CERTIK

Audit Report

Produced by CertiK

for THE SANDBOX

Dec 2nd, 2019

CertiK Audit Report For The Sandbox

Request Date: 2019-11-08
Revision Date: 2019-12-02
Platform Name: Ethereum

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 The Sandbox(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 The Sandbox to discover issues and vulnerabilities in the source code of their SAND smart contract. 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 practice 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 3 buckets based on overall risk levels:

Critical

The code implementation does not match the specification, or it could result in the loss of funds for contract owner or users.

Medium

The code implementation does not match the specification under certain conditions, or it could affect the security standard by lost of access control.

Low

The code implementation does not follow best practices, or use suboptimal design patterns, which may lead to security vulnerabilities further down the line.

Testing Summary

Type of Issues

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

Title	Description	Issues	SWC ID
Integer Overflow and Underflow	An overflow/underflow happens when an arithmetic operation reaches the maximum or minimum size of a type.	0	SWC-101
Function incorrectness	Function implementation does not meet the specification, leading to intentional or unintentional vulnerabilities.	0	
Buffer Overflow	An attacker is able to 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.	0	SWC-116
Insecure Compiler Version	Using an 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.	0	SWC-102 SWC-103
Insecure Randomness	Block attributes are insecure to generate random numbers, 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
Delegatecall to Untrusted Callee	Calling into untrusted contracts is very dangerous, 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. A malicious user is able to 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 as best practice.	0	SWC-111
Unused variables	Unused variables reduce code quality	0	

Vulnerability Details

Critical

No issue found.

Medium

No issue found.

Low

- Recommend using `SafeMath` throughtout the contracts.
- Recommend changing `.call` to `.transfer` or explicit calls.

Manual Review Notes

Source Code SHA-256 Checksum

- **Admin.sol**
  `f336e6bd77e29368a3afe4ffecdc9eafe0b2854f2c303d47405a45a85bfcfb6e`
- **SuperOperators.sol**
  `307c0411cfc020057e1d38d9ff5a715b088bd074a8351f1cf572fe2b386dfe12`
- **ERC20BaseToken.sol**
  `58dc46ab7946c54517517f0f372098edcabe043f81707a523cba451b0203492d`
- **ERC20BasicApproveExtension.sol**
  `38239773e1f444dc4119fa34c195c0459831556747e15f769b870a743f7ed453`
- **ERC20ExecuteExtension.sol**
  `9ef188608f08e7f89d816e9a8c198ea1996baad084b5bb04835a3cb31a35d051`
- **Sand.sol**
  `0fd0e82bed1f1f4ff5ecdf88f2c32230ff9bb778a63909d18d46bd1e2beff0eb`

Summary

CertiK was chosen by Sandbox to audit the design and implementation of its soon to be released `SAND` smart contracts. To ensure comprehensive protection,the source code has been 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 the best practices in the space.

Overall we found the 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, keep improving the codebase, and more test coverage and sandbox deployments before the mainnet release.

Recommendations

Items in this section are low impact to the overall aspects of the smart contracts, thus will let client to decide whether to have those reflected in the final deployed version of source codes.

ERC20BaseToken.sol

- INFO `decimals()`: Recommend marking this function with `pure`.
- INFO `decimals()`: `18` is hard coded. Recommend setting a constant for it.

- MINOR `approveFor(owner, spender, amount)`, `addAllowanceIfNeeded(owner, spender, amountNeeded)`: Recommend using `increaseAllowance` instead of direct setting of `_allowances` to prevent from front-running attack.
- INFO Recommend using `SafeMath` throughout the contract.

ERC20ExecuteExtension.sol

- INFO `_charge`: Recommend using `safeMath` or adding check for `gasCharge += baseGasCharge`.
- INFO `changeExecutionAdmin`: Recommend using pull-over-push pattern, e.g. using an `_proposedExecutionAdmin` and two functions `proposeExecutionAdmin`, `acceptExecutionAdmin`.
- INFO Recommend wrapping `require(_executionOperators[msg.sender])` and `require(msg.sender == _executionAdmin)` checks into a modifiers.
- MINOR `executeWithSpecificGas(to, gasLimit, data)`, `approveAndExecuteWithSpecificGas(from, to, amount, gasLimit, data)`: Missing `require(success)`. Otherwise recommend returning the successfulness of the low-level call.
- MINOR `executeWithSpecificGas(to, gasLimit, data)`, `approveAndExecuteWithSpecificGas(from, to, amount, gasLimit, data)`: Recommend changing `assert` to `require` with error messages provided.
- MINOR `_charge(from, gasLimit, tokenGasPrice, initialGas, baseGasCharge, tokenReceiver)`: Recommend adding overflow check for `gasCharge` for consistency.
- INFO `_charge`, `transferAndChargeForGas`: Recommend adding range check for `gasLimit`, `tokenGasPrice`, `initialGas`, `baseGasCharge`, `tokenReceiver` at the entry of the function.

ERC20BasicApproveExtension

- MINOR `approveAndCall(target, amount, data)`, `paidCall(target, amount, data)`: Recommend avoiding the use of `.call` and use explicitly call or `.transfer` instead. If `.call` must be used, recommend adding `.gas` limitation on the call.
- MINOR `approveAndCall(target, amount, data)`: The function can be modified to reuse `approveAndExecuteWithSpecificGas`.

Sand.sol

- INFO `constructor(sandAdmin, executionAdmin, beneficiary)`: Recommend setting `_admin` and `executionAdmin` to `msg.sender` and switching the administration later.
- INFO `name()` and `symbol()`: Recommend marking these functions as `pure`.

Static Analysis Results

INSECURE_COMPILER_VERSION

Line 1 in File ERC20ExecuteExtension.sol

```
1 pragma solidity 0.5.9;
```

Version to compile has the following bug: 0.5.9: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement

INSECURE_COMPILER_VERSION

Line 1 in File Admin.sol

```
1 pragma solidity ^0.5.2;
```

Only these compiler versions are safe to compile your code: 0.5.10

INSECURE_COMPILER_VERSION

Line 1 in File SuperOperators.sol

```
1 pragma solidity ^0.5.2;
```

Only these compiler versions are safe to compile your code: 0.5.10

INSECURE_COMPILER_VERSION

Line 1 in File Sand.sol

```
1 pragma solidity 0.5.9;
```

Version to compile has the following bug: 0.5.9: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement

INSECURE_COMPILER_VERSION

Line 1 in File ERC20BaseToken.sol

```
1 pragma solidity 0.5.9;
```

Version to compile has the following bug: 0.5.9: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement

INSECURE_COMPILER_VERSION

Line 1 in File ERC20BasicApproveExtension.sol

```
1 pragma solidity 0.5.9;
```

Version to compile has the following bug: 0.5.9: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement

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

If method completes, integer overflow would not happen.

02, Dec 2019
5.13 ms

Line 13 in File ERC20ExecuteExtension.sol

```
13    //@CTK NO_OVERFLOW
```

Line 19-21 in File ERC20ExecuteExtension.sol

```
19    function getExecutionAdmin() external view returns (address) {
20        return _executionAdmin;
21    }
```

The code meets the specification.

Formal Verification Request 2

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

02, Dec 2019
0.33 ms

Line 14 in File ERC20ExecuteExtension.sol

```
14    //@CTK NO_BUF_OVERFLOW
```

Line 19-21 in File ERC20ExecuteExtension.sol

```
19    function getExecutionAdmin() external view returns (address) {
20        return _executionAdmin;
21    }
```

The code meets the specification.

Formal Verification Request 3

Method will not encounter an assertion failure.

02, Dec 2019
0.31 ms

Line 15 in File ERC20ExecuteExtension.sol

```
15    //@CTK NO_ASF
```

Line 19-21 in File ERC20ExecuteExtension.sol

```
19    function getExecutionAdmin() external view returns (address) {
20        return _executionAdmin;
21    }
```

The code meets the specification.

Formal Verification Request 4

getExecutionAdmin

02, Dec 2019
0.33 ms

Line 16-18 in File ERC20ExecuteExtension.sol

```
16    /*@CTK getExecutionAdmin
17      @post __return == _executionAdmin
18     */
```

Line 19-21 in File ERC20ExecuteExtension.sol

```
19    function getExecutionAdmin() external view returns (address) {
20        return _executionAdmin;
21    }
```

The code meets the specification.

Formal Verification Request 5

If method completes, integer overflow would not happen.

02, Dec 2019
16.79 ms

Line 25 in File ERC20ExecuteExtension.sol

```
25    //@CTK NO_OVERFLOW
```

Line 37-41 in File ERC20ExecuteExtension.sol

```
37    function changeExecutionAdmin(address newAdmin) external {
38        require(msg.sender == _executionAdmin, "only executionAdmin can change
              executionAdmin");
39        emit ExecutionAdminAdminChanged(_executionAdmin, newAdmin);
40        _executionAdmin = newAdmin;
41    }
```

The code meets the specification.

Formal Verification Request 6

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

02, Dec 2019
0.49 ms

Line 26 in File ERC20ExecuteExtension.sol

```
26    //@CTK NO_BUF_OVERFLOW
```

Line 37-41 in File ERC20ExecuteExtension.sol

```
37    function changeExecutionAdmin(address newAdmin) external {
38        require(msg.sender == _executionAdmin, "only executionAdmin can change
              executionAdmin");
39        emit ExecutionAdminAdminChanged(_executionAdmin, newAdmin);
40        _executionAdmin = newAdmin;
41    }
```

The code meets the specification.

Formal Verification Request 7

Method will not encounter an assertion failure.

02, Dec 2019
0.48 ms

Line 27 in File ERC20ExecuteExtension.sol

```
27    //@CTK NO_ASF
```

Line 37-41 in File ERC20ExecuteExtension.sol

```
37    function changeExecutionAdmin(address newAdmin) external {
38        require(msg.sender == _executionAdmin, "only executionAdmin can change
              executionAdmin");
39        emit ExecutionAdminAdminChanged(_executionAdmin, newAdmin);
40        _executionAdmin = newAdmin;
41    }
```

The code meets the specification.

Formal Verification Request 8

changeExecutionAdmin_require

02, Dec 2019
0.85 ms

Line 28-31 in File ERC20ExecuteExtension.sol

```
28    /*@CTK changeExecutionAdmin_require
29      @tag assume_completion
30      @post msg.sender == _executionAdmin
31     */
```

Line 37-41 in File ERC20ExecuteExtension.sol

```
37    function changeExecutionAdmin(address newAdmin) external {
38        require(msg.sender == _executionAdmin, "only executionAdmin can change
              executionAdmin");
39        emit ExecutionAdminAdminChanged(_executionAdmin, newAdmin);
40        _executionAdmin = newAdmin;
41    }
```

The code meets the specification.

Formal Verification Request 9

changeExecutionAdmin_change

02, Dec 2019
1.29 ms

Line 32-36 in File ERC20ExecuteExtension.sol

```
32  /*@CTK changeExecutionAdmin_change
33    @tag assume_completion
34    @pre msg.sender == _executionAdmin
35    @post __post._executionAdmin == newAdmin
36   */
```

Line 37-41 in File ERC20ExecuteExtension.sol

```
37  function changeExecutionAdmin(address newAdmin) external {
38      require(msg.sender == _executionAdmin, "only executionAdmin can change
            executionAdmin");
39      emit ExecutionAdminAdminChanged(_executionAdmin, newAdmin);
40      _executionAdmin = newAdmin;
41  }
```

The code meets the specification.

Formal Verification Request 10

If method completes, integer overflow would not happen.

02, Dec 2019
18.51 ms

Line 49 in File ERC20ExecuteExtension.sol

```
49  //@CTK NO_OVERFLOW
```

Line 61-68 in File ERC20ExecuteExtension.sol

```
61  function setExecutionOperator(address executionOperator, bool enabled) external {
62      require(
63          msg.sender == _executionAdmin,
64          "only execution admin is allowed to add execution operators"
65      );
66      _executionOperators[executionOperator] = enabled;
67      emit ExecutionOperator(executionOperator, enabled);
68  }
```

The code meets the specification.

Formal Verification Request 11

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

02, Dec 2019
0.41 ms

Line 50 in File ERC20ExecuteExtension.sol

```
50    //@CTK NO_BUF_OVERFLOW
```

Line 61-68 in File ERC20ExecuteExtension.sol

```
61    function setExecutionOperator(address executionOperator, bool enabled) external {
62        require(
63            msg.sender == _executionAdmin,
64            "only execution admin is allowed to add execution operators"
65        );
66        _executionOperators[executionOperator] = enabled;
67        emit ExecutionOperator(executionOperator, enabled);
68    }
```

✅ The code meets the specification.

Formal Verification Request 12

Method will not encounter an assertion failure.

📅 02, Dec 2019
⏱ 0.38 ms

Line 51 in File ERC20ExecuteExtension.sol

```
51    //@CTK NO_ASF
```

Line 61-68 in File ERC20ExecuteExtension.sol

```
61    function setExecutionOperator(address executionOperator, bool enabled) external {
62        require(
63            msg.sender == _executionAdmin,
64            "only execution admin is allowed to add execution operators"
65        );
66        _executionOperators[executionOperator] = enabled;
67        emit ExecutionOperator(executionOperator, enabled);
68    }
```

✅ The code meets the specification.

Formal Verification Request 13

setExecutionOperator_require

📅 02, Dec 2019
⏱ 0.93 ms

Line 52-55 in File ERC20ExecuteExtension.sol

```
52    /*@CTK setExecutionOperator_require
53      @tag assume_completion
54      @post msg.sender == _executionAdmin
55     */
```

Line 61-68 in File ERC20ExecuteExtension.sol

```
61    function setExecutionOperator(address executionOperator, bool enabled) external {
62        require(
63            msg.sender == _executionAdmin,
64            "only execution admin is allowed to add execution operators"
65        );
66        _executionOperators[executionOperator] = enabled;
67        emit ExecutionOperator(executionOperator, enabled);
68    }
```

The code meets the specification.

Formal Verification Request 14

setExecutionOperator_change

02, Dec 2019
1.97 ms

Line 56-60 in File ERC20ExecuteExtension.sol

```
56    /*@CTK setExecutionOperator_change
57      @tag assume_completion
58      @pre msg.sender == _executionAdmin
59      @post __post._executionOperators[executionOperator] == enabled
60     */
```

Line 61-68 in File ERC20ExecuteExtension.sol

```
61    function setExecutionOperator(address executionOperator, bool enabled) external {
62        require(
63            msg.sender == _executionAdmin,
64            "only execution admin is allowed to add execution operators"
65        );
66        _executionOperators[executionOperator] = enabled;
67        emit ExecutionOperator(executionOperator, enabled);
68    }
```

The code meets the specification.

Formal Verification Request 15

If method completes, integer overflow would not happen.

02, Dec 2019
5.84 ms

Line 73 in File ERC20ExecuteExtension.sol

```
73    //@CTK NO_OVERFLOW
```

Line 79-81 in File ERC20ExecuteExtension.sol

```
79    function isExecutionOperator(address who) public view returns (bool) {
80        return _executionOperators[who];
81    }
```

The code meets the specification.

Formal Verification Request 16

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

02, Dec 2019
0.45 ms

Line 74 in File ERC20ExecuteExtension.sol

```
74    //@CTK NO_BUF_OVERFLOW
```

Line 79-81 in File ERC20ExecuteExtension.sol

```
79    function isExecutionOperator(address who) public view returns (bool) {
80        return _executionOperators[who];
81    }
```

The code meets the specification.

Formal Verification Request 17

Method will not encounter an assertion failure.

02, Dec 2019
0.38 ms

Line 75 in File ERC20ExecuteExtension.sol

```
75    //@CTK NO_ASF
```

Line 79-81 in File ERC20ExecuteExtension.sol

```
79    function isExecutionOperator(address who) public view returns (bool) {
80        return _executionOperators[who];
81    }
```

The code meets the specification.

Formal Verification Request 18

isExecutionOperator

02, Dec 2019
0.55 ms

Line 76-78 in File ERC20ExecuteExtension.sol

```
76    /*@CTK isExecutionOperator
77      @post __return == _executionOperators[who]
78     */
```

Line 79-81 in File ERC20ExecuteExtension.sol

```
79    function isExecutionOperator(address who) public view returns (bool) {
80        return _executionOperators[who];
81    }
```

The code meets the specification.

Formal Verification Request 19

executeWithSpecificGas_require

02, Dec 2019
20.5 ms

Line 92-95 in File ERC20ExecuteExtension.sol

```
92  /*@CTK executeWithSpecificGas_require
93    @tag assume_completion
94    @post _executionOperators[msg.sender] == true
95   */
```

Line 96-102 in File ERC20ExecuteExtension.sol

```
96  function executeWithSpecificGas(address to, uint256 gasLimit, bytes calldata data)
        external returns (bool success, bytes memory returnData) {
97      require(_executionOperators[msg.sender], "only execution operators allowed to
            execute on SAND behalf");
98      (success, returnData) = to.call.gas(gasLimit)(data);
99      assert(gasleft() > gasLimit / 63); // not enough gas provided, assert to throw
            all gas // TODO use EIP-1930
100 }
```

The code meets the specification.

Formal Verification Request 20

If method completes, integer overflow would not happen.

02, Dec 2019
19.28 ms

Line 112 in File ERC20ExecuteExtension.sol

```
112 //@CTK NO_OVERFLOW
```

Line 119-130 in File ERC20ExecuteExtension.sol

```
119 function approveAndExecuteWithSpecificGas(
120     address from,
121     address to,
122     uint256 amount,
123     uint256 gasLimit,
124     bytes calldata data
125 ) external returns (bool success, bytes memory returnData) {
126     require(_executionOperators[msg.sender], "only execution operators allowed to
            execute on SAND behalf");
127     return _approveAndExecuteWithSpecificGas(from, to, amount, gasLimit, data);
128 }
```

The code meets the specification.

Formal Verification Request 21

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

02, Dec 2019
0.65 ms

Line 113 in File ERC20ExecuteExtension.sol

```
113    //@CTK NO_BUF_OVERFLOW
```

Line 119-130 in File ERC20ExecuteExtension.sol

```
119    function approveAndExecuteWithSpecificGas(
120        address from,
121        address to,
122        uint256 amount,
123        uint256 gasLimit,
124        bytes calldata data
125    ) external returns (bool success, bytes memory returnData) {
126        require(_executionOperators[msg.sender], "only execution operators allowed to
               execute on SAND behalf");
127        return _approveAndExecuteWithSpecificGas(from, to, amount, gasLimit, data);
128    }
```

The code meets the specification.

Formal Verification Request 22

Method will not encounter an assertion failure.

02, Dec 2019
0.51 ms

Line 114 in File ERC20ExecuteExtension.sol

```
114    //@CTK NO_ASF
```

Line 119-130 in File ERC20ExecuteExtension.sol

```
119    function approveAndExecuteWithSpecificGas(
120        address from,
121        address to,
122        uint256 amount,
123        uint256 gasLimit,
124        bytes calldata data
125    ) external returns (bool success, bytes memory returnData) {
126        require(_executionOperators[msg.sender], "only execution operators allowed to
               execute on SAND behalf");
127        return _approveAndExecuteWithSpecificGas(from, to, amount, gasLimit, data);
128    }
```

The code meets the specification.

Formal Verification Request 23

approveAndExecuteWithSpecificGas_require

02, Dec 2019
1.28 ms

Line 115-118 in File ERC20ExecuteExtension.sol

```
/*@CTK approveAndExecuteWithSpecificGas_require
  @tag assume_completion
  @post _executionOperators[msg.sender] == true
 */
```

Line 119-130 in File ERC20ExecuteExtension.sol

```
function approveAndExecuteWithSpecificGas(
    address from,
    address to,
    uint256 amount,
    uint256 gasLimit,
    bytes calldata data
) external returns (bool success, bytes memory returnData) {
    require(_executionOperators[msg.sender], "only execution operators allowed to execute on SAND behalf");
    return _approveAndExecuteWithSpecificGas(from, to, amount, gasLimit, data);
}
```

The code meets the specification.

Formal Verification Request 24

If method completes, integer overflow would not happen.

02, Dec 2019
20.45 ms

Line 144 in File ERC20ExecuteExtension.sol

```
//@CTK NO_OVERFLOW
```

Line 151-171 in File ERC20ExecuteExtension.sol

```
function approveAndExecuteWithSpecificGasAndChargeForIt(
    address from,
    address to,
    uint256 amount,
    uint256 gasLimit,
    uint256 tokenGasPrice,
    uint256 baseGasCharge,
    address tokenReceiver,
    bytes calldata data
) external returns (bool success, bytes memory returnData) {
    uint256 initialGas = gasleft();
    require(_executionOperators[msg.sender], "only execution operators allowed to execute on SAND behalf");
    (success, returnData) = _approveAndExecuteWithSpecificGas(from, to, amount, gasLimit, data);
    if (tokenGasPrice > 0) {
```

```
        _charge(from, gasLimit, tokenGasPrice, initialGas, baseGasCharge, tokenReceiver);
    }
}
```

The code meets the specification.

Formal Verification Request 25

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

02, Dec 2019
0.45 ms

Line 145 in File ERC20ExecuteExtension.sol

```
145 //@CTK NO_BUF_OVERFLOW
```

Line 151-171 in File ERC20ExecuteExtension.sol

```
151 function approveAndExecuteWithSpecificGasAndChargeForIt(
152     address from,
153     address to,
154     uint256 amount,
155     uint256 gasLimit,
156     uint256 tokenGasPrice,
157     uint256 baseGasCharge,
158     address tokenReceiver,
159     bytes calldata data
160 ) external returns (bool success, bytes memory returnData) {
161     uint256 initialGas = gasleft();
162     require(_executionOperators[msg.sender], "only execution operators allowed to execute on SAND behalf");
163     (success, returnData) = _approveAndExecuteWithSpecificGas(from, to, amount, gasLimit, data);
164     if (tokenGasPrice > 0) {
165         _charge(from, gasLimit, tokenGasPrice, initialGas, baseGasCharge, tokenReceiver);
166     }
167 }
```

The code meets the specification.

Formal Verification Request 26

Method will not encounter an assertion failure.

02, Dec 2019
0.43 ms

Line 146 in File ERC20ExecuteExtension.sol

```
146 //@CTK NO_ASF
```

Line 151-171 in File ERC20ExecuteExtension.sol

```
151    function approveAndExecuteWithSpecificGasAndChargeForIt(
152        address from,
153        address to,
154        uint256 amount,
155        uint256 gasLimit,
156        uint256 tokenGasPrice,
157        uint256 baseGasCharge,
158        address tokenReceiver,
159        bytes calldata data
160    ) external returns (bool success, bytes memory returnData) {
161        uint256 initialGas = gasleft();
162        require(_executionOperators[msg.sender], "only execution operators allowed to
               execute on SAND behalf");
163        (success, returnData) = _approveAndExecuteWithSpecificGas(from, to, amount,
               gasLimit, data);
164        if (tokenGasPrice > 0) {
165            _charge(from, gasLimit, tokenGasPrice, initialGas, baseGasCharge,
                   tokenReceiver);
166        }
167    }
```

✅ The code meets the specification.

Formal Verification Request 27

approveAndExecuteWithSpecificGasAndChargeForIt_require

📅 02, Dec 2019
⏱ 1.02 ms

Line 147-150 in File ERC20ExecuteExtension.sol

```
147    /*@CTK approveAndExecuteWithSpecificGasAndChargeForIt_require
148      @tag assume_completion
149      @post _executionOperators[msg.sender] == true
150     */
```

Line 151-171 in File ERC20ExecuteExtension.sol

```
151    function approveAndExecuteWithSpecificGasAndChargeForIt(
152        address from,
153        address to,
154        uint256 amount,
155        uint256 gasLimit,
156        uint256 tokenGasPrice,
157        uint256 baseGasCharge,
158        address tokenReceiver,
159        bytes calldata data
160    ) external returns (bool success, bytes memory returnData) {
161        uint256 initialGas = gasleft();
162        require(_executionOperators[msg.sender], "only execution operators allowed to
               execute on SAND behalf");
163        (success, returnData) = _approveAndExecuteWithSpecificGas(from, to, amount,
               gasLimit, data);
164        if (tokenGasPrice > 0) {
165            _charge(from, gasLimit, tokenGasPrice, initialGas, baseGasCharge,
                   tokenReceiver);
166        }
```

```
}
```

The code meets the specification.

Formal Verification Request 28

If method completes, integer overflow would not happen.

02, Dec 2019
20.45 ms

Line 182 in File ERC20ExecuteExtension.sol

```
//@CTK NO_OVERFLOW
```

Line 194-214 in File ERC20ExecuteExtension.sol

```
function transferAndChargeForGas(
    address from,
    address to,
    uint256 amount,
    uint256 gasLimit,
    uint256 tokenGasPrice,
    uint256 baseGasCharge,
    address tokenReceiver
) external returns (bool) {
    uint256 initialGas = gasleft();
    require(_executionOperators[msg.sender], "only execution operators allowed to perfrom transfer and charge");
    _transfer(from, to, amount);
    if (tokenGasPrice > 0) {
        _charge(from, gasLimit, tokenGasPrice, initialGas, baseGasCharge, tokenReceiver);
    }
    return true;
}
```

The code meets the specification.

Formal Verification Request 29

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

02, Dec 2019
0.66 ms

Line 183 in File ERC20ExecuteExtension.sol

```
//@CTK NO_BUF_OVERFLOW
```

Line 194-214 in File ERC20ExecuteExtension.sol

```
function transferAndChargeForGas(
    address from,
    address to,
    uint256 amount,
    uint256 gasLimit,
```

```
        uint256 tokenGasPrice,
        uint256 baseGasCharge,
        address tokenReceiver
    ) external returns (bool) {
        uint256 initialGas = gasleft();
        require(_executionOperators[msg.sender], "only execution operators allowed to perfrom transfer and charge");
        _transfer(from, to, amount);
        if (tokenGasPrice > 0) {
            _charge(from, gasLimit, tokenGasPrice, initialGas, baseGasCharge, tokenReceiver);
        }
        return true;
    }
```

The code meets the specification.

Formal Verification Request 30

Method will not encounter an assertion failure.

02, Dec 2019
0.54 ms

Line 184 in File ERC20ExecuteExtension.sol

```
    //@CTK NO_ASF
```

Line 194-214 in File ERC20ExecuteExtension.sol

```
    function transferAndChargeForGas(
        address from,
        address to,
        uint256 amount,
        uint256 gasLimit,
        uint256 tokenGasPrice,
        uint256 baseGasCharge,
        address tokenReceiver
    ) external returns (bool) {
        uint256 initialGas = gasleft();
        require(_executionOperators[msg.sender], "only execution operators allowed to perfrom transfer and charge");
        _transfer(from, to, amount);
        if (tokenGasPrice > 0) {
            _charge(from, gasLimit, tokenGasPrice, initialGas, baseGasCharge, tokenReceiver);
        }
        return true;
    }
```

The code meets the specification.

Formal Verification Request 31

transferAndChargeForGas_require

02, Dec 2019

1.41 ms

Line 185-188 in File ERC20ExecuteExtension.sol

```
185 /*@CTK transferAndChargeForGas_require
186   @tag assume_completion
187   @post _executionOperators[msg.sender] == true
188  */
```

Line 194-214 in File ERC20ExecuteExtension.sol

```
194 function transferAndChargeForGas(
195     address from,
196     address to,
197     uint256 amount,
198     uint256 gasLimit,
199     uint256 tokenGasPrice,
200     uint256 baseGasCharge,
201     address tokenReceiver
202 ) external returns (bool) {
203     uint256 initialGas = gasleft();
204     require(_executionOperators[msg.sender], "only execution operators allowed to
            perfrom transfer and charge");
205     _transfer(from, to, amount);
206     if (tokenGasPrice > 0) {
207         _charge(from, gasLimit, tokenGasPrice, initialGas, baseGasCharge,
                tokenReceiver);
208     }
209     return true;
210 }
```

The code meets the specification.

Formal Verification Request 32

transferAndChargeForGas_return

02, Dec 2019
2.2 ms

Line 189-193 in File ERC20ExecuteExtension.sol

```
189 /*@CTK transferAndChargeForGas_return
190   @tag assume_completion
191   @pre _executionOperators[msg.sender] == true
192   @post __return == true
193  */
```

Line 194-214 in File ERC20ExecuteExtension.sol

```
194 function transferAndChargeForGas(
195     address from,
196     address to,
197     uint256 amount,
198     uint256 gasLimit,
199     uint256 tokenGasPrice,
200     uint256 baseGasCharge,
201     address tokenReceiver
```

```
202     ) external returns (bool) {
203         uint256 initialGas = gasleft();
204         require(_executionOperators[msg.sender], "only execution operators allowed to
                perfrom transfer and charge");
205         _transfer(from, to, amount);
206         if (tokenGasPrice > 0) {
207             _charge(from, gasLimit, tokenGasPrice, initialGas, baseGasCharge,
                    tokenReceiver);
208         }
209         return true;
210     }
```

The code meets the specification.

Formal Verification Request 33

If method completes, integer overflow would not happen.

02, Dec 2019
34.35 ms

Line 216 in File ERC20ExecuteExtension.sol

```
216     //@CTK NO_OVERFLOW
```

Line 229-250 in File ERC20ExecuteExtension.sol

```
229     function _charge(
230         address from,
231         uint256 gasLimit,
232         uint256 tokenGasPrice,
233         uint256 initialGas,
234         uint256 baseGasCharge,
235         address tokenReceiver
236     ) internal {
237         uint256 gasCharge;
238         gasCharge = initialGas - gasleft();
239         if(gasCharge > gasLimit) {
240             gasCharge = gasLimit;
241         }
242         gasCharge += baseGasCharge;
243         uint256 tokensToCharge = gasCharge * tokenGasPrice;
244         require(tokensToCharge / gasCharge == tokenGasPrice, "overflow");
245         _transfer(from, tokenReceiver, tokensToCharge);
246     }
```

The code meets the specification.

Formal Verification Request 34

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

02, Dec 2019
0.52 ms

Line 217 in File ERC20ExecuteExtension.sol

```
217 //@CTK NO_BUF_OVERFLOW
```

Line 229-250 in File ERC20ExecuteExtension.sol

```
229 function _charge(
230     address from,
231     uint256 gasLimit,
232     uint256 tokenGasPrice,
233     uint256 initialGas,
234     uint256 baseGasCharge,
235     address tokenReceiver
236 ) internal {
237     uint256 gasCharge;
238     gasCharge = initialGas - gasleft();
239     if(gasCharge > gasLimit) {
240         gasCharge = gasLimit;
241     }
242     gasCharge += baseGasCharge;
243     uint256 tokensToCharge = gasCharge * tokenGasPrice;
244     require(tokensToCharge / gasCharge == tokenGasPrice, "overflow");
245     _transfer(from, tokenReceiver, tokensToCharge);
246 }
```

✅ The code meets the specification.

Formal Verification Request 35

Method will not encounter an assertion failure.

📅 02, Dec 2019
⏱ 2.28 ms

Line 218 in File ERC20ExecuteExtension.sol

```
218 //@CTK FAIL NO_ASF
```

Line 229-250 in File ERC20ExecuteExtension.sol

```
229 function _charge(
230     address from,
231     uint256 gasLimit,
232     uint256 tokenGasPrice,
233     uint256 initialGas,
234     uint256 baseGasCharge,
235     address tokenReceiver
236 ) internal {
237     uint256 gasCharge;
238     gasCharge = initialGas - gasleft();
239     if(gasCharge > gasLimit) {
240         gasCharge = gasLimit;
241     }
242     gasCharge += baseGasCharge;
243     uint256 tokensToCharge = gasCharge * tokenGasPrice;
244     require(tokensToCharge / gasCharge == tokenGasPrice, "overflow");
245     _transfer(from, tokenReceiver, tokensToCharge);
246 }
```

❌ This code violates the specification.

```
Counter Example:
Before Execution:
    Input = {
        baseGasCharge = 0
        from = 0
        gasLimit = 0
        initialGas = 0
        tokenGasPrice = 0
        tokenReceiver = 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 = {
        block = {
          "number": 0,
          "timestamp": 0
        }
    }
    Address_Map = [
      {
        "key": "ALL_OTHERS",
        "value": {
          "contract_name": "ERC20ExecuteExtension",
          "balance": 0,
          "contract": {
            "_executionAdmin": 0,
            "_executionOperators": [
              {
                "key": "ALL_OTHERS",
                "value": false
              }
            ]
          }
        }
      }
    ]

Function invocation is reverted.
```

Formal Verification Request 36

_charge gas left lower than limit

📅 02, Dec 2019
⏱ 7.96 ms

Line 219-223 in File ERC20ExecuteExtension.sol

```
219     /*@CTK FAIL "_charge gas left lower than limit"
220       @tag assume_completion
221       @let uint256 gasCharge = initialGas - 1 + baseGasCharge
222       @post gasCharge * tokenGasPrice / gasCharge == tokenGasPrice
223      */
```

Line 229-250 in File ERC20ExecuteExtension.sol

```
229     function _charge(
230         address from,
231         uint256 gasLimit,
232         uint256 tokenGasPrice,
233         uint256 initialGas,
234         uint256 baseGasCharge,
235         address tokenReceiver
236     ) internal {
237         uint256 gasCharge;
238         gasCharge = initialGas - gasleft();
239         if(gasCharge > gasLimit) {
240             gasCharge = gasLimit;
241         }
242         gasCharge += baseGasCharge;
243         uint256 tokensToCharge = gasCharge * tokenGasPrice;
244         require(tokensToCharge / gasCharge == tokenGasPrice, "overflow");
245         _transfer(from, tokenReceiver, tokensToCharge);
246     }
```

This code violates the specification.

```
1  Counter Example:
2  Before Execution:
3      Input = {
4          baseGasCharge = 1
5          from = 0
6          gasLimit = 0
7          initialGas = 63
8          tokenGasPrice = 254
9          tokenReceiver = 0
10     }
11     This = 0
12     Internal = {
13         __has_assertion_failure = false
14         __has_buf_overflow = false
15         __has_overflow = false
16         __has_returned = false
17         __reverted = false
18         msg = {
19           "gas": 0,
20           "sender": 0,
21           "value": 0
22         }
23     }
24     Other = {
25         block = {
26           "number": 0,
27           "timestamp": 0
28         }
```

```
    }
    Address_Map = [
      {
        "key": "ALL_OTHERS",
        "value": {
          "contract_name": "ERC20ExecuteExtension",
          "balance": 0,
          "contract": {
            "_executionAdmin": 0,
            "_executionOperators": [
              {
                "key": "ALL_OTHERS",
                "value": false
              }
            ]
          }
        }
      }
    ]

After Execution:
    Input = {
        baseGasCharge = 1
        from = 0
        gasLimit = 0
        initialGas = 63
        tokenGasPrice = 254
        tokenReceiver = 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 = {
        block = {
          "number": 0,
          "timestamp": 0
        }
        gasCharge = 1
        tokensToCharge = 254
    }
    Address_Map = [
      {
        "key": "ALL_OTHERS",
        "value": {
          "contract_name": "ERC20ExecuteExtension",
          "balance": 0,
          "contract": {
            "_executionAdmin": 0,
```

```
          "_executionOperators": [
            {
              "key": "ALL_OTHERS",
              "value": false
            }
          ]
        }
      }
    }
  ]
```

Formal Verification Request 37

_charge gas left higher than limit

02, Dec 2019
8.73 ms

Line 224-228 in File ERC20ExecuteExtension.sol

```
224   /*@CTK FAIL "_charge gas left higher than limit"
225     @tag assume_completion
226     @let uint256 gasCharge = gasLimit + baseGasCharge
227     @post gasCharge * tokenGasPrice / gasCharge == tokenGasPrice
228    */
```

Line 229-250 in File ERC20ExecuteExtension.sol

```
229   function _charge(
230       address from,
231       uint256 gasLimit,
232       uint256 tokenGasPrice,
233       uint256 initialGas,
234       uint256 baseGasCharge,
235       address tokenReceiver
236   ) internal {
237       uint256 gasCharge;
238       gasCharge = initialGas - gasleft();
239       if(gasCharge > gasLimit) {
240           gasCharge = gasLimit;
241       }
242       gasCharge += baseGasCharge;
243       uint256 tokensToCharge = gasCharge * tokenGasPrice;
244       require(tokensToCharge / gasCharge == tokenGasPrice, "overflow");
245       _transfer(from, tokenReceiver, tokensToCharge);
246   }
```

This code violates the specification.

```
1  Counter Example:
2  Before Execution:
3      Input = {
4          baseGasCharge = 14
5          from = 0
6          gasLimit = 232
7          initialGas = 0
8          tokenGasPrice = 6
9          tokenReceiver = 0
10     }
```

```
    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 = {
        block = {
          "number": 0,
          "timestamp": 0
        }
    }
    Address_Map = [
      {
        "key": "ALL_OTHERS",
        "value": {
          "contract_name": "ERC20ExecuteExtension",
          "balance": 0,
          "contract": {
            "_executionAdmin": 0,
            "_executionOperators": [
              {
                "key": "ALL_OTHERS",
                "value": false
              }
            ]
          }
        }
      }
    ]

After Execution:
    Input = {
        baseGasCharge = 14
        from = 0
        gasLimit = 232
        initialGas = 0
        tokenGasPrice = 6
        tokenReceiver = 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 = {
        block = {
          "number": 0,
          "timestamp": 0
        }
        gasCharge = 14
        tokensToCharge = 84
    }
    Address_Map = [
      {
        "key": "ALL_OTHERS",
        "value": {
          "contract_name": "ERC20ExecuteExtension",
          "balance": 0,
          "contract": {
            "_executionAdmin": 0,
            "_executionOperators": [
              {
                "key": "ALL_OTHERS",
                "value": false
              }
            ]
          }
        }
      }
    ]
```

Formal Verification Request 38

If method completes, integer overflow would not happen.

📅 02, Dec 2019
⏱ 5.19 ms

Line 252 in File ERC20ExecuteExtension.sol

```
252 //@CTK NO_OVERFLOW
```

Line 255-272 in File ERC20ExecuteExtension.sol

```
255 function _approveAndExecuteWithSpecificGas(
256     address from,
257     address to,
258     uint256 amount,
259     uint256 gasLimit,
260     bytes memory data
261 ) internal returns (bool success, bytes memory returnData) {
262
263     if (amount > 0) {
264         _addAllowanceIfNeeded(from, to, amount);
265     }
266     (success, returnData) = to.call.gas(gasLimit)(data);
267     assert(gasleft() > gasLimit / 63); // not enough gas provided, assert to throw
            all gas // TODO use EIP-1930
268 }
```

The code meets the specification.

Formal Verification Request 39

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

02, Dec 2019
0.42 ms

Line 253 in File ERC20ExecuteExtension.sol

```
253    //@CTK NO_BUF_OVERFLOW
```

Line 255-272 in File ERC20ExecuteExtension.sol

```
255    function _approveAndExecuteWithSpecificGas(
256        address from,
257        address to,
258        uint256 amount,
259        uint256 gasLimit,
260        bytes memory data
261    ) internal returns (bool success, bytes memory returnData) {
262
263        if (amount > 0) {
264            _addAllowanceIfNeeded(from, to, amount);
265        }
266        (success, returnData) = to.call.gas(gasLimit)(data);
267        assert(gasleft() > gasLimit / 63); // not enough gas provided, assert to throw
               all gas // TODO use EIP-1930
268    }
```

The code meets the specification.

Formal Verification Request 40

Method will not encounter an assertion failure.

02, Dec 2019
0.37 ms

Line 254 in File ERC20ExecuteExtension.sol

```
254    //@CTK NO_ASF
```

Line 255-272 in File ERC20ExecuteExtension.sol

```
255    function _approveAndExecuteWithSpecificGas(
256        address from,
257        address to,
258        uint256 amount,
259        uint256 gasLimit,
260        bytes memory data
261    ) internal returns (bool success, bytes memory returnData) {
262
263        if (amount > 0) {
264            _addAllowanceIfNeeded(from, to, amount);
265        }
```

```
        (success, returnData) = to.call.gas(gasLimit)(data);
        assert(gasleft() > gasLimit / 63); // not enough gas provided, assert to throw
            all gas // TODO use EIP-1930
    }
```

The code meets the specification.

Formal Verification Request 41

If method completes, integer overflow would not happen.

02, Dec 2019
4.87 ms

Line 11 in File Admin.sol

```
    //@CTK NO_OVERFLOW
```

Line 18-20 in File Admin.sol

```
    function getAdmin() external view returns (address) {
        return _admin;
    }
```

The code meets the specification.

Formal Verification Request 42

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

02, Dec 2019
0.38 ms

Line 12 in File Admin.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 18-20 in File Admin.sol

```
    function getAdmin() external view returns (address) {
        return _admin;
    }
```

The code meets the specification.

Formal Verification Request 43

Method will not encounter an assertion failure.

02, Dec 2019
0.32 ms

Line 13 in File Admin.sol

```
    //@CTK NO_ASF
```

Line 18-20 in File Admin.sol

```
18  function getAdmin() external view returns (address) {
19      return _admin;
20  }
```

The code meets the specification.

Formal Verification Request 44

getAdmin

02, Dec 2019
0.35 ms

Line 14-17 in File Admin.sol

```
14  /*@CTK getAdmin
15    @tag assume_completion
16    @post __return == _admin
17   */
```

Line 18-20 in File Admin.sol

```
18  function getAdmin() external view returns (address) {
19      return _admin;
20  }
```

The code meets the specification.

Formal Verification Request 45

If method completes, integer overflow would not happen.

02, Dec 2019
16.16 ms

Line 24 in File Admin.sol

```
24  //@CTK NO_OVERFLOW
```

Line 36-40 in File Admin.sol

```
36  function changeAdmin(address newAdmin) external {
37      require(msg.sender == _admin, "only admin can change admin");
38      emit AdminChanged(_admin, newAdmin);
39      _admin = newAdmin;
40  }
```

The code meets the specification.

Formal Verification Request 46

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

02, Dec 2019
0.47 ms

Line 25 in File Admin.sol

```
25    //@CTK NO_BUF_OVERFLOW
```

Line 36-40 in File Admin.sol

```
36    function changeAdmin(address newAdmin) external {
37        require(msg.sender == _admin, "only admin can change admin");
38        emit AdminChanged(_admin, newAdmin);
39        _admin = newAdmin;
40    }
```

The code meets the specification.

Formal Verification Request 47

Method will not encounter an assertion failure.

02, Dec 2019
0.49 ms

Line 26 in File Admin.sol

```
26    //@CTK NO_ASF
```

Line 36-40 in File Admin.sol

```
36    function changeAdmin(address newAdmin) external {
37        require(msg.sender == _admin, "only admin can change admin");
38        emit AdminChanged(_admin, newAdmin);
39        _admin = newAdmin;
40    }
```

The code meets the specification.

Formal Verification Request 48

changeAdmin_requirement

02, Dec 2019
1.03 ms

Line 27-30 in File Admin.sol

```
27    /*@CTK changeAdmin_requirement
28      @tag assume_completion
29      @post msg.sender == _admin
30     */
```

Line 36-40 in File Admin.sol

```
36    function changeAdmin(address newAdmin) external {
37        require(msg.sender == _admin, "only admin can change admin");
38        emit AdminChanged(_admin, newAdmin);
39        _admin = newAdmin;
40    }
```

The code meets the specification.

Formal Verification Request 49

changeAdmin_change

02, Dec 2019
1.39 ms

Line 31-35 in File Admin.sol

```
31    /*@CTK changeAdmin_change
32      @tag assume_completion
33      @pre msg.sender == _admin
34      @post __post._admin == newAdmin
35     */
```

Line 36-40 in File Admin.sol

```
36    function changeAdmin(address newAdmin) external {
37        require(msg.sender == _admin, "only admin can change admin");
38        emit AdminChanged(_admin, newAdmin);
39        _admin = newAdmin;
40    }
```

The code meets the specification.

Formal Verification Request 50

If method completes, integer overflow would not happen.

02, Dec 2019
19.9 ms

Line 15 in File SuperOperators.sol

```
15    //@CTK NO_OVERFLOW
```

Line 27-34 in File SuperOperators.sol

```
27    function setSuperOperator(address superOperator, bool enabled) external {
28        require(
29            msg.sender == _admin,
30            "only admin is allowed to add super operators"
31        );
32        _superOperators[superOperator] = enabled;
33        emit SuperOperator(superOperator, enabled);
34    }
```

The code meets the specification.

Formal Verification Request 51

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

02, Dec 2019
0.52 ms

Line 16 in File SuperOperators.sol

```
16    //@CTK NO_BUF_OVERFLOW
```

Line 27-34 in File SuperOperators.sol

```
27    function setSuperOperator(address superOperator, bool enabled) external {
28        require(
29            msg.sender == _admin,
30            "only admin is allowed to add super operators"
31        );
32        _superOperators[superOperator] = enabled;
33        emit SuperOperator(superOperator, enabled);
34    }
```

The code meets the specification.

Formal Verification Request 52

Method will not encounter an assertion failure.

02, Dec 2019
0.45 ms

Line 17 in File SuperOperators.sol

```
17    //@CTK NO_ASF
```

Line 27-34 in File SuperOperators.sol

```
27    function setSuperOperator(address superOperator, bool enabled) external {
28        require(
29            msg.sender == _admin,
30            "only admin is allowed to add super operators"
31        );
32        _superOperators[superOperator] = enabled;
33        emit SuperOperator(superOperator, enabled);
34    }
```

The code meets the specification.

Formal Verification Request 53

setSuperOperator_admin

02, Dec 2019
0.33 ms

Line 18-21 in File SuperOperators.sol

```
18  /*@CTK setSuperOperator_admin
19    @tag assume_completion
20    @inv msg.sender == _admin
21   */
```

Line 27-34 in File SuperOperators.sol

```
27  function setSuperOperator(address superOperator, bool enabled) external {
28      require(
29          msg.sender == _admin,
30          "only admin is allowed to add super operators"
31      );
32      _superOperators[superOperator] = enabled;
33      emit SuperOperator(superOperator, enabled);
34  }
```

✅ The code meets the specification.

Formal Verification Request 54

setSuperOperator_change

📅 02, Dec 2019
⏱ 1.74 ms

Line 22-26 in File SuperOperators.sol

```
22  /*@CTK setSuperOperator_change
23    @tag assume_completion
24    @pre msg.sender == _admin
25    @post __post._superOperators[superOperator] == enabled
26   */
```

Line 27-34 in File SuperOperators.sol

```
27  function setSuperOperator(address superOperator, bool enabled) external {
28      require(
29          msg.sender == _admin,
30          "only admin is allowed to add super operators"
31      );
32      _superOperators[superOperator] = enabled;
33      emit SuperOperator(superOperator, enabled);
34  }
```

✅ The code meets the specification.

Formal Verification Request 55

If method completes, integer overflow would not happen.

📅 02, Dec 2019
⏱ 5.25 ms

Line 39 in File SuperOperators.sol

```
39  //@CTK NO_OVERFLOW
```

Line 46-48 in File SuperOperators.sol

```
46    function isSuperOperator(address who) public view returns (bool) {
47        return _superOperators[who];
48    }
```

The code meets the specification.

Formal Verification Request 56

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

02, Dec 2019
0.32 ms

Line 40 in File SuperOperators.sol

```
40    //@CTK NO_BUF_OVERFLOW
```

Line 46-48 in File SuperOperators.sol

```
46    function isSuperOperator(address who) public view returns (bool) {
47        return _superOperators[who];
48    }
```

The code meets the specification.

Formal Verification Request 57

Method will not encounter an assertion failure.

02, Dec 2019
0.32 ms

Line 41 in File SuperOperators.sol

```
41    //@CTK NO_ASF
```

Line 46-48 in File SuperOperators.sol

```
46    function isSuperOperator(address who) public view returns (bool) {
47        return _superOperators[who];
48    }
```

The code meets the specification.

Formal Verification Request 58

isSuperOperator

02, Dec 2019
0.33 ms

Line 42-45 in File SuperOperators.sol

```
42    /*@CTK isSuperOperator
43      @tag assume_completion
44      @post __return == _superOperators[who]
45     */
```

Line 46-48 in File SuperOperators.sol

```
46  function isSuperOperator(address who) public view returns (bool) {
47      return _superOperators[who];
48  }
```

✅ The code meets the specification.

Formal Verification Request 59

If method completes, integer overflow would not happen.

📅 02, Dec 2019
⏱ 75.43 ms

Line 9 in File Sand.sol

```
9   //@CTK NO_OVERFLOW
```

Line 25-29 in File Sand.sol

```
25  constructor(address sandAdmin, address executionAdmin, address beneficiary) public
        {
26      _admin = sandAdmin;
27      _executionAdmin = executionAdmin;
28      _mint(beneficiary, 3000000000000000000000000000);
29  }
```

✅ The code meets the specification.

Formal Verification Request 60

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

📅 02, Dec 2019
⏱ 6.31 ms

Line 10 in File Sand.sol

```
10  //@CTK NO_BUF_OVERFLOW
```

Line 25-29 in File Sand.sol

```
25  constructor(address sandAdmin, address executionAdmin, address beneficiary) public
        {
26      _admin = sandAdmin;
27      _executionAdmin = executionAdmin;
28      _mint(beneficiary, 3000000000000000000000000000);
29  }
```

✅ The code meets the specification.

Formal Verification Request 61

Method will not encounter an assertion failure.

02, Dec 2019
6.36 ms

Line 11 in File Sand.sol

```
11    //@CTK NO_ASF
```

Line 25-29 in File Sand.sol

```
25    constructor(address sandAdmin, address executionAdmin, address beneficiary) public
          {
26        _admin = sandAdmin;
27        _executionAdmin = executionAdmin;
28        _mint(beneficiary, 3000000000000000000000000000);
29    }
```

The code meets the specification.

Formal Verification Request 62

Sand_require

02, Dec 2019
5.88 ms

Line 12-16 in File Sand.sol

```
12    /*@CTK Sand_require
13      @tag assume_completion
14      @post beneficiary != address(0)
15      @post _totalSupply + 3000000000000000000000000000 > _totalSupply
16     */
```

Line 25-29 in File Sand.sol

```
25    constructor(address sandAdmin, address executionAdmin, address beneficiary) public
          {
26        _admin = sandAdmin;
27        _executionAdmin = executionAdmin;
28        _mint(beneficiary, 3000000000000000000000000000);
29    }
```

The code meets the specification.

Formal Verification Request 63

Sand_change

02, Dec 2019
0.37 ms

Line 17-24 in File Sand.sol

```
17  /*@CTK Sand_change
18    @tag assume_completion
19    @pre beneficiary != address(0)
20    @pre _totalSupply + 3000000000000000000000000000 > _totalSupply
21    @post __post._totalSupply == _totalSupply + 3000000000000000000000000000
22    @post __post._balances[beneficiary] == _balances[beneficiary] +
        3000000000000000000000000000
23    @post __post._balances[beneficiary] == __post._totalSupply
24   */
```

Line 25-29 in File Sand.sol

```
25  constructor(address sandAdmin, address executionAdmin, address beneficiary) public
      {
26      _admin = sandAdmin;
27      _executionAdmin = executionAdmin;
28      _mint(beneficiary, 3000000000000000000000000000);
29  }
```

✅ The code meets the specification.

Formal Verification Request 64

If method completes, integer overflow would not happen.

📅 02, Dec 2019
⏱ 4.58 ms

Line 33 in File Sand.sol

```
33  //@CTK NO_OVERFLOW
```

Line 39-41 in File Sand.sol

```
39  function name() public view returns (string memory) {
40      return "SAND";
41  }
```

✅ The code meets the specification.

Formal Verification Request 65

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

📅 02, Dec 2019
⏱ 0.28 ms

Line 34 in File Sand.sol

```
34  //@CTK NO_BUF_OVERFLOW
```

Line 39-41 in File Sand.sol

```
39  function name() public view returns (string memory) {
40      return "SAND";
41  }
```

✅ The code meets the specification.

Formal Verification Request 66

Method will not encounter an assertion failure.

02, Dec 2019
0.27 ms

Line 35 in File Sand.sol

```
35    //@CTK NO_ASF
```

Line 39-41 in File Sand.sol

```
39    function name() public view returns (string memory) {
40        return "SAND";
41    }
```

The code meets the specification.

Formal Verification Request 67

name

02, Dec 2019
0.28 ms

Line 36-38 in File Sand.sol

```
36    /*@CTK name
37      @post __return == "SAND"
38     */
```

Line 39-41 in File Sand.sol

```
39    function name() public view returns (string memory) {
40        return "SAND";
41    }
```

The code meets the specification.

Formal Verification Request 68

If method completes, integer overflow would not happen.

02, Dec 2019
4.72 ms

Line 47 in File Sand.sol

```
47    //@CTK NO_OVERFLOW
```

Line 53-55 in File Sand.sol

```
53    function symbol() public view returns (string memory) {
54        return "SAND";
55    }
```

The code meets the specification.

Formal Verification Request 69

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

02, Dec 2019
0.28 ms

Line 48 in File Sand.sol

```
48    //@CTK NO_BUF_OVERFLOW
```

Line 53-55 in File Sand.sol

```
53    function symbol() public view returns (string memory) {
54        return "SAND";
55    }
```

The code meets the specification.

Formal Verification Request 70

Method will not encounter an assertion failure.

02, Dec 2019
0.28 ms

Line 49 in File Sand.sol

```
49    //@CTK NO_ASF
```

Line 53-55 in File Sand.sol

```
53    function symbol() public view returns (string memory) {
54        return "SAND";
55    }
```

The code meets the specification.

Formal Verification Request 71

symbol

02, Dec 2019
0.29 ms

Line 50-52 in File Sand.sol

```
50    /*@CTK symbol
51      @post __return == "SAND"
52     */
```

Line 53-55 in File Sand.sol

```
53    function symbol() public view returns (string memory) {
54        return "SAND";
55    }
```

The code meets the specification.

Formal Verification Request 72

totalSupply

02, Dec 2019
11.19 ms

Line 14-16 in File ERC20BaseToken.sol

```
14    /*@CTK totalSupply
15      @post __return == _totalSupply
16     */
```

Line 17-19 in File ERC20BaseToken.sol

```
17    function totalSupply() public view returns (uint256) {
18        return _totalSupply;
19    }
```

The code meets the specification.

Formal Verification Request 73

If method completes, integer overflow would not happen.

02, Dec 2019
5.34 ms

Line 24 in File ERC20BaseToken.sol

```
24    //@CTK NO_OVERFLOW
```

Line 30-32 in File ERC20BaseToken.sol

```
30    function balanceOf(address owner) public view returns (uint256) {
31        return _balances[owner];
32    }
```

The code meets the specification.

Formal Verification Request 74

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

02, Dec 2019
0.38 ms

Line 25 in File ERC20BaseToken.sol

```
25    //@CTK NO_BUF_OVERFLOW
```

Line 30-32 in File ERC20BaseToken.sol

```
30    function balanceOf(address owner) public view returns (uint256) {
31        return _balances[owner];
32    }
```

The code meets the specification.

Formal Verification Request 75

Method will not encounter an assertion failure.

02, Dec 2019
0.39 ms

Line 26 in File ERC20BaseToken.sol

```
26  //@CTK NO_ASF
```

Line 30-32 in File ERC20BaseToken.sol

```
30  function balanceOf(address owner) public view returns (uint256) {
31      return _balances[owner];
32  }
```

The code meets the specification.

Formal Verification Request 76

balanceOf

02, Dec 2019
0.36 ms

Line 27-29 in File ERC20BaseToken.sol

```
27  /*@CTK balanceOf
28    @post __return == _balances[owner]
29   */
```

Line 30-32 in File ERC20BaseToken.sol

```
30  function balanceOf(address owner) public view returns (uint256) {
31      return _balances[owner];
32  }
```

The code meets the specification.

Formal Verification Request 77

If method completes, integer overflow would not happen.

02, Dec 2019
5.09 ms

Line 38 in File ERC20BaseToken.sol

```
38  //@CTK NO_OVERFLOW
```

Line 44-50 in File ERC20BaseToken.sol

```
44  function allowance(address owner, address spender)
45      public
46      view
47      returns (uint256)
48  {
49      return _allowances[owner][spender];
50  }
```

The code meets the specification.

Formal Verification Request 78

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

02, Dec 2019
0.37 ms

Line 39 in File ERC20BaseToken.sol

```
39    //@CTK NO_BUF_OVERFLOW
```

Line 44-50 in File ERC20BaseToken.sol

```
44    function allowance(address owner, address spender)
45        public
46        view
47        returns (uint256)
48    {
49        return _allowances[owner][spender];
50    }
```

The code meets the specification.

Formal Verification Request 79

Method will not encounter an assertion failure.

02, Dec 2019
0.35 ms

Line 40 in File ERC20BaseToken.sol

```
40    //@CTK NO_ASF
```

Line 44-50 in File ERC20BaseToken.sol

```
44    function allowance(address owner, address spender)
45        public
46        view
47        returns (uint256)
48    {
49        return _allowances[owner][spender];
50    }
```

The code meets the specification.

Formal Verification Request 80

allowance

02, Dec 2019
0.35 ms

Line 41-43 in File ERC20BaseToken.sol

```
41  /*@CTK allowance
42    @post __return == _allowances[owner][spender]
43   */
```

Line 44-50 in File ERC20BaseToken.sol

```
44  function allowance(address owner, address spender)
45      public
46      view
47      returns (uint256)
48  {
49      return _allowances[owner][spender];
50  }
```

✅ The code meets the specification.

Formal Verification Request 81

decimals

📅 02, Dec 2019
⏱ 4.84 ms

Line 54-56 in File ERC20BaseToken.sol

```
54  /*@CTK decimals
55    @post __return == 18
56   */
```

Line 57-59 in File ERC20BaseToken.sol

```
57  function decimals() public view returns (uint8) {
58      return uint8(18);
59  }
```

✅ The code meets the specification.

Formal Verification Request 82

If method completes, integer overflow would not happen.

📅 02, Dec 2019
⏱ 110.71 ms

Line 65 in File ERC20BaseToken.sol

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

Line 82-88 in File ERC20BaseToken.sol

```
82  function transfer(address to, uint256 amount)
83      public
84      returns (bool)
85  {
86      _transfer(msg.sender, to, amount);
87      return true;
88  }
```

This code violates the specification.

```
Counter Example:
Before Execution:
    Input = {
        amount = 128
        to = 2
    }
    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 = {
        __return = false
        block = {
          "number": 0,
          "timestamp": 0
        }
    }
    Address_Map = [
      {
        "key": 0,
        "value": {
          "contract_name": "ERC20BaseToken",
          "balance": 0,
          "contract": {
            "_totalSupply": 0,
            "_balances": [
              {
                "key": "ALL_OTHERS",
                "value": 128
              }
            ],
            "_allowances": [
              {
                "key": "ALL_OTHERS",
                "value": [
                  {
                    "key": "ALL_OTHERS",
                    "value": 128
                  }
                ]
              }
            ],
            "_superOperators": [
              {
                "key": "ALL_OTHERS",
                "value": false
              }
            ],
```

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

After Execution:
  Input = {
      amount = 128
      to = 2
  }
  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 = {
      __return = true
      block = {
        "number": 0,
        "timestamp": 0
      }
  }
  Address_Map = [
    {
      "key": 0,
      "value": {
        "contract_name": "ERC20BaseToken",
        "balance": 0,
        "contract": {
          "_totalSupply": 0,
          "_balances": [
            {
              "key": 0,
              "value": 0
            },
            {
              "key": 2,
              "value": 0
            },
            {
              "key": "ALL_OTHERS",
              "value": 128
            }
          ],
          "_allowances": [
```

```
            {
              "key": "ALL_OTHERS",
              "value": [
                {
                  "key": "ALL_OTHERS",
                  "value": 128
                }
              ]
            }
          ],
          "_superOperators": [
            {
              "key": "ALL_OTHERS",
              "value": false
            }
          ],
          "_admin": 0
        }
      }
    },
    {
      "key": "ALL_OTHERS",
      "value": "EmptyAddress"
    }
  ]
```


Formal Verification Request 83

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

📅 02, Dec 2019
⏱ 1.15 ms

Line 66 in File ERC20BaseToken.sol

```
66  //@CTK NO_BUF_OVERFLOW
```

Line 82-88 in File ERC20BaseToken.sol

```
82  function transfer(address to, uint256 amount)
83      public
84      returns (bool)
85  {
86      _transfer(msg.sender, to, amount);
87      return true;
88  }
```

✅ The code meets the specification.

Formal Verification Request 84

Method will not encounter an assertion failure.

📅 02, Dec 2019
⏱ 1.03 ms

Line 67 in File ERC20BaseToken.sol

```
67    //@CTK NO_ASF
```

Line 82-88 in File ERC20BaseToken.sol

```
82    function transfer(address to, uint256 amount)
83        public
84        returns (bool)
85    {
86        _transfer(msg.sender, to, amount);
87        return true;
88    }
```

✅ The code meets the specification.

Formal Verification Request 85

transfer_require

📅 02, Dec 2019
⏱ 10.99 ms

Line 68-72 in File ERC20BaseToken.sol

```
68    /*@CTK transfer_require
69      @tag assume_completion
70      @post to != address(0)
71      @post _balances[msg.sender] >= amount
72     */
```

Line 82-88 in File ERC20BaseToken.sol

```
82    function transfer(address to, uint256 amount)
83        public
84        returns (bool)
85    {
86        _transfer(msg.sender, to, amount);
87        return true;
88    }
```

✅ The code meets the specification.

Formal Verification Request 86

transfer_change

📅 02, Dec 2019
⏱ 22.5 ms

Line 73-81 in File ERC20BaseToken.sol

```
73    /*@CTK transfer_change
74      @tag assume_completion
75      @pre to != address(0)
76      @pre _balances[msg.sender] >= amount
77      @pre msg.sender != to
78      @post __post._balances[msg.sender] == _balances[msg.sender] - amount
79      @post __post._balances[to] == _balances[to] + amount
```

```
80      @post __return == true
81     */
```

Line 82-88 in File ERC20BaseToken.sol

```
82    function transfer(address to, uint256 amount)
83        public
84        returns (bool)
85    {
86        _transfer(msg.sender, to, amount);
87        return true;
88    }
```

✅ The code meets the specification.

Formal Verification Request 87

If method completes, integer overflow would not happen.

📅 02, Dec 2019
⏱ 117.57 ms

Line 95 in File ERC20BaseToken.sol

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

Line 115-129 in File ERC20BaseToken.sol

```
115   function transferFrom(address from, address to, uint256 amount)
116       public
117       returns (bool)
118   {
119       if (msg.sender != from && !_superOperators[msg.sender]) {
120           uint256 currentAllowance = _allowances[from][msg.sender];
121           if (currentAllowance != (2**256) - 1) {
122               // save gas when allowance is maximal by not reducing it (see https://
                      github.com/ethereum/EIPs/issues/717)
123               require(currentAllowance >= amount, "Not enough funds allowed");
124               _allowances[from][msg.sender] = currentAllowance - amount;
125           }
126       }
127       _transfer(from, to, amount);
128       return true;
129   }
```

❌ This code violates the specification.

```
1  Counter Example:
2  Before Execution:
3      Input = {
4          amount = 128
5          from = 0
6          to = 1
7      }
8      This = 0
9      Internal = {
10         __has_assertion_failure = false
11         __has_buf_overflow = false
12         __has_overflow = false
```

```
        __has_returned = false
        __reverted = false
        msg = {
          "gas": 0,
          "sender": 128,
          "value": 0
        }
    }
    Other = {
        __return = false
        block = {
          "number": 0,
          "timestamp": 0
        }
    }
    Address_Map = [
      {
        "key": 0,
        "value": {
          "contract_name": "ERC20BaseToken",
          "balance": 0,
          "contract": {
            "_totalSupply": 0,
            "_balances": [
              {
                "key": 2,
                "value": 4
              },
              {
                "key": 0,
                "value": 128
              },
              {
                "key": 4,
                "value": 4
              },
              {
                "key": 1,
                "value": 128
              },
              {
                "key": "ALL_OTHERS",
                "value": 46
              }
            ],
            "_allowances": [
              {
                "key": 0,
                "value": [
                  {
                    "key": 0,
                    "value": 2
                  },
                  {
                    "key": 128,
                    "value": 174
                  },
                  {
```

```
                  "key": "ALL_OTHERS",
                  "value": 0
                }
              ]
            },
            {
              "key": "ALL_OTHERS",
              "value": [
                {
                  "key": "ALL_OTHERS",
                  "value": 46
                }
              ]
            }
          ],
          "_superOperators": [
            {
              "key": "ALL_OTHERS",
              "value": false
            }
          ],
          "_admin": 0
        }
      }
    },
    {
      "key": "ALL_OTHERS",
      "value": "EmptyAddress"
    }
  ]

After Execution:
  Input = {
      amount = 128
      from = 0
      to = 1
  }
  This = 0
  Internal = {
      __has_assertion_failure = false
      __has_buf_overflow = false
      __has_overflow = true
      __has_returned = true
      __reverted = false
      msg = {
        "gas": 0,
        "sender": 128,
        "value": 0
      }
  }
  Other = {
      __return = true
      block = {
        "number": 0,
        "timestamp": 0
      }
  }
  Address_Map = [
```

```
    {
      "key": 0,
      "value": {
        "contract_name": "ERC20BaseToken",
        "balance": 0,
        "contract": {
          "_totalSupply": 0,
          "_balances": [
            {
              "key": 2,
              "value": 4
            },
            {
              "key": 0,
              "value": 0
            },
            {
              "key": 4,
              "value": 4
            },
            {
              "key": 1,
              "value": 0
            },
            {
              "key": "ALL_OTHERS",
              "value": 46
            }
          ],
          "_allowances": [
            {
              "key": 0,
              "value": [
                {
                  "key": 0,
                  "value": 2
                },
                {
                  "key": 128,
                  "value": 46
                },
                {
                  "key": "ALL_OTHERS",
                  "value": 0
                }
              ]
            },
            {
              "key": "ALL_OTHERS",
              "value": [
                {
                  "key": "ALL_OTHERS",
                  "value": 46
                }
              ]
            }
          ],
          "_superOperators": [
```

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

Formal Verification Request 88

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

📅 02, Dec 2019
⏱ 42.83 ms

Line 96 in File ERC20BaseToken.sol

```
96  //@CTK NO_BUF_OVERFLOW
```

Line 115-129 in File ERC20BaseToken.sol

```
115 function transferFrom(address from, address to, uint256 amount)
116     public
117     returns (bool)
118 {
119     if (msg.sender != from && !_superOperators[msg.sender]) {
120         uint256 currentAllowance = _allowances[from][msg.sender];
121         if (currentAllowance != (2**256) - 1) {
122             // save gas when allowance is maximal by not reducing it (see https://
                    github.com/ethereum/EIPs/issues/717)
123             require(currentAllowance >= amount, "Not enough funds allowed");
124             _allowances[from][msg.sender] = currentAllowance - amount;
125         }
126     }
127     _transfer(from, to, amount);
128     return true;
129 }
```

✅ The code meets the specification.

Formal Verification Request 89

Method will not encounter an assertion failure.

📅 02, Dec 2019
⏱ 41.27 ms

Line 97 in File ERC20BaseToken.sol

```
//@CTK NO_ASF
```

Line 115-129 in File ERC20BaseToken.sol

```
function transferFrom(address from, address to, uint256 amount)
    public
    returns (bool)
{
    if (msg.sender != from && !_superOperators[msg.sender]) {
        uint256 currentAllowance = _allowances[from][msg.sender];
        if (currentAllowance != (2**256) - 1) {
            // save gas when allowance is maximal by not reducing it (see https://
                github.com/ethereum/EIPs/issues/717)
            require(currentAllowance >= amount, "Not enough funds allowed");
            _allowances[from][msg.sender] = currentAllowance - amount;
        }
    }
    _transfer(from, to, amount);
    return true;
}
```

The code meets the specification.

Formal Verification Request 90

transferFrom_require

02, Dec 2019
39.52 ms

Line 98-103 in File ERC20BaseToken.sol

```
/*@CTK transferFrom_require
  @tag assume_completion
  @post to != address(0)
  @post _balances[from] >= amount
  @post (msg.sender != from && !_superOperators[msg.sender] && _allowances[from][
      msg.sender] != (2**256) - 1) -> _allowances[from][msg.sender] >= amount
 */
```

Line 115-129 in File ERC20BaseToken.sol

```
function transferFrom(address from, address to, uint256 amount)
    public
    returns (bool)
{
    if (msg.sender != from && !_superOperators[msg.sender]) {
        uint256 currentAllowance = _allowances[from][msg.sender];
        if (currentAllowance != (2**256) - 1) {
            // save gas when allowance is maximal by not reducing it (see https://
                github.com/ethereum/EIPs/issues/717)
            require(currentAllowance >= amount, "Not enough funds allowed");
            _allowances[from][msg.sender] = currentAllowance - amount;
        }
    }
    _transfer(from, to, amount);
    return true;
}
```

The code meets the specification.

Formal Verification Request 91

transferFrom_change

02, Dec 2019
311.45 ms

Line 104-114 in File ERC20BaseToken.sol

```
104   /*@CTK transferFrom_change
105     @tag assume_completion
106     @pre to != address(0)
107     @pre _balances[from] >= amount
108     @pre (msg.sender != from && !_superOperators[msg.sender] && _allowances[from][msg
          .sender] != (2**256) - 1) -> _allowances[from][msg.sender] >= amount
109     @pre from != to
110     @post (msg.sender != from && !_superOperators[msg.sender] && _allowances[from][
          msg.sender] != (2**256) - 1) -> __post._allowances[from][msg.sender] ==
          _allowances[from][msg.sender] - amount
111     @post __post._balances[from] == _balances[from] - amount
112     @post __post._balances[to] == _balances[to] + amount
113     @post __return == true
114    */
```

Line 115-129 in File ERC20BaseToken.sol

```
115   function transferFrom(address from, address to, uint256 amount)
116       public
117       returns (bool)
118   {
119       if (msg.sender != from && !_superOperators[msg.sender]) {
120           uint256 currentAllowance = _allowances[from][msg.sender];
121           if (currentAllowance != (2**256) - 1) {
122               // save gas when allowance is maximal by not reducing it (see https://
                    github.com/ethereum/EIPs/issues/717)
123               require(currentAllowance >= amount, "Not enough funds allowed");
124               _allowances[from][msg.sender] = currentAllowance - amount;
125           }
126       }
127       _transfer(from, to, amount);
128       return true;
129   }
```

The code meets the specification.

Formal Verification Request 92

If method completes, integer overflow would not happen.

02, Dec 2019
118.45 ms

Line 134 in File ERC20BaseToken.sol

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

Line 150-153 in File ERC20BaseToken.sol

```
150     function burn(uint256 amount) external returns (bool) {
151         _burn(msg.sender, amount);
152         return true;
153     }
```

This code violates the specification.

```
Counter Example:
Before Execution:
    Input = {
        amount = 254
    }
    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 = {
        __return = false
        block = {
          "number": 0,
          "timestamp": 0
        }
    }
    Address_Map = [
      {
        "key": 0,
        "value": {
          "contract_name": "ERC20BaseToken",
          "balance": 0,
          "contract": {
            "_totalSupply": 0,
            "_balances": [
              {
                "key": 0,
                "value": 255
              },
              {
                "key": "ALL_OTHERS",
                "value": 254
              }
            ],
            "_allowances": [
              {
                "key": "ALL_OTHERS",
                "value": [
                  {
                    "key": "ALL_OTHERS",
```

```
                  "value": 254
                }
              ]
            }
          ],
          "_superOperators": [
            {
              "key": "ALL_OTHERS",
              "value": false
            }
          ],
          "_admin": 0
        }
      }
    },
    {
      "key": "ALL_OTHERS",
      "value": "EmptyAddress"
    }
  ]

After Execution:
  Input = {
      amount = 254
  }
  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 = {
      __return = true
      block = {
        "number": 0,
        "timestamp": 0
      }
  }
  Address_Map = [
    {
      "key": 0,
      "value": {
        "contract_name": "ERC20BaseToken",
        "balance": 0,
        "contract": {
          "_totalSupply": 2,
          "_balances": [
            {
              "key": 0,
              "value": 1
            },
```

```
            {
              "key": "ALL_OTHERS",
              "value": 254
            }
          ],
          "_allowances": [
            {
              "key": "ALL_OTHERS",
              "value": [
                {
                  "key": "ALL_OTHERS",
                  "value": 254
                }
              ]
            }
          ],
          "_superOperators": [
            {
              "key": "ALL_OTHERS",
              "value": false
            }
          ],
          "_admin": 0
        }
      }
    },
    {
      "key": "ALL_OTHERS",
      "value": "EmptyAddress"
    }
  ]
```

Formal Verification Request 93

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

📅 02, Dec 2019
⏱ 6.71 ms

Line 135 in File ERC20BaseToken.sol

```
135 //@CTK NO_BUF_OVERFLOW
```

Line 150-153 in File ERC20BaseToken.sol

```
150 function burn(uint256 amount) external returns (bool) {
151     _burn(msg.sender, amount);
152     return true;
153 }
```

✅ The code meets the specification.

Formal Verification Request 94

Method will not encounter an assertion failure.

📅 02, Dec 2019

6.13 ms

Line 136 in File ERC20BaseToken.sol

```
136    //@CTK NO_ASF
```

Line 150-153 in File ERC20BaseToken.sol

```
150    function burn(uint256 amount) external returns (bool) {
151        _burn(msg.sender, amount);
152        return true;
153    }
```

The code meets the specification.

Formal Verification Request 95

burn_require

02, Dec 2019
13.84 ms

Line 137-141 in File ERC20BaseToken.sol

```
137    /*@CTK burn_require
138      @tag assume_completion
139      @post amount > 0
140      @post _balances[msg.sender] >= amount
141     */
```

Line 150-153 in File ERC20BaseToken.sol

```
150    function burn(uint256 amount) external returns (bool) {
151        _burn(msg.sender, amount);
152        return true;
153    }
```

The code meets the specification.

Formal Verification Request 96

burn_change

02, Dec 2019
93.23 ms

Line 142-149 in File ERC20BaseToken.sol

```
142    /*@CTK burn_change
143      @tag assume_completion
144      @pre amount > 0
145      @pre _balances[msg.sender] >= amount
146      @post __post._balances[msg.sender] == _balances[msg.sender] - amount
147      @post __post._totalSupply == _totalSupply - amount
148      @post __return == true
149     */
```

Line 150-153 in File ERC20BaseToken.sol

```
    function burn(uint256 amount) external returns (bool) {
        _burn(msg.sender, amount);
        return true;
    }
```

The code meets the specification.

Formal Verification Request 97

If method completes, integer overflow would not happen.

02, Dec 2019
103.77 ms

Line 159 in File ERC20BaseToken.sol

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

Line 177-180 in File ERC20BaseToken.sol

```
177     function burnFor(address owner, uint256 amount) external returns (bool) {
178         _burn(owner, amount);
179         return true;
180     }
```

This code violates the specification.

```
Counter Example:
Before Execution:
    Input = {
        amount = 127
        owner = 2
    }
    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 = {
        __return = false
        block = {
          "number": 0,
          "timestamp": 0
        }
    }
    Address_Map = [
      {
        "key": 0,
        "value": {
          "contract_name": "ERC20BaseToken",
```

```
        "balance": 0,
        "contract": {
          "_totalSupply": 0,
          "_balances": [
            {
              "key": 66,
              "value": 0
            },
            {
              "key": 0,
              "value": 1
            },
            {
              "key": 4,
              "value": 0
            },
            {
              "key": 2,
              "value": 234
            },
            {
              "key": 128,
              "value": 0
            },
            {
              "key": "ALL_OTHERS",
              "value": 64
            }
          ],
          "_allowances": [
            {
              "key": 0,
              "value": [
                {
                  "key": 0,
                  "value": 0
                },
                {
                  "key": "ALL_OTHERS",
                  "value": 64
                }
              ]
            },
            {
              "key": 1,
              "value": [
                {
                  "key": 0,
                  "value": 1
                },
                {
                  "key": "ALL_OTHERS",
                  "value": 64
                }
              ]
            },
            {
              "key": "ALL_OTHERS",
```

```
              "value": [
                {
                  "key": "ALL_OTHERS",
                  "value": 148
                }
              ]
            }
          ],
          "_superOperators": [
            {
              "key": 0,
              "value": true
            },
            {
              "key": 2,
              "value": true
            },
            {
              "key": "ALL_OTHERS",
              "value": false
            }
          ],
          "_admin": 0
        }
      }
    },
    {
      "key": "ALL_OTHERS",
      "value": "EmptyAddress"
    }
  ]

After Execution:
  Input = {
      amount = 127
      owner = 2
  }
  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 = {
      __return = true
      block = {
        "number": 0,
        "timestamp": 0
      }
  }
  Address_Map = [
```

```
    {
      "key": 0,
      "value": {
        "contract_name": "ERC20BaseToken",
        "balance": 0,
        "contract": {
          "_totalSupply": 129,
          "_balances": [
            {
              "key": 0,
              "value": 1
            },
            {
              "key": 66,
              "value": 0
            },
            {
              "key": 4,
              "value": 0
            },
            {
              "key": 2,
              "value": 107
            },
            {
              "key": 128,
              "value": 0
            },
            {
              "key": "ALL_OTHERS",
              "value": 64
            }
          ],
          "_allowances": [
            {
              "key": 0,
              "value": [
                {
                  "key": 0,
                  "value": 0
                },
                {
                  "key": "ALL_OTHERS",
                  "value": 64
                }
              ]
            },
            {
              "key": 1,
              "value": [
                {
                  "key": 0,
                  "value": 1
                },
                {
                  "key": "ALL_OTHERS",
                  "value": 64
                }
```

```
              ]
            },
            {
              "key": "ALL_OTHERS",
              "value": [
                {
                  "key": "ALL_OTHERS",
                  "value": 148
                }
              ]
            }
          ],
          "_superOperators": [
            {
              "key": 0,
              "value": true
            },
            {
              "key": 2,
              "value": true
            },
            {
              "key": "ALL_OTHERS",
              "value": false
            }
          ],
          "_admin": 0
        }
      }
    },
    {
      "key": "ALL_OTHERS",
      "value": "EmptyAddress"
    }
  ]
```


Formal Verification Request 98

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

📅 02, Dec 2019
⏱ 8.17 ms

Line 160 in File ERC20BaseToken.sol

```
160   //@CTK NO_BUF_OVERFLOW
```

Line 177-180 in File ERC20BaseToken.sol

```
177   function burnFor(address owner, uint256 amount) external returns (bool) {
178       _burn(owner, amount);
179       return true;
180   }
```

✅ The code meets the specification.

Formal Verification Request 99

Method will not encounter an assertion failure.

02, Dec 2019
9.12 ms

Line 161 in File ERC20BaseToken.sol

```
161    //@CTK NO_ASF
```

Line 177-180 in File ERC20BaseToken.sol

```
177    function burnFor(address owner, uint256 amount) external returns (bool) {
178        _burn(owner, amount);
179        return true;
180    }
```

The code meets the specification.

Formal Verification Request 100

burnFor_require

02, Dec 2019
34.34 ms

Line 162-167 in File ERC20BaseToken.sol

```
162    /*@CTK burnFor_require
163      @tag assume_completion
164      @post amount > 0
165      @post _balances[owner] >= amount
166      @post (msg.sender != owner && !_superOperators[msg.sender]) -> _allowances[owner
             ][msg.sender] >= amount
167     */
```

Line 177-180 in File ERC20BaseToken.sol

```
177    function burnFor(address owner, uint256 amount) external returns (bool) {
178        _burn(owner, amount);
179        return true;
180    }
```

The code meets the specification.

Formal Verification Request 101

burnFor_change

02, Dec 2019
398.27 ms

Line 168-176 in File ERC20BaseToken.sol

```
168 /*@CTK burnFor_change
169   @tag assume_completion
170   @pre amount > 0
171   @pre _balances[owner] >= amount
172   @pre (msg.sender != owner && !_superOperators[msg.sender]) -> _allowances[owner][
        msg.sender] >= amount
173   @post (msg.sender != owner && !_superOperators[msg.sender] && _allowances[owner][
        msg.sender] != (2**256) - 1) -> __post._allowances[owner][msg.sender] ==
        _allowances[owner][msg.sender] - amount
174   @post __post._balances[owner] == _balances[owner] - amount
175   @post __post._totalSupply == _totalSupply - amount
176  */
```

Line 177-180 in File ERC20BaseToken.sol

```
177 function burnFor(address owner, uint256 amount) external returns (bool) {
178     _burn(owner, amount);
179     return true;
180 }
```

The code meets the specification.

Formal Verification Request 102

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

02, Dec 2019
154.88 ms

Line 243 in File ERC20BaseToken.sol

```
243 //@CTK NO_BUF_OVERFLOW
```

Line 258-268 in File ERC20BaseToken.sol

```
258 function addAllowanceIfNeeded(address owner, address spender, uint256 amountNeeded
      )
259     public
260     returns (bool success)
261 {
262     require(
263         msg.sender == owner || _superOperators[msg.sender],
264         "msg.sender != owner && !superOperator"
265     );
266     _addAllowanceIfNeeded(owner, spender, amountNeeded);
267     return true;
268 }
```

The code meets the specification.

Formal Verification Request 103

Method will not encounter an assertion failure.

02, Dec 2019
18.53 ms

Line 244 in File ERC20BaseToken.sol

```
244     //@CTK NO_ASF
```

Line 258-268 in File ERC20BaseToken.sol

```
258     function addAllowanceIfNeeded(address owner, address spender, uint256 amountNeeded
            )
259         public
260         returns (bool success)
261     {
262         require(
263             msg.sender == owner || _superOperators[msg.sender],
264             "msg.sender != owner && !superOperator"
265         );
266         _addAllowanceIfNeeded(owner, spender, amountNeeded);
267         return true;
268     }
```

The code meets the specification.

Formal Verification Request 104

addAllowanceIfNeeded_require

02, Dec 2019
23.47 ms

Line 245-250 in File ERC20BaseToken.sol

```
245     /*@CTK addAllowanceIfNeeded_require
246       @tag assume_completion
247       @post msg.sender == owner || _superOperators[msg.sender]
248       @post (amountNeeded > 0 && !_superOperators[spender] && _allowances[owner][
              spender] < amountNeeded) -> owner != address(0)
249       @post (amountNeeded > 0 && !_superOperators[spender] && _allowances[owner][
              spender] < amountNeeded) -> spender != address(0)
250      */
```

Line 258-268 in File ERC20BaseToken.sol

```
258     function addAllowanceIfNeeded(address owner, address spender, uint256 amountNeeded
            )
259         public
260         returns (bool success)
261     {
262         require(
263             msg.sender == owner || _superOperators[msg.sender],
264             "msg.sender != owner && !superOperator"
265         );
266         _addAllowanceIfNeeded(owner, spender, amountNeeded);
267         return true;
268     }
```

The code meets the specification.

Formal Verification Request 105

addAllowanceIfNeeded_change

02, Dec 2019
30.66 ms

Line 251-257 in File ERC20BaseToken.sol

```
251    /*@CTK addAllowanceIfNeeded_change
252      @tag assume_completion
253      @pre msg.sender == owner || _superOperators[msg.sender]
254      @pre (amountNeeded > 0 && !_superOperators[spender] && _allowances[owner][spender
         ] < amountNeeded) -> owner != address(0)
255      @pre (amountNeeded > 0 && !_superOperators[spender] && _allowances[owner][spender
         ] < amountNeeded) -> spender != address(0)
256      @post (amountNeeded > 0 && !_superOperators[spender] && _allowances[owner][
         spender] < amountNeeded) -> __post._allowances[owner][spender] ==
         amountNeeded
257     */
```

Line 258-268 in File ERC20BaseToken.sol

```
258    function addAllowanceIfNeeded(address owner, address spender, uint256 amountNeeded
       )
259        public
260        returns (bool success)
261    {
262        require(
263            msg.sender == owner || _superOperators[msg.sender],
264            "msg.sender != owner && !superOperator"
265        );
266        _addAllowanceIfNeeded(owner, spender, amountNeeded);
267        return true;
268    }
```

The code meets the specification.

Formal Verification Request 106

If method completes, integer overflow would not happen.

02, Dec 2019
20.98 ms

Line 270 in File ERC20BaseToken.sol

```
270    //@CTK NO_OVERFLOW
```

Line 284-293 in File ERC20BaseToken.sol

```
284    function _addAllowanceIfNeeded(address owner, address spender, uint256
       amountNeeded)
285        internal
286    {
287        if(amountNeeded > 0 && !isSuperOperator(spender)) {
288            uint256 currentAllowance = _allowances[owner][spender];
289            if(currentAllowance < amountNeeded) {
290                _approveFor(owner, spender, amountNeeded);
```

```
        }
    }
}
```

The code meets the specification.

Formal Verification Request 107

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

02, Dec 2019
22.73 ms

Line 271 in File ERC20BaseToken.sol

```
271 //@CTK NO_BUF_OVERFLOW
```

Line 284-293 in File ERC20BaseToken.sol

```
284 function _addAllowanceIfNeeded(address owner, address spender, uint256
        amountNeeded)
285     internal
286 {
287     if(amountNeeded > 0 && !isSuperOperator(spender)) {
288         uint256 currentAllowance = _allowances[owner][spender];
289         if(currentAllowance < amountNeeded) {
290             _approveFor(owner, spender, amountNeeded);
291         }
292     }
293 }
```

The code meets the specification.

Formal Verification Request 108

Method will not encounter an assertion failure.

02, Dec 2019
21.97 ms

Line 272 in File ERC20BaseToken.sol

```
272 //@CTK NO_ASF
```

Line 284-293 in File ERC20BaseToken.sol

```
284 function _addAllowanceIfNeeded(address owner, address spender, uint256
        amountNeeded)
285     internal
286 {
287     if(amountNeeded > 0 && !isSuperOperator(spender)) {
288         uint256 currentAllowance = _allowances[owner][spender];
289         if(currentAllowance < amountNeeded) {
290             _approveFor(owner, spender, amountNeeded);
291         }
292     }
293 }
```

The code meets the specification.

Formal Verification Request 109

__addAllowanceIfNeeded__require

02, Dec 2019
13.04 ms

Line 273-277 in File ERC20BaseToken.sol

```
273   /*@CTK _addAllowanceIfNeeded_require
274     @tag assume_completion
275     @post (amountNeeded > 0 && !_superOperators[spender] && _allowances[owner][
          spender] < amountNeeded) -> owner != address(0)
276     @post (amountNeeded > 0 && !_superOperators[spender] && _allowances[owner][
          spender] < amountNeeded) -> spender != address(0)
277    */
```

Line 284-293 in File ERC20BaseToken.sol

```
284   function _addAllowanceIfNeeded(address owner, address spender, uint256
        amountNeeded)
285       internal
286   {
287       if(amountNeeded > 0 && !isSuperOperator(spender)) {
288           uint256 currentAllowance = _allowances[owner][spender];
289           if(currentAllowance < amountNeeded) {
290               _approveFor(owner, spender, amountNeeded);
291           }
292       }
293   }
```

The code meets the specification.

Formal Verification Request 110

__addAllowanceIfNeeded__change

02, Dec 2019
41.05 ms

Line 278-283 in File ERC20BaseToken.sol

```
278   /*@CTK _addAllowanceIfNeeded_change
279     @tag assume_completion
280     @pre (amountNeeded > 0 && !_superOperators[spender] && _allowances[owner][spender
          ] < amountNeeded) -> owner != address(0)
281     @pre (amountNeeded > 0 && !_superOperators[spender] && _allowances[owner][spender
          ] < amountNeeded) -> spender != address(0)
282     @post (amountNeeded > 0 && !_superOperators[spender] && _allowances[owner][
          spender] < amountNeeded) -> __post._allowances[owner][spender] ==
          amountNeeded
283    */
```

Line 284-293 in File ERC20BaseToken.sol

```
    function _addAllowanceIfNeeded(address owner, address spender, uint256
        amountNeeded)
        internal
    {
        if(amountNeeded > 0 && !isSuperOperator(spender)) {
            uint256 currentAllowance = _allowances[owner][spender];
            if(currentAllowance < amountNeeded) {
                _approveFor(owner, spender, amountNeeded);
            }
        }
    }
```

✅ The code meets the specification.

Formal Verification Request 111

If method completes, integer overflow would not happen.

📅 02, Dec 2019
⏱ 0.69 ms

Line 295 in File ERC20BaseToken.sol

```
    //@CTK NO_OVERFLOW
```

Line 309-318 in File ERC20BaseToken.sol

```
    function _approveFor(address owner, address spender, uint256 amount)
        internal
    {
        require(
            owner != address(0) && spender != address(0),
            "Cannot approve with 0x0"
        );
        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }
```

✅ The code meets the specification.

Formal Verification Request 112

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

📅 02, Dec 2019
⏱ 0.6 ms

Line 296 in File ERC20BaseToken.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 309-318 in File ERC20BaseToken.sol

```
    function _approveFor(address owner, address spender, uint256 amount)
        internal
    {
        require(
```

```
        owner != address(0) && spender != address(0),
        "Cannot approve with 0x0"
    );
    _allowances[owner][spender] = amount;
    emit Approval(owner, spender, amount);
}
```

The code meets the specification.

Formal Verification Request 113

Method will not encounter an assertion failure.

02, Dec 2019
0.51 ms

Line 297 in File ERC20BaseToken.sol

```
297 //@CTK NO_ASF
```

Line 309-318 in File ERC20BaseToken.sol

```
309 function _approveFor(address owner, address spender, uint256 amount)
310     internal
311 {
312     require(
313         owner != address(0) && spender != address(0),
314         "Cannot approve with 0x0"
315     );
316     _allowances[owner][spender] = amount;
317     emit Approval(owner, spender, amount);
318 }
```

The code meets the specification.

Formal Verification Request 114

_approveFor_require

02, Dec 2019
1.31 ms

Line 298-302 in File ERC20BaseToken.sol

```
298 /*@CTK _approveFor_require
299   @tag assume_completion
300   @post owner != address(0)
301   @post spender != address(0)
302  */
```

Line 309-318 in File ERC20BaseToken.sol

```
309 function _approveFor(address owner, address spender, uint256 amount)
310     internal
311 {
312     require(
313         owner != address(0) && spender != address(0),
```

```
            "Cannot approve with 0x0"
        );
        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }
```

The code meets the specification.

Formal Verification Request 115

_approveFor_change

02, Dec 2019
1.74 ms

Line 303-308 in File ERC20BaseToken.sol

```
303     /*@CTK _approveFor_change
304       @tag assume_completion
305       @pre owner != address(0)
306       @pre spender != address(0)
307       @post __post._allowances[owner][spender] == amount
308      */
```

Line 309-318 in File ERC20BaseToken.sol

```
309     function _approveFor(address owner, address spender, uint256 amount)
310         internal
311     {
312         require(
313             owner != address(0) && spender != address(0),
314             "Cannot approve with 0x0"
315         );
316         _allowances[owner][spender] = amount;
317         emit Approval(owner, spender, amount);
318     }
```

The code meets the specification.

Formal Verification Request 116

If method completes, integer overflow would not happen.

02, Dec 2019
23.5 ms

Line 320 in File ERC20BaseToken.sol

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

Line 336-343 in File ERC20BaseToken.sol

```
336     function _transfer(address from, address to, uint256 amount) internal {
337         require(to != address(0), "Cannot send to 0x0");
338         uint256 currentBalance = _balances[from];
339         require(currentBalance >= amount, "not enough fund");
340         _balances[from] = currentBalance - amount;
```

```
        _balances[to] += amount;
        emit Transfer(from, to, amount);
    }
```

This code violates the specification.

```
Counter Example:
Before Execution:
    Input = {
        amount = 192
        from = 0
        to = 1
    }
    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 = {
        block = {
          "number": 0,
          "timestamp": 0
        }
    }
    Address_Map = [
      {
        "key": 0,
        "value": {
          "contract_name": "ERC20BaseToken",
          "balance": 0,
          "contract": {
            "_totalSupply": 0,
            "_balances": [
              {
                "key": 0,
                "value": 192
              },
              {
                "key": 1,
                "value": 64
              },
              {
                "key": "ALL_OTHERS",
                "value": 255
              }
            ],
            "_allowances": [
              {
                "key": "ALL_OTHERS",
                "value": [
                  {
```

```
                  "key": "ALL_OTHERS",
                  "value": 255
                }
              ]
            }
          ],
          "_superOperators": [
            {
              "key": "ALL_OTHERS",
              "value": false
            }
          ],
          "_admin": 0
        }
      }
    },
    {
      "key": "ALL_OTHERS",
      "value": "EmptyAddress"
    }
  ]

After Execution:
  Input = {
      amount = 192
      from = 0
      to = 1
  }
  This = 0
  Internal = {
      __has_assertion_failure = false
      __has_buf_overflow = false
      __has_overflow = true
      __has_returned = false
      __reverted = false
      msg = {
        "gas": 0,
        "sender": 0,
        "value": 0
      }
  }
  Other = {
      block = {
        "number": 0,
        "timestamp": 0
      }
      currentBalance = 192
  }
  Address_Map = [
    {
      "key": 0,
      "value": {
        "contract_name": "ERC20BaseToken",
        "balance": 0,
        "contract": {
          "_totalSupply": 0,
          "_balances": [
            {
```

```
              "key": 0,
              "value": 0
            },
            {
              "key": 1,
              "value": 0
            },
            {
              "key": "ALL_OTHERS",
              "value": 255
            }
          ],
          "_allowances": [
            {
              "key": "ALL_OTHERS",
              "value": [
                {
                  "key": "ALL_OTHERS",
                  "value": 255
                }
              ]
            }
          ],
          "_superOperators": [
            {
              "key": "ALL_OTHERS",
              "value": false
            }
          ],
          "_admin": 0
        }
      }
    },
    {
      "key": "ALL_OTHERS",
      "value": "EmptyAddress"
    }
  ]
```


Formal Verification Request 117

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

📅 02, Dec 2019
⏱ 0.82 ms

Line 321 in File ERC20BaseToken.sol

```
321   //@CTK NO_BUF_OVERFLOW
```

Line 336-343 in File ERC20BaseToken.sol

```
336   function _transfer(address from, address to, uint256 amount) internal {
337       require(to != address(0), "Cannot send to 0x0");
338       uint256 currentBalance = _balances[from];
339       require(currentBalance >= amount, "not enough fund");
340       _balances[from] = currentBalance - amount;
341       _balances[to] += amount;
```

```
        emit Transfer(from, to, amount);
    }
```

The code meets the specification.

Formal Verification Request 118

Method will not encounter an assertion failure.

02, Dec 2019
0.63 ms

Line 322 in File ERC20BaseToken.sol

```
    //@CTK NO_ASF
```

Line 336-343 in File ERC20BaseToken.sol

```
    function _transfer(address from, address to, uint256 amount) internal {
        require(to != address(0), "Cannot send to 0x0");
        uint256 currentBalance = _balances[from];
        require(currentBalance >= amount, "not enough fund");
        _balances[from] = currentBalance - amount;
        _balances[to] += amount;
        emit Transfer(from, to, amount);
    }
```

The code meets the specification.

Formal Verification Request 119

__transfer__require

02, Dec 2019
11.0 ms

Line 323-327 in File ERC20BaseToken.sol

```
    /*@CTK _transfer_require
      @tag assume_completion
      @post to != address(0)
      @post _balances[from] >= amount
     */
```

Line 336-343 in File ERC20BaseToken.sol

```
    function _transfer(address from, address to, uint256 amount) internal {
        require(to != address(0), "Cannot send to 0x0");
        uint256 currentBalance = _balances[from];
        require(currentBalance >= amount, "not enough fund");
        _balances[from] = currentBalance - amount;
        _balances[to] += amount;
        emit Transfer(from, to, amount);
    }
```

The code meets the specification.

Formal Verification Request 120

__transfer__change

02, Dec 2019
22.61 ms

Line 328-335 in File ERC20BaseToken.sol

```
328    /*@CTK _transfer_change
329      @tag assume_completion
330      @pre to != address(0)
331      @pre _balances[from] >= amount
332      @pre from != to
333      @post __post._balances[from] == _balances[from] - amount
334      @post __post._balances[to] == _balances[to] + amount
335     */
```

Line 336-343 in File ERC20BaseToken.sol

```
336    function _transfer(address from, address to, uint256 amount) internal {
337        require(to != address(0), "Cannot send to 0x0");
338        uint256 currentBalance = _balances[from];
339        require(currentBalance >= amount, "not enough fund");
340        _balances[from] = currentBalance - amount;
341        _balances[to] += amount;
342        emit Transfer(from, to, amount);
343    }
```

The code meets the specification.

Formal Verification Request 121

If method completes, integer overflow would not happen.

02, Dec 2019
54.82 ms

Line 345 in File ERC20BaseToken.sol

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

Line 362-371 in File ERC20BaseToken.sol

```
362    function _mint(address to, uint256 amount) internal {
363        require(to != address(0), "Cannot mint to 0x0");
364        require(amount > 0, "cannot mint 0 tokens");
365        uint256 currentTotalSupply = _totalSupply;
366        uint256 newTotalSupply = currentTotalSupply + amount;
367        require(newTotalSupply > currentTotalSupply, "overflow");
368        _totalSupply = newTotalSupply;
369        _balances[to] += amount;
370        emit Transfer(address(0), to, amount);
371    }
```

This code violates the specification.

```
1 Counter Example:
2 Before Execution:
```

```
Input = {
    amount = 224
    to = 2
}
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 = {
    block = {
      "number": 0,
      "timestamp": 0
    }
}
Address_Map = [
  {
    "key": 0,
    "value": {
      "contract_name": "ERC20BaseToken",
      "balance": 0,
      "contract": {
        "_totalSupply": 0,
        "_balances": [
          {
            "key": 2,
            "value": 34
          },
          {
            "key": "ALL_OTHERS",
            "value": 2
          }
        ],
        "_allowances": [
          {
            "key": "ALL_OTHERS",
            "value": [
              {
                "key": "ALL_OTHERS",
                "value": 2
              }
            ]
          }
        ],
        "_superOperators": [
          {
            "key": "ALL_OTHERS",
            "value": false
          }
        ],
```

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

After Execution:
  Input = {
      amount = 224
      to = 2
  }
  This = 0
  Internal = {
      __has_assertion_failure = false
      __has_buf_overflow = false
      __has_overflow = true
      __has_returned = false
      __reverted = false
      msg = {
        "gas": 0,
        "sender": 0,
        "value": 0
      }
  }
  Other = {
      block = {
        "number": 0,
        "timestamp": 0
      }
      currentTotalSupply = 0
      newTotalSupply = 224
  }
  Address_Map = [
    {
      "key": 0,
      "value": {
        "contract_name": "ERC20BaseToken",
        "balance": 0,
        "contract": {
          "_totalSupply": 224,
          "_balances": [
            {
              "key": "ALL_OTHERS",
              "value": 2
            }
          ],
          "_allowances": [
            {
              "key": "ALL_OTHERS",
              "value": [
                {
                  "key": "ALL_OTHERS",
                  "value": 2
                }
```

```
              ]
            }
          ],
          "_superOperators": [
            {
              "key": "ALL_OTHERS",
              "value": false
            }
          ],
          "_admin": 0
        }
      }
    },
    {
      "key": "ALL_OTHERS",
      "value": "EmptyAddress"
    }
  ]
```

Formal Verification Request 122

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

📅 02, Dec 2019
⏱ 4.89 ms

Line 346 in File ERC20BaseToken.sol

```
346   //@CTK NO_BUF_OVERFLOW
```

Line 362-371 in File ERC20BaseToken.sol

```
362   function _mint(address to, uint256 amount) internal {
363       require(to != address(0), "Cannot mint to 0x0");
364       require(amount > 0, "cannot mint 0 tokens");
365       uint256 currentTotalSupply = _totalSupply;
366       uint256 newTotalSupply = currentTotalSupply + amount;
367       require(newTotalSupply > currentTotalSupply, "overflow");
368       _totalSupply = newTotalSupply;
369       _balances[to] += amount;
370       emit Transfer(address(0), to, amount);
371   }
```

✅ The code meets the specification.

Formal Verification Request 123

Method will not encounter an assertion failure.

📅 02, Dec 2019
⏱ 7.04 ms

Line 347 in File ERC20BaseToken.sol

```
347   //@CTK NO_ASF
```

Line 362-371 in File ERC20BaseToken.sol

```
362    function _mint(address to, uint256 amount) internal {
363        require(to != address(0), "Cannot mint to 0x0");
364        require(amount > 0, "cannot mint 0 tokens");
365        uint256 currentTotalSupply = _totalSupply;
366        uint256 newTotalSupply = currentTotalSupply + amount;
367        require(newTotalSupply > currentTotalSupply, "overflow");
368        _totalSupply = newTotalSupply;
369        _balances[to] += amount;
370        emit Transfer(address(0), to, amount);
371    }
```

✅ The code meets the specification.

Formal Verification Request 124

_mint_require_no_overflow

📅 02, Dec 2019
⏱ 22.79 ms

Line 348-353 in File ERC20BaseToken.sol

```
348    /*@CTK _mint_require_no_overflow
349      @tag assume_completion
350      @post to != address(0)
351      @post amount > 0
352      @post _totalSupply + amount > _totalSupply
353     */
```

Line 362-371 in File ERC20BaseToken.sol

```
362    function _mint(address to, uint256 amount) internal {
363        require(to != address(0), "Cannot mint to 0x0");
364        require(amount > 0, "cannot mint 0 tokens");
365        uint256 currentTotalSupply = _totalSupply;
366        uint256 newTotalSupply = currentTotalSupply + amount;
367        require(newTotalSupply > currentTotalSupply, "overflow");
368        _totalSupply = newTotalSupply;
369        _balances[to] += amount;
370        emit Transfer(address(0), to, amount);
371    }
```

✅ The code meets the specification.

Formal Verification Request 125

_mint_change

📅 02, Dec 2019
⏱ 78.97 ms

Line 354-361 in File ERC20BaseToken.sol

```
354    /*@CTK _mint_change
355      @tag assume_completion
356      @pre to != address(0)
```

```
357     @pre amount > 0
358     @pre _totalSupply + amount > _totalSupply
359     @post __post._totalSupply == _totalSupply + amount
360     @post __post._balances[to] == _balances[to] + amount
361    */
```

Line 362-371 in File ERC20BaseToken.sol

```
362    function _mint(address to, uint256 amount) internal {
363        require(to != address(0), "Cannot mint to 0x0");
364        require(amount > 0, "cannot mint 0 tokens");
365        uint256 currentTotalSupply = _totalSupply;
366        uint256 newTotalSupply = currentTotalSupply + amount;
367        require(newTotalSupply > currentTotalSupply, "overflow");
368        _totalSupply = newTotalSupply;
369        _balances[to] += amount;
370        emit Transfer(address(0), to, amount);
371    }
```

✅ The code meets the specification.

Formal Verification Request 126

If method completes, integer overflow would not happen.

📅 02, Dec 2019
⏱ 44.89 ms

Line 373 in File ERC20BaseToken.sol

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

Line 391-410 in File ERC20BaseToken.sol

```
391    function _burn(address from, uint256 amount) internal {
392        require(amount > 0, "cannot burn 0 tokens");
393        if (msg.sender != from && !_superOperators[msg.sender]) {
394            uint256 currentAllowance = _allowances[from][msg.sender];
395            require(
396                currentAllowance >= amount,
397                "Not enough funds allowed"
398            );
399            if (currentAllowance != (2**256) - 1) {
400                // save gas when allowance is maximal by not reducing it (see https://
                       github.com/ethereum/EIPs/issues/717)
401                _allowances[from][msg.sender] = currentAllowance - amount;
402            }
403        }
404
405        uint256 currentBalance = _balances[from];
406        require(currentBalance >= amount, "Not enough funds");
407        _balances[from] = currentBalance - amount;
408        _totalSupply -= amount;
409        emit Transfer(from, address(0), amount);
410    }
```

❌ This code violates the specification.

```
Counter Example:
Before Execution:
    Input = {
        amount = 124
        from = 128
    }
    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 = {
        block = {
          "number": 0,
          "timestamp": 0
        }
    }
    Address_Map = [
      {
        "key": 0,
        "value": {
          "contract_name": "ERC20BaseToken",
          "balance": 0,
          "contract": {
            "_totalSupply": 0,
            "_balances": [
              {
                "key": 0,
                "value": 1
              },
              {
                "key": 128,
                "value": 208
              },
              {
                "key": "ALL_OTHERS",
                "value": 131
              }
            ],
            "_allowances": [
              {
                "key": 0,
                "value": [
                  {
                    "key": 0,
                    "value": 0
                  },
                  {
                    "key": "ALL_OTHERS",
                    "value": 131
```

```
                }
              ]
            },
            {
              "key": 128,
              "value": [
                {
                  "key": 0,
                  "value": 255
                },
                {
                  "key": "ALL_OTHERS",
                  "value": 124
                }
              ]
            },
            {
              "key": "ALL_OTHERS",
              "value": [
                {
                  "key": "ALL_OTHERS",
                  "value": 131
                }
              ]
            }
          ],
          "_superOperators": [
            {
              "key": "ALL_OTHERS",
              "value": false
            }
          ],
          "_admin": 0
        }
      }
    },
    {
      "key": "ALL_OTHERS",
      "value": "EmptyAddress"
    }
  ]

After Execution:
  Input = {
      amount = 124
      from = 128
  }
  This = 0
  Internal = {
      __has_assertion_failure = false
      __has_buf_overflow = false
      __has_overflow = true
      __has_returned = false
      __reverted = false
      msg = {
        "gas": 0,
        "sender": 0,
        "value": 0
```

```
        }
    }
    Other = {
        block = {
          "number": 0,
          "timestamp": 0
        }
        currentBalance = 208
    }
    Address_Map = [
      {
        "key": 0,
        "value": {
          "contract_name": "ERC20BaseToken",
          "balance": 0,
          "contract": {
            "_totalSupply": 132,
            "_balances": [
              {
                "key": 0,
                "value": 1
              },
              {
                "key": 128,
                "value": 84
              },
              {
                "key": "ALL_OTHERS",
                "value": 131
              }
            ],
            "_allowances": [
              {
                "key": 0,
                "value": [
                  {
                    "key": 0,
                    "value": 0
                  },
                  {
                    "key": "ALL_OTHERS",
                    "value": 131
                  }
                ]
              },
              {
                "key": 128,
                "value": [
                  {
                    "key": 0,
                    "value": 255
                  },
                  {
                    "key": "ALL_OTHERS",
                    "value": 124
                  }
                ]
              },
```

```
            {
              "key": "ALL_OTHERS",
              "value": [
                {
                  "key": "ALL_OTHERS",
                  "value": 131
                }
              ]
            }
          ],
          "_superOperators": [
            {
              "key": "ALL_OTHERS",
              "value": false
            }
          ],
          "_admin": 0
        }
      }
    },
    {
      "key": "ALL_OTHERS",
      "value": "EmptyAddress"
    }
  ]
```

Formal Verification Request 127

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

📅 02, Dec 2019
⏱ 7.46 ms

Line 374 in File ERC20BaseToken.sol

```
374   //@CTK NO_BUF_OVERFLOW
```

Line 391-410 in File ERC20BaseToken.sol

```
391   function _burn(address from, uint256 amount) internal {
392       require(amount > 0, "cannot burn 0 tokens");
393       if (msg.sender != from && !_superOperators[msg.sender]) {
394           uint256 currentAllowance = _allowances[from][msg.sender];
395           require(
396               currentAllowance >= amount,
397               "Not enough funds allowed"
398           );
399           if (currentAllowance != (2**256) - 1) {
400               // save gas when allowance is maximal by not reducing it (see https://
                      github.com/ethereum/EIPs/issues/717)
401               _allowances[from][msg.sender] = currentAllowance - amount;
402           }
403       }
404
405       uint256 currentBalance = _balances[from];
406       require(currentBalance >= amount, "Not enough funds");
407       _balances[from] = currentBalance - amount;
408       _totalSupply -= amount;
```

```
        emit Transfer(from, address(0), amount);
    }
```

The code meets the specification.

Formal Verification Request 128

Method will not encounter an assertion failure.

02, Dec 2019
7.56 ms

Line 375 in File ERC20BaseToken.sol

```
    //@CTK NO_ASF
```

Line 391-410 in File ERC20BaseToken.sol

```
    function _burn(address from, uint256 amount) internal {
        require(amount > 0, "cannot burn 0 tokens");
        if (msg.sender != from && !_superOperators[msg.sender]) {
            uint256 currentAllowance = _allowances[from][msg.sender];
            require(
                currentAllowance >= amount,
                "Not enough funds allowed"
            );
            if (currentAllowance != (2**256) - 1) {
                // save gas when allowance is maximal by not reducing it (see https://
                    github.com/ethereum/EIPs/issues/717)
                _allowances[from][msg.sender] = currentAllowance - amount;
            }
        }

        uint256 currentBalance = _balances[from];
        require(currentBalance >= amount, "Not enough funds");
        _balances[from] = currentBalance - amount;
        _totalSupply -= amount;
        emit Transfer(from, address(0), amount);
    }
```

The code meets the specification.

Formal Verification Request 129

__burn__require__identity

02, Dec 2019
46.14 ms

Line 376-381 in File ERC20BaseToken.sol

```
    /*@CTK _burn_require_identity
      @tag assume_completion
      @post amount > 0
      @post _balances[from] >= amount
```

```
380     @post (msg.sender != from && !_superOperators[msg.sender]) -> _allowances[from][
            msg.sender] >= amount
381    */
```

Line 391-410 in File ERC20BaseToken.sol

```
391    function _burn(address from, uint256 amount) internal {
392        require(amount > 0, "cannot burn 0 tokens");
393        if (msg.sender != from && !_superOperators[msg.sender]) {
394            uint256 currentAllowance = _allowances[from][msg.sender];
395            require(
396                currentAllowance >= amount,
397                "Not enough funds allowed"
398            );
399            if (currentAllowance != (2**256) - 1) {
400                // save gas when allowance is maximal by not reducing it (see https://
                       github.com/ethereum/EIPs/issues/717)
401                _allowances[from][msg.sender] = currentAllowance - amount;
402            }
403        }
404
405        uint256 currentBalance = _balances[from];
406        require(currentBalance >= amount, "Not enough funds");
407        _balances[from] = currentBalance - amount;
408        _totalSupply -= amount;
409        emit Transfer(from, address(0), amount);
410    }
```

✅ The code meets the specification.

Formal Verification Request 130

__burn__change

📅 02, Dec 2019
⏱ 409.01 ms

Line 382-390 in File ERC20BaseToken.sol

```
382    /*@CTK _burn_change
383      @tag assume_completion
384      @pre amount > 0
385      @pre _balances[from] >= amount
386      @pre (msg.sender != from && !_superOperators[msg.sender]) -> _allowances[from][
             msg.sender] >= amount
387      @post (msg.sender != from && !_superOperators[msg.sender] && _allowances[from][
             msg.sender] != (2**256) - 1) -> __post._allowances[from][msg.sender] ==
             _allowances[from][msg.sender] - amount
388      @post __post._balances[from] == _balances[from] - amount
389      @post __post._totalSupply == _totalSupply - amount
390     */
```

Line 391-410 in File ERC20BaseToken.sol

```
391    function _burn(address from, uint256 amount) internal {
392        require(amount > 0, "cannot burn 0 tokens");
393        if (msg.sender != from && !_superOperators[msg.sender]) {
394            uint256 currentAllowance = _allowances[from][msg.sender];
```

```
        require(
            currentAllowance >= amount,
            "Not enough funds allowed"
        );
        if (currentAllowance != (2**256) - 1) {
            // save gas when allowance is maximal by not reducing it (see https://
                github.com/ethereum/EIPs/issues/717)
            _allowances[from][msg.sender] = currentAllowance - amount;
        }
    }

    uint256 currentBalance = _balances[from];
    require(currentBalance >= amount, "Not enough funds");
    _balances[from] = currentBalance - amount;
    _totalSupply -= amount;
    emit Transfer(from, address(0), amount);
}
```

The code meets the specification.

Formal Verification Request 131

If method completes, integer overflow would not happen.

02, Dec 2019
41.84 ms

Line 10 in File ERC20BasicApproveExtension.sol

```
10  //@CTK NO_OVERFLOW
```

Line 13-30 in File ERC20BasicApproveExtension.sol

```
13  function approveAndCall(
14      address target,
15      uint256 amount,
16      bytes calldata data
17  ) external payable returns (bytes memory) {
18      require(
19          doFirstParamEqualsAddress(data, msg.sender),
20          "first param != sender"
21      );
22      _approveFor(msg.sender, target, amount);
23
24      // solium-disable-next-line security/no-call-value
25      (bool success, bytes memory returnData) = target.call.value(msg.value)(data);
26      require(success, string(returnData));
27      return returnData;
28  }
```

The code meets the specification.

Formal Verification Request 132

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

02, Dec 2019

0.5 ms

Line 11 in File ERC20BasicApproveExtension.sol

```
11  //@CTK NO_BUF_OVERFLOW
```

Line 13-30 in File ERC20BasicApproveExtension.sol

```
13  function approveAndCall(
14      address target,
15      uint256 amount,
16      bytes calldata data
17  ) external payable returns (bytes memory) {
18      require(
19          doFirstParamEqualsAddress(data, msg.sender),
20          "first param != sender"
21      );
22      _approveFor(msg.sender, target, amount);
23
24      // solium-disable-next-line security/no-call-value
25      (bool success, bytes memory returnData) = target.call.value(msg.value)(data);
26      require(success, string(returnData));
27      return returnData;
28  }
```

The code meets the specification.

Formal Verification Request 133

Method will not encounter an assertion failure.

02, Dec 2019
0.49 ms

Line 12 in File ERC20BasicApproveExtension.sol

```
12  //@CTK NO_ASF
```

Line 13-30 in File ERC20BasicApproveExtension.sol

```
13  function approveAndCall(
14      address target,
15      uint256 amount,
16      bytes calldata data
17  ) external payable returns (bytes memory) {
18      require(
19          doFirstParamEqualsAddress(data, msg.sender),
20          "first param != sender"
21      );
22      _approveFor(msg.sender, target, amount);
23
24      // solium-disable-next-line security/no-call-value
25      (bool success, bytes memory returnData) = target.call.value(msg.value)(data);
26      require(success, string(returnData));
27      return returnData;
28  }
```

The code meets the specification.

Formal Verification Request 134

If method completes, integer overflow would not happen.

02, Dec 2019
32.41 ms

Line 37 in File ERC20BasicApproveExtension.sol

```
37  //@CTK NO_OVERFLOW
```

Line 40-63 in File ERC20BasicApproveExtension.sol

```
40  function paidCall(
41      address target,
42      uint256 amount,
43      bytes calldata data
44  ) external payable returns (bytes memory) {
45      require(
46          doFirstParamEqualsAddress(data, msg.sender),
47          "first param != sender"
48      );
49
50      if (amount > 0) {
51          _addAllowanceIfNeeded(msg.sender, target, amount);
52      }
53
54      // solium-disable-next-line security/no-call-value
55      (bool success, bytes memory returnData) = target.call.value(msg.value)(data);
56      require(success, string(returnData));
57
58      return returnData;
59  }
```

The code meets the specification.

Formal Verification Request 135

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

02, Dec 2019
0.73 ms

Line 38 in File ERC20BasicApproveExtension.sol

```
38  //@CTK NO_BUF_OVERFLOW
```

Line 40-63 in File ERC20BasicApproveExtension.sol

```
40  function paidCall(
41      address target,
42      uint256 amount,
43      bytes calldata data
44  ) external payable returns (bytes memory) {
45      require(
46          doFirstParamEqualsAddress(data, msg.sender),
47          "first param != sender"
48      );
49
```

```
        if (amount > 0) {
            _addAllowanceIfNeeded(msg.sender, target, amount);
        }

        // solium-disable-next-line security/no-call-value
        (bool success, bytes memory returnData) = target.call.value(msg.value)(data);
        require(success, string(returnData));

        return returnData;
    }
```

✅ The code meets the specification.

Formal Verification Request 136

Method will not encounter an assertion failure.

📅 02, Dec 2019
⏱ 0.57 ms

Line 39 in File ERC20BasicApproveExtension.sol

```
39    //@CTK NO_ASF
```

Line 40-63 in File ERC20BasicApproveExtension.sol

```
40    function paidCall(
41        address target,
42        uint256 amount,
43        bytes calldata data
44    ) external payable returns (bytes memory) {
45        require(
46            doFirstParamEqualsAddress(data, msg.sender),
47            "first param != sender"
48        );
49
50        if (amount > 0) {
51            _addAllowanceIfNeeded(msg.sender, target, amount);
52        }
53
54        // solium-disable-next-line security/no-call-value
55        (bool success, bytes memory returnData) = target.call.value(msg.value)(data);
56        require(success, string(returnData));
57
58        return returnData;
59    }
```

✅ The code meets the specification.

Formal Verification Request 137

If method completes, integer overflow would not happen.

📅 02, Dec 2019
⏱ 0.42 ms

Line 70 in File ERC20BasicApproveExtension.sol

```
70    //@CTK NO_OVERFLOW
```

Line 77-94 in File ERC20BasicApproveExtension.sol

```
77    function doFirstParamEqualsAddress(bytes memory data, address _address)
78        internal
79        pure
80        returns (bool)
81    {
82        if (data.length < (36 + 32)) {
83            return false;
84        }
85        uint256 value;
86        /*@CTK "Load 32 bytes from mem[data+36]"
87          @var uint value
88          @post value == uint256(_address)
89         */
90        assembly {
91            value := mload(add(data, 36))
92        }
93        return value == uint256(_address);
94    }
```

✅ The code meets the specification.

Formal Verification Request 138

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

📅 02, Dec 2019
⏱ 0.38 ms

Line 71 in File ERC20BasicApproveExtension.sol

```
71    //@CTK NO_BUF_OVERFLOW
```

Line 77-94 in File ERC20BasicApproveExtension.sol

```
77    function doFirstParamEqualsAddress(bytes memory data, address _address)
78        internal
79        pure
80        returns (bool)
81    {
82        if (data.length < (36 + 32)) {
83            return false;
84        }
85        uint256 value;
86        /*@CTK "Load 32 bytes from mem[data+36]"
87          @var uint value
88          @post value == uint256(_address)
89         */
90        assembly {
91            value := mload(add(data, 36))
92        }
93        return value == uint256(_address);
94    }
```

✅ The code meets the specification.

Formal Verification Request 139

Method will not encounter an assertion failure.

02, Dec 2019
0.37 ms

Line 72 in File ERC20BasicApproveExtension.sol

```
72    //@CTK NO_ASF
```

Line 77-94 in File ERC20BasicApproveExtension.sol

```
77    function doFirstParamEqualsAddress(bytes memory data, address _address)
78        internal
79        pure
80        returns (bool)
81    {
82        if (data.length < (36 + 32)) {
83            return false;
84        }
85        uint256 value;
86        /*@CTK "Load 32 bytes from mem[data+36]"
87          @var uint value
88          @post value == uint256(_address)
89         */
90        assembly {
91            value := mload(add(data, 36))
92        }
93        return value == uint256(_address);
94    }
```

The code meets the specification.

Formal Verification Request 140

doFirstParamEqualsAddress_require

02, Dec 2019
1.46 ms

Line 73-76 in File ERC20BasicApproveExtension.sol

```
73    /*@CTK doFirstParamEqualsAddress_require
74      @tag assume_completion
75      @post (data.length < (36 + 32)) -> !__return
76     */
```

Line 77-94 in File ERC20BasicApproveExtension.sol

```
77    function doFirstParamEqualsAddress(bytes memory data, address _address)
78        internal
79        pure
80        returns (bool)
81    {
82        if (data.length < (36 + 32)) {
83            return false;
84        }
85        uint256 value;
```

```
    /*@CTK "Load 32 bytes from mem[data+36]"
      @var uint value
      @post value == uint256(_address)
     */
    assembly {
        value := mload(add(data, 36))
    }
    return value == uint256(_address);
}
```

The code meets the specification.

Source Code with CertiK Labels

File ERC20ExecuteExtension.sol

```
pragma solidity 0.5.9;


contract ERC20ExecuteExtension {

    /// @dev _executionAdmin != _admin so that this super power can be disabled independently
    address internal _executionAdmin;

    event ExecutionAdminAdminChanged(address oldAdmin, address newAdmin);

    /// @notice give the address responsible for adding execution rights.
    /// @return address of the execution administrator.
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK getExecutionAdmin
      @post __return == _executionAdmin
     */
    function getExecutionAdmin() external view returns (address) {
        return _executionAdmin;
    }

    /// @notice change the execution adminstrator to be `newAdmin`.
    /// @param newAdmin address of the new administrator.
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK changeExecutionAdmin_require
      @tag assume_completion
      @post msg.sender == _executionAdmin
     */
    /*@CTK changeExecutionAdmin_change
      @tag assume_completion
      @pre msg.sender == _executionAdmin
      @post __post._executionAdmin == newAdmin
     */
    function changeExecutionAdmin(address newAdmin) external {
        require(msg.sender == _executionAdmin, "only executionAdmin can change executionAdmin");
        emit ExecutionAdminAdminChanged(_executionAdmin, newAdmin);
        _executionAdmin = newAdmin;
    }

    mapping(address => bool) internal _executionOperators;
    event ExecutionOperator(address executionOperator, bool enabled);

    /// @notice set `executionOperator` as executionOperator: `enabled`.
    /// @param executionOperator address that will be given/removed executionOperator right.
    /// @param enabled set whether the executionOperator is enabled or disabled.
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
```

```
    /*@CTK setExecutionOperator_require
      @tag assume_completion
      @post msg.sender == _executionAdmin
     */
    /*@CTK setExecutionOperator_change
      @tag assume_completion
      @pre msg.sender == _executionAdmin
      @post __post._executionOperators[executionOperator] == enabled
     */
    function setExecutionOperator(address executionOperator, bool enabled) external {
        require(
            msg.sender == _executionAdmin,
            "only execution admin is allowed to add execution operators"
        );
        _executionOperators[executionOperator] = enabled;
        emit ExecutionOperator(executionOperator, enabled);
    }

    /// @notice check whether address `who` is given executionOperator rights.
    /// @param who The address to query.
    /// @return whether the address has executionOperator rights.
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK isExecutionOperator
      @post __return == _executionOperators[who]
     */
    function isExecutionOperator(address who) public view returns (bool) {
        return _executionOperators[who];
    }

    /// @notice execute on behalf of the contract.
    /// @param to destination address fo the call.
    /// @param gasLimit exact amount of gas to be passed to the call.
    /// @param data the bytes sent to the destination address.
    /// @return success whether the execution was successful.
    /// @return returnData data resulting from the execution.
    //@FIXME NO_OVERFLOW
    //@FIXME NO_BUF_OVERFLOW
    //@FIXME NO_ASF
    /*@CTK executeWithSpecificGas_require
      @tag assume_completion
      @post _executionOperators[msg.sender] == true
     */
    function executeWithSpecificGas(address to, uint256 gasLimit, bytes calldata data)
         external returns (bool success, bytes memory returnData) {
        require(_executionOperators[msg.sender], "only execution operators allowed to
            execute on SAND behalf");
        (success, returnData) = to.call.gas(gasLimit)(data);
        assert(gasleft() > gasLimit / 63); // not enough gas provided, assert to throw
            all gas // TODO use EIP-1930
    }

    /// @notice approve a specific amount of token for `from` and execute on behalf of
         the contract.
    /// @param from address of which token will be transfered.
    /// @param to destination address fo the call.
```

```
    /// @param amount number of tokens allowed that can be transfer by the code at `to`.
    /// @param gasLimit exact amount of gas to be passed to the call.
    /// @param data the bytes sent to the destination address.
    /// @return success whether the execution was successful.
    /// @return returnData data resulting from the execution.
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK approveAndExecuteWithSpecificGas_require
      @tag assume_completion
      @post _executionOperators[msg.sender] == true
     */
    function approveAndExecuteWithSpecificGas(
        address from,
        address to,
        uint256 amount,
        uint256 gasLimit,
        bytes calldata data
    ) external returns (bool success, bytes memory returnData) {
        require(_executionOperators[msg.sender], "only execution operators allowed to execute on SAND behalf");
        return _approveAndExecuteWithSpecificGas(from, to, amount, gasLimit, data);
    }

    /// @dev the reason for this function is that charging for gas here is more gas-efficient than doing it in the caller.
    /// @notice approve a specific amount of token for `from` and execute on behalf of the contract. Plus charge the gas required to perform it.
    /// @param from address of which token will be transfered.
    /// @param to destination address fo the call.
    /// @param amount number of tokens allowed that can be transfer by the code at `to`.
    /// @param gasLimit exact amount of gas to be passed to the call.
    /// @param tokenGasPrice price in token for the gas to be charged.
    /// @param baseGasCharge amount of gas charged on top of the gas used for the call.
    /// @param tokenReceiver recipient address of the token charged for the gas used.
    /// @param data the bytes sent to the destination address.
    /// @return success whether the execution was successful.
    /// @return returnData data resulting from the execution.
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK approveAndExecuteWithSpecificGasAndChargeForIt_require
      @tag assume_completion
      @post _executionOperators[msg.sender] == true
     */
    function approveAndExecuteWithSpecificGasAndChargeForIt(
        address from,
        address to,
        uint256 amount,
        uint256 gasLimit,
        uint256 tokenGasPrice,
        uint256 baseGasCharge,
        address tokenReceiver,
        bytes calldata data
    ) external returns (bool success, bytes memory returnData) {
```

```
        uint256 initialGas = gasleft();
        require(_executionOperators[msg.sender], "only execution operators allowed to execute on SAND behalf");
        (success, returnData) = _approveAndExecuteWithSpecificGas(from, to, amount, gasLimit, data);
        if (tokenGasPrice > 0) {
            _charge(from, gasLimit, tokenGasPrice, initialGas, baseGasCharge, tokenReceiver);
        }
    }

    /// @notice transfer 1amount1 token from `from` to `to` and charge the gas required to perform that transfer.
    /// @param from address of which token will be transfered.
    /// @param to destination address fo the call.
    /// @param amount number of tokens allowed that can be transfer by the code at `to`.
    /// @param gasLimit exact amount of gas to be passed to the call.
    /// @param tokenGasPrice price in token for the gas to be charged.
    /// @param baseGasCharge amount of gas charged on top of the gas used for the call.
    /// @param tokenReceiver recipient address of the token charged for the gas used.
    /// @return whether the transfer was successful.
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK transferAndChargeForGas_require
      @tag assume_completion
      @post _executionOperators[msg.sender] == true
     */
    /*@CTK transferAndChargeForGas_return
      @tag assume_completion
      @pre _executionOperators[msg.sender] == true
      @post __return == true
     */
    function transferAndChargeForGas(
        address from,
        address to,
        uint256 amount,
        uint256 gasLimit,
        uint256 tokenGasPrice,
        uint256 baseGasCharge,
        address tokenReceiver
    ) external returns (bool) {
        uint256 initialGas = gasleft();
        require(_executionOperators[msg.sender], "only execution operators allowed to perfrom transfer and charge");
        _transfer(from, to, amount);
        if (tokenGasPrice > 0) {
            _charge(from, gasLimit, tokenGasPrice, initialGas, baseGasCharge, tokenReceiver);
        }
        return true;
    }

    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK FAIL NO_ASF
```

```
    /*@CTK FAIL "_charge gas left lower than limit"
      @tag assume_completion
      @let uint256 gasCharge = initialGas - 1 + baseGasCharge
      @post gasCharge * tokenGasPrice / gasCharge == tokenGasPrice
     */
    /*@CTK FAIL "_charge gas left higher than limit"
      @tag assume_completion
      @let uint256 gasCharge = gasLimit + baseGasCharge
      @post gasCharge * tokenGasPrice / gasCharge == tokenGasPrice
     */
    function _charge(
        address from,
        uint256 gasLimit,
        uint256 tokenGasPrice,
        uint256 initialGas,
        uint256 baseGasCharge,
        address tokenReceiver
    ) internal {
        uint256 gasCharge;
        gasCharge = initialGas - gasleft();
        if(gasCharge > gasLimit) {
            gasCharge = gasLimit;
        }
        gasCharge += baseGasCharge;
        uint256 tokensToCharge = gasCharge * tokenGasPrice;
        require(tokensToCharge / gasCharge == tokenGasPrice, "overflow");
        _transfer(from, tokenReceiver, tokensToCharge);
    }

    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    function _approveAndExecuteWithSpecificGas(
        address from,
        address to,
        uint256 amount,
        uint256 gasLimit,
        bytes memory data
    ) internal returns (bool success, bytes memory returnData) {

        if (amount > 0) {
            _addAllowanceIfNeeded(from, to, amount);
        }
        (success, returnData) = to.call.gas(gasLimit)(data);
        assert(gasleft() > gasLimit / 63); // not enough gas provided, assert to throw
            all gas // TODO use EIP-1930
    }

    function _transfer(address from, address to, uint256 amount) internal;
    function _addAllowanceIfNeeded(address owner, address spender, uint256
        amountNeeded) internal;
}
```

File Admin.sol

```
pragma solidity ^0.5.2;

contract Admin {

```

```
    address internal _admin;

    event AdminChanged(address oldAdmin, address newAdmin);

    /// @notice gives the current administrator of this contract.
    /// @return the current administrator of this contract.
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK getAdmin
      @tag assume_completion
      @post __return == _admin
     */
    function getAdmin() external view returns (address) {
        return _admin;
    }

    /// @notice change the administrator to be `newAdmin`.
    /// @param newAdmin address of the new administrator.
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK changeAdmin_requirement
      @tag assume_completion
      @post msg.sender == _admin
     */
    /*@CTK changeAdmin_change
      @tag assume_completion
      @pre msg.sender == _admin
      @post __post._admin == newAdmin
     */
    function changeAdmin(address newAdmin) external {
        require(msg.sender == _admin, "only admin can change admin");
        emit AdminChanged(_admin, newAdmin);
        _admin = newAdmin;
    }

    modifier onlyAdmin() {
        require (msg.sender == _admin, "only admin allowed");
        _;
    }

}
```

File SuperOperators.sol

```
pragma solidity ^0.5.2;

//import "./Admin.sol";
import "../sandbox-private-contracts/contracts_common/src/BaseWithStorage/Admin.sol";

contract SuperOperators is Admin {

    mapping(address => bool) internal _superOperators;

    event SuperOperator(address superOperator, bool enabled);

    /// @notice Enable or disable the ability of `superOperator` to transfer tokens of
        all (superOperator rights).
```

```
    /// @param superOperator address that will be given/removed superOperator right.
    /// @param enabled set whether the superOperator is enabled or disabled.
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK setSuperOperator_admin
      @tag assume_completion
      @inv msg.sender == _admin
     */
    /*@CTK setSuperOperator_change
      @tag assume_completion
      @pre msg.sender == _admin
      @post __post._superOperators[superOperator] == enabled
     */
    function setSuperOperator(address superOperator, bool enabled) external {
        require(
            msg.sender == _admin,
            "only admin is allowed to add super operators"
        );
        _superOperators[superOperator] = enabled;
        emit SuperOperator(superOperator, enabled);
    }

    /// @notice check whether address `who` is given superOperator rights.
    /// @param who The address to query.
    /// @return whether the address has superOperator rights.
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK isSuperOperator
      @tag assume_completion
      @post __return == _superOperators[who]
     */
    function isSuperOperator(address who) public view returns (bool) {
        return _superOperators[who];
    }
}
```

File Sand.sol

```
pragma solidity 0.5.9;

import "../sandbox-private-contracts/src/Sand/erc20/ERC20ExecuteExtension.sol";
import "../sandbox-private-contracts/src/Sand/erc20/ERC20BaseToken.sol";
import "../sandbox-private-contracts/src/Sand/erc20/ERC20BasicApproveExtension.sol";

contract Sand is ERC20ExecuteExtension, ERC20BasicApproveExtension, ERC20BaseToken {

    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK Sand_require
      @tag assume_completion
      @post beneficiary != address(0)
      @post _totalSupply + 3000000000000000000000000000 > _totalSupply
     */
    /*@CTK Sand_change
      @tag assume_completion
      @pre beneficiary != address(0)
```

```
     @pre _totalSupply + 3000000000000000000000000000 > _totalSupply
     @post __post._totalSupply == _totalSupply + 3000000000000000000000000000
     @post __post._balances[beneficiary] == _balances[beneficiary] +
         3000000000000000000000000000
     @post __post._balances[beneficiary] == __post._totalSupply
    */
   constructor(address sandAdmin, address executionAdmin, address beneficiary) public
        {
       _admin = sandAdmin;
       _executionAdmin = executionAdmin;
       _mint(beneficiary, 3000000000000000000000000000);
   }

   /// @notice A descriptive name for the tokens
   /// @return name of the tokens
   //@CTK NO_OVERFLOW
   //@CTK NO_BUF_OVERFLOW
   //@CTK NO_ASF
   /*@CTK name
     @post __return == "SAND"
    */
   function name() public view returns (string memory) {
       return "SAND";
   }

   /// @notice An abbreviated name for the tokens
   /// @return symbol of the tokens
  /// @notice A descriptive name for the tokens
   /// @return name of the tokens
   //@CTK NO_OVERFLOW
   //@CTK NO_BUF_OVERFLOW
   //@CTK NO_ASF
   /*@CTK symbol
     @post __return == "SAND"
    */
   function symbol() public view returns (string memory) {
       return "SAND";
   }

}
```

File ERC20BaseToken.sol

```
pragma solidity 0.5.9;

import "../sandbox-private-contracts/contracts_common/src/Interfaces/ERC20Events.sol";
import "../sandbox-private-contracts/contracts_common/src/BaseWithStorage/
    SuperOperators.sol";

contract ERC20BaseToken is SuperOperators, ERC20Events {

   uint256 internal _totalSupply;
   mapping(address => uint256) internal _balances;
   mapping(address => mapping(address => uint256)) internal _allowances;

   /// @notice Gets the total number of tokens in existence.
   /// @return the total number of tokens in existence.
   /*@CTK totalSupply
     @post __return == _totalSupply
```

```
     */
    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }

    /// @notice Gets the balance of `owner`.
    /// @param owner The address to query the balance of.
    /// @return The amount owned by `owner`.
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK balanceOf
      @post __return == _balances[owner]
     */
    function balanceOf(address owner) public view returns (uint256) {
        return _balances[owner];
    }

    /// @notice gets allowance of `spender` for `owner`'s tokens.
    /// @param owner address whose token is allowed.
    /// @param spender address allowed to transfer.
    /// @return the amount of token `spender` is allowed to transfer on behalf of `
        owner`.
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK allowance
      @post __return == _allowances[owner][spender]
     */
    function allowance(address owner, address spender)
        public
        view
        returns (uint256)
    {
        return _allowances[owner][spender];
    }

    /// @notice returns the number of decimals for that token.
    /// @return the number of decimals.
    /*@CTK decimals
      @post __return == 18
     */
    function decimals() public view returns (uint8) {
        return uint8(18);
    }

    /// @notice Transfer `amount` tokens to `to`.
    /// @param to the recipient address of the tokens transfered.
    /// @param amount the number of tokens transfered.
    /// @return true if success.
    //@CTK FAIL NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK transfer_require
      @tag assume_completion
      @post to != address(0)
      @post _balances[msg.sender] >= amount
     */
```

```
    /*@CTK transfer_change
      @tag assume_completion
      @pre to != address(0)
      @pre _balances[msg.sender] >= amount
      @pre msg.sender != to
      @post __post._balances[msg.sender] == _balances[msg.sender] - amount
      @post __post._balances[to] == _balances[to] + amount
      @post __return == true
     */
    function transfer(address to, uint256 amount)
        public
        returns (bool)
    {
        _transfer(msg.sender, to, amount);
        return true;
    }

    /// @notice Transfer `amount` tokens from `from` to `to`.
    /// @param from whose token it is transferring from.
    /// @param to the recipient address of the tokens transfered.
    /// @param amount the number of tokens transfered.
    /// @return true if success.
    //@CTK FAIL NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK transferFrom_require
      @tag assume_completion
      @post to != address(0)
      @post _balances[from] >= amount
      @post (msg.sender != from && !_superOperators[msg.sender] && _allowances[from][msg.sender] != (2**256) - 1) -> _allowances[from][msg.sender] >= amount
     */
    /*@CTK transferFrom_change
      @tag assume_completion
      @pre to != address(0)
      @pre _balances[from] >= amount
      @pre (msg.sender != from && !_superOperators[msg.sender] && _allowances[from][msg.sender] != (2**256) - 1) -> _allowances[from][msg.sender] >= amount
      @pre from != to
      @post (msg.sender != from && !_superOperators[msg.sender] && _allowances[from][msg.sender] != (2**256) - 1) -> __post._allowances[from][msg.sender] == _allowances[from][msg.sender] - amount
      @post __post._balances[from] == _balances[from] - amount
      @post __post._balances[to] == _balances[to] + amount
      @post __return == true
     */
    function transferFrom(address from, address to, uint256 amount)
        public
        returns (bool)
    {
        if (msg.sender != from && !_superOperators[msg.sender]) {
            uint256 currentAllowance = _allowances[from][msg.sender];
            if (currentAllowance != (2**256) - 1) {
                // save gas when allowance is maximal by not reducing it (see https://github.com/ethereum/EIPs/issues/717)
                require(currentAllowance >= amount, "Not enough funds allowed");
                _allowances[from][msg.sender] = currentAllowance - amount;
            }
```

```
        }
        _transfer(from, to, amount);
        return true;
    }

    /// @notice burn `amount` tokens.
    /// @param amount the number of tokens to burn.
    /// @return true if success.
    //@CTK FAIL NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK burn_require
      @tag assume_completion
      @post amount > 0
      @post _balances[msg.sender] >= amount
     */
    /*@CTK burn_change
      @tag assume_completion
      @pre amount > 0
      @pre _balances[msg.sender] >= amount
      @post __post._balances[msg.sender] == _balances[msg.sender] - amount
      @post __post._totalSupply == _totalSupply - amount
      @post __return == true
     */
    function burn(uint256 amount) external returns (bool) {
        _burn(msg.sender, amount);
        return true;
    }

    /// @notice burn `amount` tokens from `owner`.
    /// @param owner address whose token is to burn.
    /// @param amount the number of token to burn.
    /// @return true if success.
    //@CTK FAIL NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK burnFor_require
      @tag assume_completion
      @post amount > 0
      @post _balances[owner] >= amount
      @post (msg.sender != owner && !_superOperators[msg.sender]) -> _allowances[owner
          ][msg.sender] >= amount
     */
    /*@CTK burnFor_change
      @tag assume_completion
      @pre amount > 0
      @pre _balances[owner] >= amount
      @pre (msg.sender != owner && !_superOperators[msg.sender]) -> _allowances[owner][
          msg.sender] >= amount
      @post (msg.sender != owner && !_superOperators[msg.sender] && _allowances[owner][
          msg.sender] != (2**256) - 1) -> __post._allowances[owner][msg.sender] ==
          _allowances[owner][msg.sender] - amount
      @post __post._balances[owner] == _balances[owner] - amount
      @post __post._totalSupply == _totalSupply - amount
     */
    function burnFor(address owner, uint256 amount) external returns (bool) {
        _burn(owner, amount);
        return true;
```

```
    }

    /// @notice approve `spender` to transfer `amount` tokens.
    /// @param spender address to be given rights to transfer.
    /// @param amount the number of tokens allowed.
    /// @return true if success.
    //@FIXME NO_OVERFLOW
    //@FIXME NO_BUF_OVERFLOW
    //@FIXME NO_ASF
    /*@FIXME approve_require
      @tag assume_completion
      @post msg.sender != address(0)
      @post spender != address(0)
     */
    /*@FIXME approve_change
      @tag assume_completion
      @pre msg.sender != address(0)
      @pre spender != address(0)
      @post __post._allowances[msg.sender][spender] == amount
     */
    function approve(address spender, uint256 amount)
        public
        returns (bool success)
    {
        _approveFor(msg.sender, spender, amount);
        return true;
    }

    /// @notice approve `spender` to transfer `amount` tokens from `owner`.
    /// @param owner address whose token is allowed.
    /// @param spender address to be given rights to transfer.
    /// @param amount the number of tokens allowed.
    /// @return true if success.
    //@FIXME NO_OVERFLOW
    //@FIXME NO_BUF_OVERFLOW
    //@FIXME NO_ASF
    /*@FIXME approveFor_require
      @tag assume_completion
      @post msg.sender == owner || _superOperators[msg.sender]
      @post owner != address(0)
      @post spender != address(0)
     */
    /*@FIXME approveFor_change
      @tag assume_completion
      @pre msg.sender == owner || _superOperators[msg.sender]
      @pre owner != address(0)
      @pre spender != address(0)
      @post __post._allowances[owner][spender] == amount
      @post __return == true
     */
    function approveFor(address owner, address spender, uint256 amount)
        public
        returns (bool success)
    {
        require(
            msg.sender == owner || _superOperators[msg.sender],
            "msg.sender != owner && !superOperator"
        );
```

```
        _approveFor(owner, spender, amount);
        return true;
    }

    //@FIXME NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK addAllowanceIfNeeded_require
      @tag assume_completion
      @post msg.sender == owner || _superOperators[msg.sender]
      @post (amountNeeded > 0 && !_superOperators[spender] && _allowances[owner][spender] < amountNeeded) -> owner != address(0)
      @post (amountNeeded > 0 && !_superOperators[spender] && _allowances[owner][spender] < amountNeeded) -> spender != address(0)
     */
    /*@CTK addAllowanceIfNeeded_change
      @tag assume_completion
      @pre msg.sender == owner || _superOperators[msg.sender]
      @pre (amountNeeded > 0 && !_superOperators[spender] && _allowances[owner][spender] < amountNeeded) -> owner != address(0)
      @pre (amountNeeded > 0 && !_superOperators[spender] && _allowances[owner][spender] < amountNeeded) -> spender != address(0)
      @post (amountNeeded > 0 && !_superOperators[spender] && _allowances[owner][spender] < amountNeeded) -> __post._allowances[owner][spender] == amountNeeded
     */
    function addAllowanceIfNeeded(address owner, address spender, uint256 amountNeeded)
        public
        returns (bool success)
    {
        require(
            msg.sender == owner || _superOperators[msg.sender],
            "msg.sender != owner && !superOperator"
        );
        _addAllowanceIfNeeded(owner, spender, amountNeeded);
        return true;
    }

    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK _addAllowanceIfNeeded_require
      @tag assume_completion
      @post (amountNeeded > 0 && !_superOperators[spender] && _allowances[owner][spender] < amountNeeded) -> owner != address(0)
      @post (amountNeeded > 0 && !_superOperators[spender] && _allowances[owner][spender] < amountNeeded) -> spender != address(0)
     */
    /*@CTK _addAllowanceIfNeeded_change
      @tag assume_completion
      @pre (amountNeeded > 0 && !_superOperators[spender] && _allowances[owner][spender] < amountNeeded) -> owner != address(0)
      @pre (amountNeeded > 0 && !_superOperators[spender] && _allowances[owner][spender] < amountNeeded) -> spender != address(0)
      @post (amountNeeded > 0 && !_superOperators[spender] && _allowances[owner][spender] < amountNeeded) -> __post._allowances[owner][spender] == amountNeeded
```

```
 */
function _addAllowanceIfNeeded(address owner, address spender, uint256
    amountNeeded)
    internal
{
    if(amountNeeded > 0 && !isSuperOperator(spender)) {
        uint256 currentAllowance = _allowances[owner][spender];
        if(currentAllowance < amountNeeded) {
            _approveFor(owner, spender, amountNeeded);
        }
    }
}

//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK _approveFor_require
  @tag assume_completion
  @post owner != address(0)
  @post spender != address(0)
 */
/*@CTK _approveFor_change
  @tag assume_completion
  @pre owner != address(0)
  @pre spender != address(0)
  @post __post._allowances[owner][spender] == amount
 */
function _approveFor(address owner, address spender, uint256 amount)
    internal
{
    require(
        owner != address(0) && spender != address(0),
        "Cannot approve with 0x0"
    );
    _allowances[owner][spender] = amount;
    emit Approval(owner, spender, amount);
}

//@CTK FAIL NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK _transfer_require
  @tag assume_completion
  @post to != address(0)
  @post _balances[from] >= amount
 */
/*@CTK _transfer_change
  @tag assume_completion
  @pre to != address(0)
  @pre _balances[from] >= amount
  @pre from != to
  @post __post._balances[from] == _balances[from] - amount
  @post __post._balances[to] == _balances[to] + amount
 */
function _transfer(address from, address to, uint256 amount) internal {
    require(to != address(0), "Cannot send to 0x0");
    uint256 currentBalance = _balances[from];
    require(currentBalance >= amount, "not enough fund");
```

```
        _balances[from] = currentBalance - amount;
        _balances[to] += amount;
        emit Transfer(from, to, amount);
    }

    //@CTK FAIL NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK _mint_require_no_overflow
      @tag assume_completion
      @post to != address(0)
      @post amount > 0
      @post _totalSupply + amount > _totalSupply
     */
    /*@CTK _mint_change
      @tag assume_completion
      @pre to != address(0)
      @pre amount > 0
      @pre _totalSupply + amount > _totalSupply
      @post __post._totalSupply == _totalSupply + amount
      @post __post._balances[to] == _balances[to] + amount
     */
    function _mint(address to, uint256 amount) internal {
        require(to != address(0), "Cannot mint to 0x0");
        require(amount > 0, "cannot mint 0 tokens");
        uint256 currentTotalSupply = _totalSupply;
        uint256 newTotalSupply = currentTotalSupply + amount;
        require(newTotalSupply > currentTotalSupply, "overflow");
        _totalSupply = newTotalSupply;
        _balances[to] += amount;
        emit Transfer(address(0), to, amount);
    }

    //@CTK FAIL NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK _burn_require_identity
      @tag assume_completion
      @post amount > 0
      @post _balances[from] >= amount
      @post (msg.sender != from && !_superOperators[msg.sender]) -> _allowances[from][
          msg.sender] >= amount
     */
    /*@CTK _burn_change
      @tag assume_completion
      @pre amount > 0
      @pre _balances[from] >= amount
      @pre (msg.sender != from && !_superOperators[msg.sender]) -> _allowances[from][
          msg.sender] >= amount
      @post (msg.sender != from && !_superOperators[msg.sender] && _allowances[from][
          msg.sender] != (2**256) - 1) -> __post._allowances[from][msg.sender] ==
          _allowances[from][msg.sender] - amount
      @post __post._balances[from] == _balances[from] - amount
      @post __post._totalSupply == _totalSupply - amount
     */
    function _burn(address from, uint256 amount) internal {
        require(amount > 0, "cannot burn 0 tokens");
        if (msg.sender != from && !_superOperators[msg.sender]) {
```

```
        uint256 currentAllowance = _allowances[from][msg.sender];
        require(
            currentAllowance >= amount,
            "Not enough funds allowed"
        );
        if (currentAllowance != (2**256) - 1) {
            // save gas when allowance is maximal by not reducing it (see https://
                github.com/ethereum/EIPs/issues/717)
            _allowances[from][msg.sender] = currentAllowance - amount;
        }
    }

    uint256 currentBalance = _balances[from];
    require(currentBalance >= amount, "Not enough funds");
    _balances[from] = currentBalance - amount;
    _totalSupply -= amount;
    emit Transfer(from, address(0), amount);
  }
}
```

File ERC20BasicApproveExtension.sol

```
pragma solidity 0.5.9;

contract ERC20BasicApproveExtension {

  /// @notice approve `target` to spend `amount` and call it with data.
  /// @param target address to be given rights to transfer and destination of the
      call.
  /// @param amount the number of tokens allowed.
  /// @param data bytes for the call.
  /// @return data of the call.
  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  function approveAndCall(
    address target,
    uint256 amount,
    bytes calldata data
  ) external payable returns (bytes memory) {
    require(
      doFirstParamEqualsAddress(data, msg.sender),
      "first param != sender"
    );
    _approveFor(msg.sender, target, amount);

    // solium-disable-next-line security/no-call-value
    (bool success, bytes memory returnData) = target.call.value(msg.value)(data);
    require(success, string(returnData));
    return returnData;
  }

  /// @notice temporarly approve `target` to spend `amount` and call it with data.
      Previous approvals remains unchanged.
  /// @param target destination of the call, allowed to spend the amount specified
  /// @param amount the number of tokens allowed to spend.
  /// @param data bytes for the call.
  /// @return data of the call.
  //@CTK NO_OVERFLOW
```

```
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    function paidCall(
        address target,
        uint256 amount,
        bytes calldata data
    ) external payable returns (bytes memory) {
        require(
            doFirstParamEqualsAddress(data, msg.sender),
            "first param != sender"
        );

        if (amount > 0) {
            _addAllowanceIfNeeded(msg.sender, target, amount);
        }

        // solium-disable-next-line security/no-call-value
        (bool success, bytes memory returnData) = target.call.value(msg.value)(data);
        require(success, string(returnData));

        return returnData;
    }

    function _approveFor(address owner, address target, uint256 amount) internal;
    function _addAllowanceIfNeeded(address owner, address spender, uint256
        amountNeeded) internal;

    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK doFirstParamEqualsAddress_require
      @tag assume_completion
      @post (data.length < (36 + 32)) -> !__return
     */
    function doFirstParamEqualsAddress(bytes memory data, address _address)
        internal
        pure
        returns (bool)
    {
        if (data.length < (36 + 32)) {
            return false;
        }
        uint256 value;
        /*@CTK "Load 32 bytes from mem[data+36]"
          @var uint value
          @post value == uint256(_address)
         */
        assembly {
            value := mload(add(data, 36))
        }
        return value == uint256(_address);
    }
}
```