns (
```

```
459         uint256, uint256, uint256, uint256, uint256, uint256) {
460
461         unlockTokens();
462
463         // Global accounting
464         uint256 newStakingShareSeconds =
465             now
466             .sub(_lastAccountingTimestampSec)
467             .mul(totalStakingShares);
468         _totalStakingShareSeconds = _totalStakingShareSeconds.add(newStakingShareSeconds);
469         _lastAccountingTimestampSec = now;
470
471         // User Accounting
472         UserTotals storage totals = _userTotals[msg.sender];
473         uint256 newUserStakingShareSeconds =
474             now
475             .sub(totals.lastAccountingTimestampSec)
476             .mul(totals.stakingShares);
477         totals.stakingShareSeconds =
478             totals.stakingShareSeconds
479             .add(newUserStakingShareSeconds);
480         totals.lastAccountingTimestampSec = now;
481
482         uint256 totalUserRewards = (_totalStakingShareSeconds > 0)
483             ? totalUnlocked().mul(totals.stakingShareSeconds).div(_totalStakingShareSeconds)
484             : 0;
485
486         return (
487             totalLocked(),
488             totalUnlocked(),
489             totals.stakingShareSeconds,
490             _totalStakingShareSeconds,
491             totalUserRewards,
492             now
493         );
494     }
```

✅ The code meets the specification.

Formal Verification Request 25

Method will not encounter an assertion failure.

📅 28, Feb 2020
⏱ 865.11 ms

Line 450 in File TokenGeyser.sol

```
450     //@CTK NO_ASF
```

Line 458-494 in File TokenGeyser.sol

```
458     function updateAccounting() public returns (
459         uint256, uint256, uint256, uint256, uint256, uint256) {
460
461         unlockTokens();
462
463         // Global accounting
```

```
        uint256 newStakingShareSeconds =
            now
            .sub(_lastAccountingTimestampSec)
            .mul(totalStakingShares);
        _totalStakingShareSeconds = _totalStakingShareSeconds.add(newStakingShareSeconds);
        _lastAccountingTimestampSec = now;

        // User Accounting
        UserTotals storage totals = _userTotals[msg.sender];
        uint256 newUserStakingShareSeconds =
            now
            .sub(totals.lastAccountingTimestampSec)
            .mul(totals.stakingShares);
        totals.stakingShareSeconds =
            totals.stakingShareSeconds
            .add(newUserStakingShareSeconds);
        totals.lastAccountingTimestampSec = now;

        uint256 totalUserRewards = (_totalStakingShareSeconds > 0)
            ? totalUnlocked().mul(totals.stakingShareSeconds).div(_totalStakingShareSeconds)
            : 0;

        return (
            totalLocked(),
            totalUnlocked(),
            totals.stakingShareSeconds,
            _totalStakingShareSeconds,
            totalUserRewards,
            now
        );
    }
```

✅ The code meets the specification.

Formal Verification Request 26

totalLocked

📅 28, Feb 2020
⏱ 5.38 ms

Line 499-502 in File TokenGeyser.sol

```
499 /*@CTK "totalLocked"
500   @tag assume_completion
501   @post __return == _lockedPool.token._gonBalances[_lockedPool] / _lockedPool.token.
          _gonsPerFragment
502  */
```

Line 503-505 in File TokenGeyser.sol

```
503 function totalLocked() public view returns (uint256) {
504     return _lockedPool.balance();
505 }
```

✅ The code meets the specification.

Formal Verification Request 27

totalUnlocked

28, Feb 2020
4.62 ms

Line 510-513 in File TokenGeyser.sol

```
510 /*@CTK "totalUnlocked"
511   @tag assume_completion
512   @post __return == _unlockedPool.token._gonBalances[_unlockedPool] / _unlockedPool.
        token._gonsPerFragment
513  */
```

Line 514-516 in File TokenGeyser.sol

```
514 function totalUnlocked() public view returns (uint256) {
515     return _unlockedPool.balance();
516 }
```

The code meets the specification.

Formal Verification Request 28

unlockScheduleCount

28, Feb 2020
4.5 ms

Line 521-523 in File TokenGeyser.sol

```
521 /*@CTK "unlockScheduleCount"
522   @post __return == unlockSchedules.length
523  */
```

Line 524-526 in File TokenGeyser.sol

```
524 function unlockScheduleCount() public view returns (uint256) {
525     return unlockSchedules.length;
526 }
```

The code meets the specification.

Formal Verification Request 29

If method completes, integer overflow would not happen.

28, Feb 2020
9402.0 ms

Line 535 in File TokenGeyser.sol

```
535 //@CTK NO_OVERFLOW
```

Line 538-562 in File TokenGeyser.sol

```
538     function lockTokens(uint256 amount, uint256 durationSec) external onlyOwner {
539         require(unlockSchedules.length < _maxUnlockSchedules,
540             'TokenGeyser: reached maximum unlock schedules');
541
542         // Update lockedTokens amount before using it in computations after.
543         updateAccounting();
544
545         uint256 lockedTokens = totalLocked();
546         uint256 mintedLockedShares = (lockedTokens > 0)
547             ? totalLockedShares.mul(amount).div(lockedTokens)
548             : amount.mul(_initialSharesPerToken);
549
550         UnlockSchedule memory schedule;
551         schedule.initialLockedShares = mintedLockedShares;
552         schedule.lastUnlockTimestampSec = now;
553         schedule.endAtSec = now.add(durationSec);
554         schedule.durationSec = durationSec;
555         unlockSchedules.push(schedule);
556
557         totalLockedShares = totalLockedShares.add(mintedLockedShares);
558
559         require(_lockedPool.token.transferFrom(msg.sender, address(_lockedPool), amount),
560             'TokenGeyser: transfer into locked pool failed');
561         emit TokensLocked(amount, durationSec, totalLocked());
562     }
```

✅ The code meets the specification.

Formal Verification Request 30

Buffer overflow / array index out of bound would never happen.

📅 28, Feb 2020
⏱ 3952.08 ms

Line 536 in File TokenGeyser.sol

```
536     //@CTK NO_BUF_OVERFLOW
```

Line 538-562 in File TokenGeyser.sol

```
538     function lockTokens(uint256 amount, uint256 durationSec) external onlyOwner {
539         require(unlockSchedules.length < _maxUnlockSchedules,
540             'TokenGeyser: reached maximum unlock schedules');
541
542         // Update lockedTokens amount before using it in computations after.
543         updateAccounting();
544
545         uint256 lockedTokens = totalLocked();
546         uint256 mintedLockedShares = (lockedTokens > 0)
547             ? totalLockedShares.mul(amount).div(lockedTokens)
548             : amount.mul(_initialSharesPerToken);
549
550         UnlockSchedule memory schedule;
551         schedule.initialLockedShares = mintedLockedShares;
552         schedule.lastUnlockTimestampSec = now;
553         schedule.endAtSec = now.add(durationSec);
554         schedule.durationSec = durationSec;
```

```
        unlockSchedules.push(schedule);

        totalLockedShares = totalLockedShares.add(mintedLockedShares);

        require(_lockedPool.token.transferFrom(msg.sender, address(_lockedPool), amount),
            'TokenGeyser: transfer into locked pool failed');
        emit TokensLocked(amount, durationSec, totalLocked());
    }
```

The code meets the specification.

Formal Verification Request 31

Method will not encounter an assertion failure.

28, Feb 2020
2687.08 ms

Line 537 in File TokenGeyser.sol

```
537     //@CTK NO_ASF
```

Line 538-562 in File TokenGeyser.sol

```
538     function lockTokens(uint256 amount, uint256 durationSec) external onlyOwner {
539         require(unlockSchedules.length < _maxUnlockSchedules,
540             'TokenGeyser: reached maximum unlock schedules');
541
542         // Update lockedTokens amount before using it in computations after.
543         updateAccounting();
544
545         uint256 lockedTokens = totalLocked();
546         uint256 mintedLockedShares = (lockedTokens > 0)
547             ? totalLockedShares.mul(amount).div(lockedTokens)
548             : amount.mul(_initialSharesPerToken);
549
550         UnlockSchedule memory schedule;
551         schedule.initialLockedShares = mintedLockedShares;
552         schedule.lastUnlockTimestampSec = now;
553         schedule.endAtSec = now.add(durationSec);
554         schedule.durationSec = durationSec;
555         unlockSchedules.push(schedule);
556
557         totalLockedShares = totalLockedShares.add(mintedLockedShares);
558
559         require(_lockedPool.token.transferFrom(msg.sender, address(_lockedPool), amount),
560             'TokenGeyser: transfer into locked pool failed');
561         emit TokensLocked(amount, durationSec, totalLocked());
562     }
```

The code meets the specification.

Formal Verification Request 32

If method completes, integer overflow would not happen.

28, Feb 2020

359.09 ms

Line 569 in File TokenGeyser.sol

```
569    //@CTK NO_OVERFLOW
```

Line 588-613 in File TokenGeyser.sol

```
588    function unlockTokens() public returns (uint256) {
589        uint256 unlockedTokens = 0;
590        uint256 lockedTokens = totalLocked();
591
592        if (totalLockedShares == 0) {
593            unlockedTokens = lockedTokens;
594        } else {
595            uint256 unlockedShares = 0;
596            /*@CTK "unlockTokens for"
597              @inv s <= unlockSchedules.length
598             */
599            for (uint256 s = 0; s < unlockSchedules.length; s++) {
600                unlockedShares += unlockScheduleShares(s);
601            }
602            unlockedTokens = unlockedShares.mul(lockedTokens).div(totalLockedShares);
603            totalLockedShares = totalLockedShares.sub(unlockedShares);
604        }
605
606        if (unlockedTokens > 0) {
607            require(_lockedPool.transfer(address(_unlockedPool), unlockedTokens),
608                'TokenGeyser: transfer out of locked pool failed');
609            emit TokensUnlocked(unlockedTokens, totalLocked());
610        }
611
612        return unlockedTokens;
613    }
```

The code meets the specification.

Formal Verification Request 33

Buffer overflow / array index out of bound would never happen.

28, Feb 2020
109.84 ms

Line 570 in File TokenGeyser.sol

```
570    //@CTK NO_BUF_OVERFLOW
```

Line 588-613 in File TokenGeyser.sol

```
588    function unlockTokens() public returns (uint256) {
589        uint256 unlockedTokens = 0;
590        uint256 lockedTokens = totalLocked();
591
592        if (totalLockedShares == 0) {
593            unlockedTokens = lockedTokens;
594        } else {
595            uint256 unlockedShares = 0;
```

```
        /*@CTK "unlockTokens for"
          @inv s <= unlockSchedules.length
         */
        for (uint256 s = 0; s < unlockSchedules.length; s++) {
            unlockedShares += unlockScheduleShares(s);
        }
        unlockedTokens = unlockedShares.mul(lockedTokens).div(totalLockedShares);
        totalLockedShares = totalLockedShares.sub(unlockedShares);
    }

    if (unlockedTokens > 0) {
        require(_lockedPool.transfer(address(_unlockedPool), unlockedTokens),
            'TokenGeyser: transfer out of locked pool failed');
        emit TokensUnlocked(unlockedTokens, totalLocked());
    }

    return unlockedTokens;
}
```

✅ The code meets the specification.

Formal Verification Request 34

Method will not encounter an assertion failure.

📅 28, Feb 2020
⏱ 122.97 ms

Line 571 in File TokenGeyser.sol

```
571 //@CTK NO_ASF
```

Line 588-613 in File TokenGeyser.sol

```
588 function unlockTokens() public returns (uint256) {
589     uint256 unlockedTokens = 0;
590     uint256 lockedTokens = totalLocked();
591
592     if (totalLockedShares == 0) {
593         unlockedTokens = lockedTokens;
594     } else {
595         uint256 unlockedShares = 0;
596         /*@CTK "unlockTokens for"
597           @inv s <= unlockSchedules.length
598          */
599         for (uint256 s = 0; s < unlockSchedules.length; s++) {
600             unlockedShares += unlockScheduleShares(s);
601         }
602         unlockedTokens = unlockedShares.mul(lockedTokens).div(totalLockedShares);
603         totalLockedShares = totalLockedShares.sub(unlockedShares);
604     }
605
606     if (unlockedTokens > 0) {
607         require(_lockedPool.transfer(address(_unlockedPool), unlockedTokens),
608             'TokenGeyser: transfer out of locked pool failed');
609         emit TokensUnlocked(unlockedTokens, totalLocked());
610     }
611
```

```
        return unlockedTokens;
    }
```

The code meets the specification.

Formal Verification Request 35

unlockTokens case 1

28, Feb 2020
29.59 ms

Line 572-577 in File TokenGeyser.sol

```
572     /*@CTK "unlockTokens case 1"
573       @tag assume_completion
574       @pre totalLockedShares == 0
575       @pre _lockedPool.token._gonBalances[_lockedPool] == 0
576       @post __return == 0
577      */
```

Line 588-613 in File TokenGeyser.sol

```
588     function unlockTokens() public returns (uint256) {
589         uint256 unlockedTokens = 0;
590         uint256 lockedTokens = totalLocked();
591 
592         if (totalLockedShares == 0) {
593             unlockedTokens = lockedTokens;
594         } else {
595             uint256 unlockedShares = 0;
596             /*@CTK "unlockTokens for"
597               @inv s <= unlockSchedules.length
598              */
599             for (uint256 s = 0; s < unlockSchedules.length; s++) {
600                 unlockedShares += unlockScheduleShares(s);
601             }
602             unlockedTokens = unlockedShares.mul(lockedTokens).div(totalLockedShares);
603             totalLockedShares = totalLockedShares.sub(unlockedShares);
604         }
605 
606         if (unlockedTokens > 0) {
607             require(_lockedPool.transfer(address(_unlockedPool), unlockedTokens),
608                 'TokenGeyser: transfer out of locked pool failed');
609             emit TokensUnlocked(unlockedTokens, totalLocked());
610         }
611 
612         return unlockedTokens;
613     }
```

The code meets the specification.

Formal Verification Request 36

unlockTokens case 2

28, Feb 2020

28.99 ms

Line 578-587 in File TokenGeyser.sol

```
578   /*@CTK "unlockTokens case 2"
579     @tag assume_completion
580     @pre totalLockedShares == 0
581     @pre _lockedPool.token._gonBalances[_lockedPool] > 0
582     @pre _lockedPool != _unlockedPool
583     @let uint lockedTokens = _lockedPool.token._gonBalances[_lockedPool] / _lockedPool.
            token._gonsPerFragment
584     //@post __post._unlockedPool.token._gonBalances[_unlockedPool] == _unlockedPool.token.
            _gonBalances[_unlockedPool] + lockedTokens * _lockedPool.token._gonsPerFragment
585     //@post __post._lockedPool.token._gonBalances[_lockedPool] == _lockedPool.token.
            _gonBalances[_lockedPool] - lockedTokens * _lockedPool.token._gonsPerFragment
586     @post __return == lockedTokens
587    */
```

Line 588-613 in File TokenGeyser.sol

```
588   function unlockTokens() public returns (uint256) {
589       uint256 unlockedTokens = 0;
590       uint256 lockedTokens = totalLocked();
591
592       if (totalLockedShares == 0) {
593           unlockedTokens = lockedTokens;
594       } else {
595           uint256 unlockedShares = 0;
596           /*@CTK "unlockTokens for"
597             @inv s <= unlockSchedules.length
598            */
599           for (uint256 s = 0; s < unlockSchedules.length; s++) {
600               unlockedShares += unlockScheduleShares(s);
601           }
602           unlockedTokens = unlockedShares.mul(lockedTokens).div(totalLockedShares);
603           totalLockedShares = totalLockedShares.sub(unlockedShares);
604       }
605
606       if (unlockedTokens > 0) {
607           require(_lockedPool.transfer(address(_unlockedPool), unlockedTokens),
608               'TokenGeyser: transfer out of locked pool failed');
609           emit TokensUnlocked(unlockedTokens, totalLocked());
610       }
611
612       return unlockedTokens;
613   }
```

The code meets the specification.

Formal Verification Request 37

If method completes, integer overflow would not happen.

28, Feb 2020
35695.85 ms

Line 622 in File TokenGeyser.sol

```
    //@CTK FAIL NO_OVERFLOW
```

Line 639-660 in File TokenGeyser.sol

```
    function unlockScheduleShares(uint256 s) private returns (uint256) {
        UnlockSchedule storage schedule = unlockSchedules[s];

        if(schedule.unlockedShares >= schedule.initialLockedShares){
            return 0;
        }

        uint256 sharesToUnlock = 0;
        // Special case to handle any leftover dust from integer division
        if(now >= schedule.endAtSec){
            sharesToUnlock = (schedule.initialLockedShares - schedule.unlockedShares);
            schedule.lastUnlockTimestampSec = schedule.endAtSec;
        } else {
            sharesToUnlock = now.sub(schedule.lastUnlockTimestampSec)
                .mul(schedule.initialLockedShares)
                .div(schedule.durationSec);
            schedule.lastUnlockTimestampSec = now;
        }

        schedule.unlockedShares += sharesToUnlock;
        return sharesToUnlock;
    }
```

This code violates the specification.

```
Counter Example:
Before Execution:
    Input = {
        s = 0
    }
    This = 0
    Internal = {
        __has_assertion_failure = false
        __has_buf_overflow = false
        __has_overflow = false
        __has_returned = false
        __reverted = false
        msg = {
          "gas": 0,
          "sender": 0,
          "value": 0
        }
    }
    Other = {
        BONUS_DECIMALS = 2
        __return = 0
        block = {
          "number": 0,
          "timestamp": 37
        }
    }
    Address_Map = [
      {
        "key": 0,
        "value": {
          "contract_name": "TokenGeyser",
```

```
        "balance": 0,
        "contract": {
          "_stakingPool": 24,
          "_unlockedPool": 0,
          "_lockedPool": 4,
          "startBonus": 4,
          "bonusPeriodSec": 0,
          "totalLockedShares": 0,
          "totalStakingShares": 64,
          "_totalStakingShareSeconds": 0,
          "_lastAccountingTimestampSec": 0,
          "_maxUnlockSchedules": 0,
          "_initialSharesPerToken": 0,
          "_userTotals": [
            {
              "key": 0,
              "value": {
                "stakingShares": 0,
                "stakingShareSeconds": 0,
                "lastAccountingTimestampSec": 4
              }
            },
            {
              "key": "ALL_OTHERS",
              "value": {
                "stakingShares": 128,
                "stakingShareSeconds": 128,
                "lastAccountingTimestampSec": 128
              }
            }
          ],
          "_userStakes": [
            {
              "key": 0,
              "value": []
            },
            {
              "key": "ALL_OTHERS",
              "value": [
                {
                  "key": "ALL_OTHERS",
                  "value": {
                    "stakingShares": 128,
                    "timestampSec": 128
                  }
                }
              ]
            }
          ],
          "unlockSchedules": [
            {
              "initialLockedShares": 33,
              "unlockedShares": 32,
              "lastUnlockTimestampSec": 30,
              "endAtSec": 44,
              "durationSec": 1
            }
          ],
```

```
          "_owner": 0
        }
      }
    },
    {
      "key": "ALL_OTHERS",
      "value": "EmptyAddress"
    }
  ]

After Execution:
  Input = {
      s = 0
  }
  This = 0
  Internal = {
      __has_assertion_failure = false
      __has_buf_overflow = false
      __has_overflow = true
      __has_returned = true
      __reverted = false
      msg = {
        "gas": 0,
        "sender": 0,
        "value": 0
      }
  }
  Other = {
      BONUS_DECIMALS = 2
      __return = 231
      block = {
        "number": 0,
        "timestamp": 37
      }
  }
  Address_Map = [
    {
      "key": 0,
      "value": {
        "contract_name": "TokenGeyser",
        "balance": 0,
        "contract": {
          "_stakingPool": 24,
          "_unlockedPool": 0,
          "_lockedPool": 4,
          "startBonus": 4,
          "bonusPeriodSec": 0,
          "totalLockedShares": 0,
          "totalStakingShares": 64,
          "_totalStakingShareSeconds": 0,
          "_lastAccountingTimestampSec": 0,
          "_maxUnlockSchedules": 0,
          "_initialSharesPerToken": 0,
          "_userTotals": [
            {
              "key": 0,
              "value": {
                "stakingShares": 0,
```

```
                "stakingShareSeconds": 0,
                "lastAccountingTimestampSec": 4
              }
            },
            {
              "key": "ALL_OTHERS",
              "value": {
                "stakingShares": 128,
                "stakingShareSeconds": 128,
                "lastAccountingTimestampSec": 128
              }
            }
          ],
          "_userStakes": [
            {
              "key": 0,
              "value": []
            },
            {
              "key": "ALL_OTHERS",
              "value": [
                {
                  "key": "ALL_OTHERS",
                  "value": {
                    "stakingShares": 128,
                    "timestampSec": 128
                  }
                }
              ]
            }
          ],
          "unlockSchedules": [
            {
              "initialLockedShares": 33,
              "unlockedShares": 7,
              "lastUnlockTimestampSec": 37,
              "endAtSec": 44,
              "durationSec": 1
            }
          ],
          "_owner": 0
        }
      }
    },
    {
      "key": "ALL_OTHERS",
      "value": "EmptyAddress"
    }
  ]
```


Formal Verification Request 38

Buffer overflow / array index out of bound would never happen.

📅 28, Feb 2020
⏱ 655.94 ms

Line 623 in File TokenGeyser.sol

```
623     //@CTK FAIL NO_BUF_OVERFLOW
```

Line 639-660 in File TokenGeyser.sol

```
639     function unlockScheduleShares(uint256 s) private returns (uint256) {
640         UnlockSchedule storage schedule = unlockSchedules[s];
641 
642         if(schedule.unlockedShares >= schedule.initialLockedShares){
643             return 0;
644         }
645 
646         uint256 sharesToUnlock = 0;
647         // Special case to handle any leftover dust from integer division
648         if(now >= schedule.endAtSec){
649             sharesToUnlock = (schedule.initialLockedShares - schedule.unlockedShares);
650             schedule.lastUnlockTimestampSec = schedule.endAtSec;
651         } else {
652             sharesToUnlock = now.sub(schedule.lastUnlockTimestampSec)
653                 .mul(schedule.initialLockedShares)
654                 .div(schedule.durationSec);
655             schedule.lastUnlockTimestampSec = now;
656         }
657 
658         schedule.unlockedShares += sharesToUnlock;
659         return sharesToUnlock;
660     }
```

This code violates the specification.

```
1  Counter Example:
2  Before Execution:
3      Input = {
4          s = 128
5      }
6      This = 0
7      Internal = {
8          __has_assertion_failure = false
9          __has_buf_overflow = false
10         __has_overflow = false
11         __has_returned = false
12         __reverted = false
13         msg = {
14           "gas": 0,
15           "sender": 0,
16           "value": 0
17         }
18     }
19     Other = {
20         BONUS_DECIMALS = 2
21         __return = 0
22         block = {
23           "number": 0,
24           "timestamp": 8
25         }
26     }
27     Address_Map = [
28       {
```

```
      "key": 0,
      "value": {
        "contract_name": "TokenGeyser",
        "balance": 0,
        "contract": {
          "_stakingPool": 0,
          "_unlockedPool": 0,
          "_lockedPool": 0,
          "startBonus": 0,
          "bonusPeriodSec": 0,
          "totalLockedShares": 0,
          "totalStakingShares": 0,
          "_totalStakingShareSeconds": 0,
          "_lastAccountingTimestampSec": 0,
          "_maxUnlockSchedules": 0,
          "_initialSharesPerToken": 0,
          "_userTotals": [
            {
              "key": 0,
              "value": {
                "stakingShares": 0,
                "stakingShareSeconds": 0,
                "lastAccountingTimestampSec": 0
              }
            },
            {
              "key": "ALL_OTHERS",
              "value": {
                "stakingShares": 128,
                "stakingShareSeconds": 128,
                "lastAccountingTimestampSec": 128
              }
            }
          ],
          "_userStakes": [
            {
              "key": 0,
              "value": []
            },
            {
              "key": "ALL_OTHERS",
              "value": [
                {
                  "key": "ALL_OTHERS",
                  "value": {
                    "stakingShares": 128,
                    "timestampSec": 128
                  }
                }
              ]
            }
          ],
          "unlockSchedules": [
            {
              "initialLockedShares": 0,
              "unlockedShares": 0,
              "lastUnlockTimestampSec": 0,
              "endAtSec": 0,
```

```
              "durationSec": 0
            }
          ],
          "_owner": 0
        }
      }
    },
    {
      "key": "ALL_OTHERS",
      "value": "EmptyAddress"
    }
  ]

After Execution:
  Input = {
      s = 128
  }
  This = 0
  Internal = {
      __has_assertion_failure = false
      __has_buf_overflow = true
      __has_overflow = false
      __has_returned = true
      __reverted = false
      msg = {
        "gas": 0,
        "sender": 0,
        "value": 0
      }
  }
  Other = {
      BONUS_DECIMALS = 2
      __return = 0
      block = {
        "number": 0,
        "timestamp": 8
      }
  }
  Address_Map = [
    {
      "key": 0,
      "value": {
        "contract_name": "TokenGeyser",
        "balance": 0,
        "contract": {
          "_stakingPool": 0,
          "_unlockedPool": 0,
          "_lockedPool": 0,
          "startBonus": 0,
          "bonusPeriodSec": 0,
          "totalLockedShares": 0,
          "totalStakingShares": 0,
          "_totalStakingShareSeconds": 0,
          "_lastAccountingTimestampSec": 0,
          "_maxUnlockSchedules": 0,
          "_initialSharesPerToken": 0,
          "_userTotals": [
            {
```

```
              "key": 0,
              "value": {
                "stakingShares": 0,
                "stakingShareSeconds": 0,
                "lastAccountingTimestampSec": 0
              }
            },
            {
              "key": "ALL_OTHERS",
              "value": {
                "stakingShares": 128,
                "stakingShareSeconds": 128,
                "lastAccountingTimestampSec": 128
              }
            }
          ],
          "_userStakes": [
            {
              "key": 0,
              "value": []
            },
            {
              "key": "ALL_OTHERS",
              "value": [
                {
                  "key": "ALL_OTHERS",
                  "value": {
                    "stakingShares": 128,
                    "timestampSec": 128
                  }
                }
              ]
            }
          ],
          "unlockSchedules": [
            {
              "initialLockedShares": 0,
              "unlockedShares": 0,
              "lastUnlockTimestampSec": 0,
              "endAtSec": 0,
              "durationSec": 0
            }
          ],
          "_owner": 0
        }
      }
    },
    {
      "key": "ALL_OTHERS",
      "value": "EmptyAddress"
    }
  ]
```

Formal Verification Request 39

Method will not encounter an assertion failure.

28, Feb 2020
10.69 ms

Line 624 in File TokenGeyser.sol

```
624    //@CTK NO_ASF
```

Line 639-660 in File TokenGeyser.sol

```
639    function unlockScheduleShares(uint256 s) private returns (uint256) {
640        UnlockSchedule storage schedule = unlockSchedules[s];
641
642        if(schedule.unlockedShares >= schedule.initialLockedShares){
643            return 0;
644        }
645
646        uint256 sharesToUnlock = 0;
647        // Special case to handle any leftover dust from integer division
648        if(now >= schedule.endAtSec){
649            sharesToUnlock = (schedule.initialLockedShares - schedule.unlockedShares);
650            schedule.lastUnlockTimestampSec = schedule.endAtSec;
651        } else {
652            sharesToUnlock = now.sub(schedule.lastUnlockTimestampSec)
653                .mul(schedule.initialLockedShares)
654                .div(schedule.durationSec);
655            schedule.lastUnlockTimestampSec = now;
656        }
657
658        schedule.unlockedShares += sharesToUnlock;
659        return sharesToUnlock;
660    }
```

The code meets the specification.

Formal Verification Request 40

unlockScheduleShares case 1

28, Feb 2020
15.73 ms

Line 625-628 in File TokenGeyser.sol

```
625    /*@CTK "unlockScheduleShares case 1"
626      @pre unlockSchedules[s].unlockedShares >= unlockSchedules[s].initialLockedShares
627      @post __return == 0
628     */
```

Line 639-660 in File TokenGeyser.sol

```
639    function unlockScheduleShares(uint256 s) private returns (uint256) {
640        UnlockSchedule storage schedule = unlockSchedules[s];
641
642        if(schedule.unlockedShares >= schedule.initialLockedShares){
643            return 0;
```

```
    }

    uint256 sharesToUnlock = 0;
    // Special case to handle any leftover dust from integer division
    if(now >= schedule.endAtSec){
        sharesToUnlock = (schedule.initialLockedShares - schedule.unlockedShares);
        schedule.lastUnlockTimestampSec = schedule.endAtSec;
    } else {
        sharesToUnlock = now.sub(schedule.lastUnlockTimestampSec)
            .mul(schedule.initialLockedShares)
            .div(schedule.durationSec);
        schedule.lastUnlockTimestampSec = now;
    }

    schedule.unlockedShares += sharesToUnlock;
    return sharesToUnlock;
}
```

The code meets the specification.

Formal Verification Request 41

unlockScheduleShares case 2

28, Feb 2020
61611.96 ms

Line 629-638 in File TokenGeyser.sol

```
629 /*@CTK "unlockScheduleShares case 2"
630   @tag assume_completion
631   @pre unlockSchedules[s].unlockedShares < unlockSchedules[s].initialLockedShares
632   @post now >= unlockSchedules[s].endAtSec -> __post.unlockSchedules[s].
          lastUnlockTimestampSec == unlockSchedules[s].endAtSec
633   @post now >= unlockSchedules[s].endAtSec -> __post.unlockSchedules[s].unlockedShares
          == unlockSchedules[s].unlockedShares + unlockSchedules[s].initialLockedShares -
          unlockSchedules[s].unlockedShares
634   @post now >= unlockSchedules[s].endAtSec -> __return == unlockSchedules[s].
          initialLockedShares - unlockSchedules[s].unlockedShares
635   @post now < unlockSchedules[s].endAtSec -> __post.unlockSchedules[s].
          lastUnlockTimestampSec == now
636   @post now < unlockSchedules[s].endAtSec -> __post.unlockSchedules[s].unlockedShares ==
           unlockSchedules[s].unlockedShares + (now - unlockSchedules[s].
          lastUnlockTimestampSec) * unlockSchedules[s].initialLockedShares / unlockSchedules
          [s].durationSec
637   @post now < unlockSchedules[s].endAtSec -> __return == (now - unlockSchedules[s].
          lastUnlockTimestampSec) * unlockSchedules[s].initialLockedShares / unlockSchedules
          [s].durationSec
638  */
```

Line 639-660 in File TokenGeyser.sol

```
639 function unlockScheduleShares(uint256 s) private returns (uint256) {
640     UnlockSchedule storage schedule = unlockSchedules[s];
641
642     if(schedule.unlockedShares >= schedule.initialLockedShares){
643         return 0;
644     }
```

```
        uint256 sharesToUnlock = 0;
        // Special case to handle any leftover dust from integer division
        if(now >= schedule.endAtSec){
            sharesToUnlock = (schedule.initialLockedShares - schedule.unlockedShares);
            schedule.lastUnlockTimestampSec = schedule.endAtSec;
        } else {
            sharesToUnlock = now.sub(schedule.lastUnlockTimestampSec)
                .mul(schedule.initialLockedShares)
                .div(schedule.durationSec);
            schedule.lastUnlockTimestampSec = now;
        }

        schedule.unlockedShares += sharesToUnlock;
        return sharesToUnlock;
    }
```

The code meets the specification.

Formal Verification Request 42

unlockTokens for___Generated

28, Feb 2020
175.24 ms

(Loop) Line 596-598 in File TokenGeyser.sol

```
596         /*@CTK "unlockTokens for"
597           @inv s <= unlockSchedules.length
598          */
```

(Loop) Line 596-601 in File TokenGeyser.sol

```
596         /*@CTK "unlockTokens for"
597           @inv s <= unlockSchedules.length
598          */
599         for (uint256 s = 0; s < unlockSchedules.length; s++) {
600             unlockedShares += unlockScheduleShares(s);
601         }
```

The code meets the specification.

Source Code with CertiK Labels

File TokenPool.sol

```
pragma solidity 0.5.0;

import "../openzeppelin-solidity/contracts/ownership/Ownable.sol";
import "../dependencies/UFragments.sol";

/**
 * @title A simple holder of tokens.
 * This is a simple contract to hold tokens. It's useful in the case where a separate contract
 * needs to hold multiple distinct pools of the same token.
 */
contract TokenPool is Ownable {
    UFragments public token;

    /*@CTK "TokenPool"
      @post __post.token == _token
     */
    constructor(UFragments _token) public {
        token = _token;
    }

    /*@CTK "balance"
      @tag assume_completion
      @post __return == token._gonBalances[address(this)] / token._gonsPerFragment
     */
    function balance() public view returns (uint256) {
        return token.balanceOf(address(this));
    }

    /*@CTK "transfer"
      @tag assume_completion
      @pre msg.sender == _owner
      @pre msg.sender != address(0)
      @pre to != address(0)
      @post msg.sender != to -> __post.token._gonBalances[to] == token._gonBalances[to] + value * token._gonsPerFragment
      @post msg.sender != to -> __post.token._gonBalances[msg.sender] == token._gonBalances[msg.sender] - value * token._gonsPerFragment
      @post msg.sender == to -> __post.token._gonBalances[msg.sender] == token._gonBalances[msg.sender]
     */
    function transfer(address to, uint256 value) external onlyOwner returns (bool) {
        return token.transfer(to, value);
    }
}
```

File IStaking.sol

```
pragma solidity 0.5.0;

/**
 * @title Staking interface, as defined by EIP-900.
 * @dev https://github.com/ethereum/EIPs/blob/master/EIPS/eip-900.md
 */
contract IStaking {
```

```
    event Staked(address indexed user, uint256 amount, uint256 total, bytes data);
    event Unstaked(address indexed user, uint256 amount, uint256 total, bytes data);

    function stake(uint256 amount, bytes calldata data) external;
    function stakeFor(address user, uint256 amount, bytes calldata data) external;
    function unstake(uint256 amount, bytes calldata data) external;
    function totalStakedFor(address addr) public view returns (uint256);
    function totalStaked() public view returns (uint256);
    function token() external view returns (address);

    /**
     * @return False. This application does not support staking history.
     */
    /*@CTK "supportsHistory"
      @post __return == false
     */
    function supportsHistory() external pure returns (bool) {
        return false;
    }
}
```

File TokenGeyser.sol

```
pragma solidity 0.5.0;

import "../openzeppelin-solidity/contracts/math/SafeMath.sol";
import "../openzeppelin-solidity/contracts/ownership/Ownable.sol";
import "../dependencies/UFragments.sol";

import "./IStaking.sol";
import "./TokenPool.sol";

/**
 * @title Token Geyser
 * @dev A smart-contract based mechanism to distribute tokens over time, inspired loosely by
 *     Compound and Uniswap.
 *
 *     Distribution tokens are added to a locked pool in the contract and become unlocked over time
 *     according to a once-configurable unlock schedule. Once unlocked, they are available to be
 *     claimed by users.
 *
 *     A user may deposit tokens to accrue ownership share over the unlocked pool. This owner share
 *     is a function of the number of tokens deposited as well as the length of time deposited.
 *     Specifically, a user's share of the currently-unlocked pool equals their "deposit-seconds"
 *     divided by the global "deposit-seconds". This aligns the new token distribution with long
 *     term supporters of the project, addressing one of the major drawbacks of simple airdrops.
 *
 *     More background and motivation available at:
 *     https://github.com/ampleforth/RFCs/blob/master/RFCs/rfc-1.md
 */
contract TokenGeyser is IStaking, Ownable {
```

```
    using SafeMath for uint256;

    event Staked(address indexed user, uint256 amount, uint256 total, bytes data);
    event Unstaked(address indexed user, uint256 amount, uint256 total, bytes data);
    event TokensClaimed(address indexed user, uint256 amount);
    event TokensLocked(uint256 amount, uint256 durationSec, uint256 total);
    event TokensUnlocked(uint256 amount, uint256 total);

    TokenPool private _stakingPool;
    TokenPool private _unlockedPool;
    TokenPool private _lockedPool;

    //
    // Time-bonus params
    //
    uint256 public constant BONUS_DECIMALS = 2;
    uint256 public startBonus = 0;
    uint256 public bonusPeriodSec = 0;

    //
    // Global accounting state
    //
    uint256 public totalLockedShares = 0;
    uint256 public totalStakingShares = 0;
    uint256 private _totalStakingShareSeconds = 0;
    uint256 private _lastAccountingTimestampSec = now;
    uint256 private _maxUnlockSchedules = 0;
    uint256 private _initialSharesPerToken = 0;

    //
    // User accounting state
    //
    // Represents a single stake for a user. A user may have multiple.
    struct Stake {
        uint256 stakingShares;
        uint256 timestampSec;
    }

    // Caches aggregated values from the User->Stake[] map to save computation.
    // If lastAccountingTimestampSec is 0, there's no entry for that user.
    struct UserTotals {
        uint256 stakingShares;
        uint256 stakingShareSeconds;
        uint256 lastAccountingTimestampSec;
    }

    // Aggregated staking values per user
    mapping(address => UserTotals) private _userTotals;

    // The collection of stakes for each user. Ordered by timestamp, earliest to latest.
    mapping(address => Stake[]) private _userStakes;

    //
    // Locked/Unlocked Accounting state
    //
    struct UnlockSchedule {
        uint256 initialLockedShares;
        uint256 unlockedShares;
```

```
        uint256 lastUnlockTimestampSec;
        uint256 endAtSec;
        uint256 durationSec;
    }

    UnlockSchedule[] public unlockSchedules;

    /**
     * @param stakingToken The token users deposit as stake.
     * @param distributionToken The token users receive as they unstake.
     * @param maxUnlockSchedules Max number of unlock stages, to guard against hitting gas limit.
     * @param startBonus_ Starting time bonus, BONUS_DECIMALS fixed point.
     *                    e.g. 25% means user gets 25% of max distribution tokens.
     * @param bonusPeriodSec_ Length of time for bonus to increase linearly to max.
     * @param initialSharesPerToken Number of shares to mint per staking token on first stake.
     */
    /*@CTK "TokenGeyser_require"
      @tag assume_completion
      @post startBonus_ <= 10**BONUS_DECIMALS
      @post bonusPeriodSec_ != 0
      @post initialSharesPerToken > 0
     */
    /*@CTK "TokenGeyser"
      @tag assume_completion
      @pre startBonus_ <= 10**BONUS_DECIMALS
      @pre bonusPeriodSec_ != 0
      @pre initialSharesPerToken > 0
      @post __post._stakingPool.token == stakingToken
      @post __post._unlockedPool.token == distributionToken
      @post __post._lockedPool.token == distributionToken
      @post __post.startBonus == startBonus_
      @post __post.bonusPeriodSec == bonusPeriodSec_
      @post __post._maxUnlockSchedules == maxUnlockSchedules
      @post __post._initialSharesPerToken == initialSharesPerToken
     */
    constructor(UFragments stakingToken, UFragments distributionToken, uint256 maxUnlockSchedules,
                uint256 startBonus_, uint256 bonusPeriodSec_, uint256 initialSharesPerToken) public {
        // The start bonus must be some fraction of the max. (i.e. <= 100%)
        require(startBonus_ <= 10**BONUS_DECIMALS, 'TokenGeyser: start bonus too high');
        // If no period is desired, instead set startBonus = 100%
        // and bonusPeriod to a small value like 1sec.
        require(bonusPeriodSec_ != 0, 'TokenGeyser: bonus period is zero');
        require(initialSharesPerToken > 0);

        _stakingPool = new TokenPool(stakingToken);
        _unlockedPool = new TokenPool(distributionToken);
        _lockedPool = new TokenPool(distributionToken);
        startBonus = startBonus_;
        bonusPeriodSec = bonusPeriodSec_;
        _maxUnlockSchedules = maxUnlockSchedules;
        _initialSharesPerToken = initialSharesPerToken;
    }

    /**
```

```
     * @return The token users deposit as stake.
     */
    /*@CTK "getStakingToken"
      @post __return == _stakingPool.token
     */
    function getStakingToken() public view returns (UFragments) {
        return _stakingPool.token;
    }

    /**
     * @return The token users receive as they unstake.
     */
    /*@CTK "getDistributionToken"
      @tag assume_completion
      @pre _unlockedPool.token == _lockedPool.token
      @post __return == _unlockedPool.token
     */
    function getDistributionToken() public view returns (UFragments) {
        assert(_unlockedPool.token == _lockedPool.token);
        return _unlockedPool.token;
    }

    /**
     * @dev Transfers amount of deposit tokens from the user.
     * @param amount Number of deposit tokens to stake.
     * @param data Not used.
     */
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    function stake(uint256 amount, bytes calldata data) external {
        _stakeFor(msg.sender, msg.sender, amount);
    }

    /**
     * @dev Transfers amount of deposit tokens from the caller on behalf of user.
     * @param user User address who gains credit for this stake operation.
     * @param amount Number of deposit tokens to stake.
     * @param data Not used.
     */
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    function stakeFor(address user, uint256 amount, bytes calldata data) external {
        _stakeFor(msg.sender, user, amount);
    }

    /**
     * @dev Private implementation of staking methods.
     * @param staker User address who deposits tokens to stake.
     * @param beneficiary User address who gains credit for this stake operation.
     * @param amount Number of deposit tokens to stake.
     */
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "_stakeFor require"
      @tag assume_completion
      @let uint totalStaked = _stakingPool.token._gonBalances[_stakingPool] / _stakingPool.
          token._gonsPerFragment
      @post amount > 0
```

```
     @post beneficiary != address(0)
     @post totalStakingShares > 0 -> totalStaked > 0
    */
   /*@CTK "_stakeFor case 1"
     @tag assume_completion
     @pre totalStakingShares > 0
     @let uint totalStaked = _stakingPool.token._gonBalances[_stakingPool] / _stakingPool.
         token._gonsPerFragment
     @let uint mintedStakingShares = totalStakingShares * amount / totalStaked
     @post __post._userTotals[beneficiary].stakingShares == _userTotals[beneficiary].
         stakingShares + mintedStakingShares
     @post __post.totalStakingShares == totalStakingShares + mintedStakingShares
    */
   /*@CTK "_stakeFor case 2"
     @tag assume_completion
     @pre totalStakingShares == 0
     @let uint mintedStakingShares = amount * _initialSharesPerToken
     @post __post._userTotals[beneficiary].stakingShares == _userTotals[beneficiary].
         stakingShares + mintedStakingShares
     @post __post.totalStakingShares == totalStakingShares + mintedStakingShares
    */
   function _stakeFor(address staker, address beneficiary, uint256 amount) private {
       require(amount > 0, 'TokenGeyser: stake amount is zero');
       require(beneficiary != address(0), 'TokenGeyser: beneficiary is zero address');
       require(totalStakingShares == 0 || totalStaked() > 0,
               'TokenGeyser: Invalid state. Staking shares exist, but no staking tokens do');

       uint256 mintedStakingShares = (totalStakingShares > 0)
           ? totalStakingShares.mul(amount).div(totalStaked())
           : amount.mul(_initialSharesPerToken);
       require(mintedStakingShares > 0, 'TokenGeyser: Stake amount is too small');

       updateAccounting();

       // 1. User Accounting
       UserTotals storage totals = _userTotals[beneficiary];
       totals.stakingShares = totals.stakingShares.add(mintedStakingShares);
       totals.lastAccountingTimestampSec = now;

       Stake memory newStake = Stake(mintedStakingShares, now);
       _userStakes[beneficiary].push(newStake);

       // 2. Global Accounting
       totalStakingShares = totalStakingShares.add(mintedStakingShares);
       // Already set in updateAccounting()
       // _lastAccountingTimestampSec = now;

       // interactions
       require(_stakingPool.token.transferFrom(staker, address(_stakingPool), amount),
           'TokenGeyser: transfer into staking pool failed');

       emit Staked(beneficiary, amount, totalStakedFor(beneficiary), "");
   }

   /**
    * @dev Unstakes a certain amount of previously deposited tokens. User also receives
        their
    * alotted number of distribution tokens.
```

```
     * @param amount Number of deposit tokens to unstake / withdraw.
     * @param data Not used.
     */
    //CTK NO_OVERFLOW
    //CTK NO_BUF_OVERFLOW
    //CKT NO_ASF
    function unstake(uint256 amount, bytes calldata data) external {
        _unstake(amount);
    }

    /**
     * @param amount Number of deposit tokens to unstake / withdraw.
     * @return The total number of distribution tokens that would be rewarded.
     */
    //CTK NO_OVERFLOW
    //CTK NO_BUF_OVERFLOW
    //CKT NO_ASF
    function unstakeQuery(uint256 amount) public returns (uint256) {
        return _unstake(amount);
    }

    /**
     * @dev Unstakes a certain amount of previously deposited tokens. User also receives their
     * alotted number of distribution tokens.
     * @param amount Number of deposit tokens to unstake / withdraw.
     * @return The total number of distribution tokens rewarded.
     */
    //CTK NO_OVERFLOW
    //CTK NO_BUF_OVERFLOW
    //CKT NO_ASF
    function _unstake(uint256 amount) private returns (uint256) {
        updateAccounting();

        // checks
        require(amount > 0, 'TokenGeyser: unstake amount is zero');
        require(totalStakedFor(msg.sender) >= amount,
            'TokenGeyser: unstake amount is greater than total user stakes');
        uint256 stakingSharesToBurn = totalStakingShares.mul(amount).div(totalStaked());
        require(stakingSharesToBurn > 0, 'TokenGeyser: Unable to unstake amount this small');

        // 1. User Accounting
        UserTotals storage totals = _userTotals[msg.sender];
        Stake[] storage accountStakes = _userStakes[msg.sender];

        // Redeem from most recent stake and go backwards in time.
        uint256 stakingShareSecondsToBurn = 0;
        uint256 sharesLeftToBurn = stakingSharesToBurn;
        uint256 rewardAmount = 0;
        /*@*CTK "_unstake while"
          @inv sharesLeftToBurn >= 0
         */
        while (sharesLeftToBurn > 0) {
            Stake storage lastStake = accountStakes[accountStakes.length - 1];
            uint256 stakeTimeSec = now.sub(lastStake.timestampSec);
            uint256 newStakingShareSecondsToBurn = 0;
            if (lastStake.stakingShares <= sharesLeftToBurn) {
```

```
            // fully redeem a past stake
            newStakingShareSecondsToBurn = lastStake.stakingShares.mul(stakeTimeSec);
            rewardAmount = computeNewReward(rewardAmount, newStakingShareSecondsToBurn, stakeTimeSec);
            stakingShareSecondsToBurn = stakingShareSecondsToBurn.add(newStakingShareSecondsToBurn);
            sharesLeftToBurn = sharesLeftToBurn.sub(lastStake.stakingShares);
            accountStakes.length--;
        } else {
            // partially redeem a past stake
            newStakingShareSecondsToBurn = sharesLeftToBurn.mul(stakeTimeSec);
            rewardAmount = computeNewReward(rewardAmount, newStakingShareSecondsToBurn, stakeTimeSec);
            stakingShareSecondsToBurn = stakingShareSecondsToBurn.add(newStakingShareSecondsToBurn);
            lastStake.stakingShares = lastStake.stakingShares.sub(sharesLeftToBurn);
            sharesLeftToBurn = 0;
        }
    }
    totals.stakingShareSeconds = totals.stakingShareSeconds.sub(stakingShareSecondsToBurn);
    totals.stakingShares = totals.stakingShares.sub(stakingSharesToBurn);
    // Already set in updateAccounting
    // totals.lastAccountingTimestampSec = now;

    // 2. Global Accounting
    _totalStakingShareSeconds = _totalStakingShareSeconds.sub(stakingShareSecondsToBurn);

    totalStakingShares = totalStakingShares.sub(stakingSharesToBurn);
    // Already set in updateAccounting
    // _lastAccountingTimestampSec = now;

    // interactions
    require(_stakingPool.transfer(msg.sender, amount),
        'TokenGeyser: transfer out of staking pool failed');
    require(_unlockedPool.transfer(msg.sender, rewardAmount),
        'TokenGeyser: transfer out of unlocked pool failed');

    emit Unstaked(msg.sender, amount, totalStakedFor(msg.sender), "");
    emit TokensClaimed(msg.sender, rewardAmount);

    require(totalStakingShares == 0 || totalStaked() > 0,
            "TokenGeyser: Error unstaking. Staking shares exist, but no staking tokens do");
    return rewardAmount;
}

/**
 * @dev Applies an additional time-bonus to a distribution amount. This is necessary to
 *     encourage long-term deposits instead of constant unstake/restakes.
 *     The bonus-multiplier is the result of a linear function that starts at startBonus and
 *     ends at 100% over bonusPeriodSec, then stays at 100% thereafter.
 * @param currentRewardTokens The current number of distribution tokens already alotted for this
 *                        unstake op. Any bonuses are already applied.
 * @param stakingShareSeconds The stakingShare-seconds that are being burned for new
 *                        distribution tokens.
```

```
     * @param stakeTimeSec Length of time for which the tokens were staked. Needed to calculate
     *                  the time-bonus.
     * @return Updated amount of distribution tokens to award, with any bonus included on the
     *        newly added tokens.
     */
    /*@CTK "computeNewReward case 0"
      @tag assume_completion
      @pre stakeTimeSec >= bonusPeriodSec
      @let uint totalUnlocked = _unlockedPool.token._gonBalances[_unlockedPool] / _unlockedPool.token._gonsPerFragment
      @let uint newRewardTokens = totalUnlocked * stakingShareSeconds / _totalStakingShareSeconds
      @post __return == currentRewardTokens + newRewardTokens
     */
    /*@CTK "computeNewReward case 1"
      @tag assume_completion
      @pre stakeTimeSec < bonusPeriodSec
      @let uint totalUnlocked = _unlockedPool.token._gonBalances[_unlockedPool] / _unlockedPool.token._gonsPerFragment
      @let uint newRewardTokens = totalUnlocked * stakingShareSeconds / _totalStakingShareSeconds
      @let uint oneHundredPct = 10**BONUS_DECIMALS
      @let uint bonusedReward = (startBonus + (oneHundredPct - startBonus) * stakeTimeSec / bonusPeriodSec) * newRewardTokens / oneHundredPct
      @post __return == currentRewardTokens + bonusedReward
     */
    function computeNewReward(uint256 currentRewardTokens,
                              uint256 stakingShareSeconds,
                              uint256 stakeTimeSec) private view returns (uint256) {

        uint256 newRewardTokens =
            totalUnlocked()
            .mul(stakingShareSeconds)
            .div(_totalStakingShareSeconds);

        if (stakeTimeSec >= bonusPeriodSec) {
            return currentRewardTokens.add(newRewardTokens);
        }

        uint256 oneHundredPct = 10**BONUS_DECIMALS;
        uint256 bonusedReward =
            startBonus
            .add(oneHundredPct.sub(startBonus).mul(stakeTimeSec).div(bonusPeriodSec))
            .mul(newRewardTokens)
            .div(oneHundredPct);
        return currentRewardTokens.add(bonusedReward);
    }

    /**
     * @param addr The user to look up staking information for.
     * @return The number of staking tokens deposited for addr.
     */
    /*@CTK "totalStakedFor"
      @tag assume_completion
      @let uint totalStaked = _stakingPool.token._gonBalances[_stakingPool] / _stakingPool.token._gonsPerFragment
```

```
     @post totalStakingShares == 0 -> __return == 0
     @post totalStakingShares > 0 -> __return == totalStaked * _userTotals[addr].stakingShares / totalStakingShares
    */
   function totalStakedFor(address addr) public view returns (uint256) {
       return totalStakingShares > 0 ?
           totalStaked().mul(_userTotals[addr].stakingShares).div(totalStakingShares) : 0;
   }

   /**
    * @return The total number of deposit tokens staked globally, by all users.
    */
   /*@CTK "totalStaked"
     @tag assume_completion
     @post __return == _stakingPool.token._gonBalances[_stakingPool] / _stakingPool.token._gonsPerFragment
    */
   function totalStaked() public view returns (uint256) {
       return _stakingPool.balance();
   }

   /**
    * @dev Note that this application has a staking token as well as a distribution token, which
    * may be different. This function is required by EIP-900.
    * @return The deposit token used for staking.
    */
   /*@CTK "token"
     @post __return == _stakingPool.token
    */
   function token() external view returns (address) {
       return address(getStakingToken());
   }

   /**
    * @dev A globally callable function to update the accounting state of the system.
    *      Global state and state for the caller are updated.
    * @return [0] balance of the locked pool
    * @return [1] balance of the unlocked pool
    * @return [2] caller's staking share seconds
    * @return [3] global staking share seconds
    * @return [4] Rewards caller has accumulated, optimistically assumes max time-bonus.
    * @return [5] block timestamp
    */
   //@CTK NO_OVERFLOW
   //@CTK NO_BUF_OVERFLOW
   //@CTK NO_ASF
   /*@*CTK "updateAccounting"
     @tag assume_completion
     @post __post._totalStakingShareSeconds == _totalStakingShareSeconds + (now - _lastAccountingTimestampSec) * totalStakingShares
     @post __post._lastAccountingTimestampSec == now
     @post __post._userTotals[msg.sender].stakingShareSeconds == _userTotals[msg.sender].stakingShareSeconds + (now - _userTotals[msg.sender].lastAccountingTimestampSec) * _userTotals[msg.sender].stakingShares
     @post __post._userTotals[msg.sender].lastAccountingTimestampSec == now
    */
   function updateAccounting() public returns (
```

```
        uint256, uint256, uint256, uint256, uint256, uint256) {

        unlockTokens();

        // Global accounting
        uint256 newStakingShareSeconds =
            now
            .sub(_lastAccountingTimestampSec)
            .mul(totalStakingShares);
        _totalStakingShareSeconds = _totalStakingShareSeconds.add(newStakingShareSeconds);
        _lastAccountingTimestampSec = now;

        // User Accounting
        UserTotals storage totals = _userTotals[msg.sender];
        uint256 newUserStakingShareSeconds =
            now
            .sub(totals.lastAccountingTimestampSec)
            .mul(totals.stakingShares);
        totals.stakingShareSeconds =
            totals.stakingShareSeconds
            .add(newUserStakingShareSeconds);
        totals.lastAccountingTimestampSec = now;

        uint256 totalUserRewards = (_totalStakingShareSeconds > 0)
            ? totalUnlocked().mul(totals.stakingShareSeconds).div(_totalStakingShareSeconds)
            : 0;

        return (
            totalLocked(),
            totalUnlocked(),
            totals.stakingShareSeconds,
            _totalStakingShareSeconds,
            totalUserRewards,
            now
        );
    }

    /**
     * @return Total number of locked distribution tokens.
     */
    /*@CTK "totalLocked"
      @tag assume_completion
      @post __return == _lockedPool.token._gonBalances[_lockedPool] / _lockedPool.token.
          _gonsPerFragment
     */
    function totalLocked() public view returns (uint256) {
        return _lockedPool.balance();
    }

    /**
     * @return Total number of unlocked distribution tokens.
     */
    /*@CTK "totalUnlocked"
      @tag assume_completion
      @post __return == _unlockedPool.token._gonBalances[_unlockedPool] / _unlockedPool.
          token._gonsPerFragment
     */
    function totalUnlocked() public view returns (uint256) {
```

```
        return _unlockedPool.balance();
    }

    /**
     * @return Number of unlock schedules.
     */
    /*@CTK "unlockScheduleCount"
      @post __return == unlockSchedules.length
     */
    function unlockScheduleCount() public view returns (uint256) {
        return unlockSchedules.length;
    }

    /**
     * @dev This funcion allows the contract owner to add more locked distribution tokens, along
     *     with the associated "unlock schedule". These locked tokens immediately begin unlocking
     *     linearly over the duraction of durationSec timeframe.
     * @param amount Number of distribution tokens to lock. These are transferred from the caller.
     * @param durationSec Length of time to linear unlock the tokens.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    function lockTokens(uint256 amount, uint256 durationSec) external onlyOwner {
        require(unlockSchedules.length < _maxUnlockSchedules,
            'TokenGeyser: reached maximum unlock schedules');

        // Update lockedTokens amount before using it in computations after.
        updateAccounting();

        uint256 lockedTokens = totalLocked();
        uint256 mintedLockedShares = (lockedTokens > 0)
            ? totalLockedShares.mul(amount).div(lockedTokens)
            : amount.mul(_initialSharesPerToken);

        UnlockSchedule memory schedule;
        schedule.initialLockedShares = mintedLockedShares;
        schedule.lastUnlockTimestampSec = now;
        schedule.endAtSec = now.add(durationSec);
        schedule.durationSec = durationSec;
        unlockSchedules.push(schedule);

        totalLockedShares = totalLockedShares.add(mintedLockedShares);

        require(_lockedPool.token.transferFrom(msg.sender, address(_lockedPool), amount),
            'TokenGeyser: transfer into locked pool failed');
        emit TokensLocked(amount, durationSec, totalLocked());
    }

    /**
     * @dev Moves distribution tokens from the locked pool to the unlocked pool, according to the
     *     previously defined unlock schedules. Publicly callable.
     * @return Number of newly unlocked distribution tokens.
     */
```

```
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "unlockTokens case 1"
      @tag assume_completion
      @pre totalLockedShares == 0
      @pre _lockedPool.token._gonBalances[_lockedPool] == 0
      @post __return == 0
     */
    /*@CTK "unlockTokens case 2"
      @tag assume_completion
      @pre totalLockedShares == 0
      @pre _lockedPool.token._gonBalances[_lockedPool] > 0
      @pre _lockedPool != _unlockedPool
      @let uint lockedTokens = _lockedPool.token._gonBalances[_lockedPool] / _lockedPool.
          token._gonsPerFragment
      //@post __post._unlockedPool.token._gonBalances[_unlockedPool] == _unlockedPool.token.
          _gonBalances[_unlockedPool] + lockedTokens * _lockedPool.token._gonsPerFragment
      //@post __post._lockedPool.token._gonBalances[_lockedPool] == _lockedPool.token.
          _gonBalances[_lockedPool] - lockedTokens * _lockedPool.token._gonsPerFragment
      @post __return == lockedTokens
     */
    function unlockTokens() public returns (uint256) {
        uint256 unlockedTokens = 0;
        uint256 lockedTokens = totalLocked();

        if (totalLockedShares == 0) {
            unlockedTokens = lockedTokens;
        } else {
            uint256 unlockedShares = 0;
            /*@CTK "unlockTokens for"
              @inv s <= unlockSchedules.length
             */
            for (uint256 s = 0; s < unlockSchedules.length; s++) {
                unlockedShares += unlockScheduleShares(s);
            }
            unlockedTokens = unlockedShares.mul(lockedTokens).div(totalLockedShares);
            totalLockedShares = totalLockedShares.sub(unlockedShares);
        }

        if (unlockedTokens > 0) {
            require(_lockedPool.transfer(address(_unlockedPool), unlockedTokens),
                'TokenGeyser: transfer out of locked pool failed');
            emit TokensUnlocked(unlockedTokens, totalLocked());
        }

        return unlockedTokens;
    }

    /**
     * @dev Returns the number of unlockable shares from a given schedule. The returned
         value
     *     depends on the time since the last unlock. This function updates schedule
         accounting,
     *     but does not actually transfer any tokens.
     * @param s Index of the unlock schedule.
     * @return The number of unlocked shares.
     */
```

```
    //@CTK FAIL NO_OVERFLOW
    //@CTK FAIL NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "unlockScheduleShares case 1"
      @pre unlockSchedules[s].unlockedShares >= unlockSchedules[s].initialLockedShares
      @post __return == 0
     */
    /*@CTK "unlockScheduleShares case 2"
      @tag assume_completion
      @pre unlockSchedules[s].unlockedShares < unlockSchedules[s].initialLockedShares
      @post now >= unlockSchedules[s].endAtSec -> __post.unlockSchedules[s].
          lastUnlockTimestampSec == unlockSchedules[s].endAtSec
      @post now >= unlockSchedules[s].endAtSec -> __post.unlockSchedules[s].unlockedShares
          == unlockSchedules[s].unlockedShares + unlockSchedules[s].initialLockedShares -
          unlockSchedules[s].unlockedShares
      @post now >= unlockSchedules[s].endAtSec -> __return == unlockSchedules[s].
          initialLockedShares - unlockSchedules[s].unlockedShares
      @post now < unlockSchedules[s].endAtSec -> __post.unlockSchedules[s].
          lastUnlockTimestampSec == now
      @post now < unlockSchedules[s].endAtSec -> __post.unlockSchedules[s].unlockedShares ==
           unlockSchedules[s].unlockedShares + (now - unlockSchedules[s].
          lastUnlockTimestampSec) * unlockSchedules[s].initialLockedShares / unlockSchedules
          [s].durationSec
      @post now < unlockSchedules[s].endAtSec -> __return == (now - unlockSchedules[s].
          lastUnlockTimestampSec) * unlockSchedules[s].initialLockedShares / unlockSchedules
          [s].durationSec
     */
    function unlockScheduleShares(uint256 s) private returns (uint256) {
        UnlockSchedule storage schedule = unlockSchedules[s];

        if(schedule.unlockedShares >= schedule.initialLockedShares){
            return 0;
        }

        uint256 sharesToUnlock = 0;
        // Special case to handle any leftover dust from integer division
        if(now >= schedule.endAtSec){
            sharesToUnlock = (schedule.initialLockedShares - schedule.unlockedShares);
            schedule.lastUnlockTimestampSec = schedule.endAtSec;
        } else {
            sharesToUnlock = now.sub(schedule.lastUnlockTimestampSec)
                .mul(schedule.initialLockedShares)
                .div(schedule.durationSec);
            schedule.lastUnlockTimestampSec = now;
        }

        schedule.unlockedShares += sharesToUnlock;
        return sharesToUnlock;
    }
}
```