

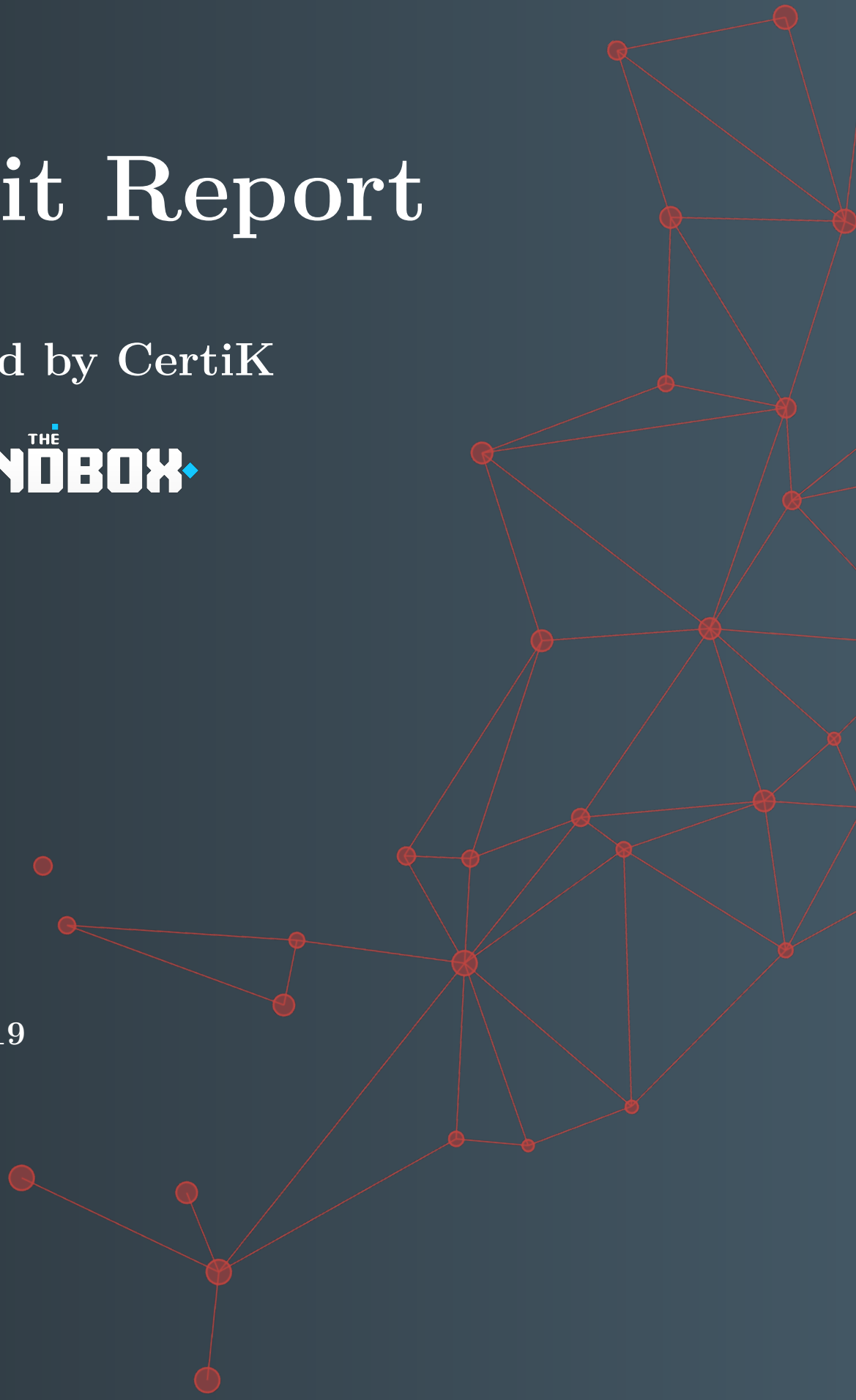


Audit Report

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	1
About CertiK	2
Executive Summary	3
Vulnerability Classification	3
Testing Summary	4
Audit Score	4
Type of Issues	4
Vulnerability Details	5
Manual Review Notes	6
Static Analysis Results	8
Formal Verification Results	9
How to read	9
Source Code with CertiK Labels	102

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

PASS

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

Dec 02, 2019



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` throughout 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;
```

i 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;
```

i 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
	52	}
	53	balance: 0x0
	54	}
	55	}
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)
97      external returns (bool success, bytes memory returnData) {
98      require(_executionOperators[msg.sender], "only execution operators allowed to
99          execute on SAND behalf");
100      (success, returnData) = to.call.gas(gasLimit)(data);
101      assert(gasleft() > gasLimit / 63); // not enough gas provided, assert to throw
102      all gas // TODO use EIP-1930
103  }
```

 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
127          execute on SAND behalf");
128      return _approveAndExecuteWithSpecificGas(from, to, amount, gasLimit, data);
129  }
```

 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

```
115 /*@CTK approveAndExecuteWithSpecificGas_require
116    @tag assume_completion
117    @post _executionOperators[msg.sender] == true
118 */
```

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 24

If method completes, integer overflow would not happen.

📅 02, Dec 2019

🕒 20.45 ms

Line 144 in File ERC20ExecuteExtension.sol

```
144 //@CTK NO_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,
166                 tokenReceiver);
167     }

```

✓ 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
163         execute on SAND behalf");
164     (success, returnData) = _approveAndExecuteWithSpecificGas(from, to, amount,
165         gasLimit, data);
166     if (tokenGasPrice > 0) {
167         _charge(from, gasLimit, tokenGasPrice, initialGas, baseGasCharge,
168             tokenReceiver);
169     }
170 }

```

✓ 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     }

```



167 }

✓ 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

182 `//@CTK NO_OVERFLOW`

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
        perform 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 29

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

 02, Dec 2019

 0.66 ms

Line 183 in File ERC20ExecuteExtension.sol

183 `//@CTK NO_BUF_OVERFLOW`

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
        perform 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 30

Method will not encounter an assertion failure.

📅 02, Dec 2019

🕒 0.54 ms

Line 184 in File ERC20ExecuteExtension.sol

```
184 // @CTK NO_ASF
```

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
        perform 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 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
        perform 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
           perform 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.

```


1 Counter Example:
2 Before Execution:
3   Input = {
4     baseGasCharge = 0
5     from = 0
6     gasLimit = 0
7     initialGas = 0
8     tokenGasPrice = 0
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    }
29  }
30  Address_Map = [
31    {
32      "key": "ALL_OTHERS",
33      "value": {
34        "contract_name": "ERC20ExecuteExtension",
35        "balance": 0,
36        "contract": {
37          "_executionAdmin": 0,
38          "_executionOperators": [
39            {
40              "key": "ALL_OTHERS",
41              "value": false
42            }
43          ]
44        }
45      }
46    }
47  ]
48
49 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    }
```

```

29 }
30 Address_Map = [
31   {
32     "key": "ALL_OTHERS",
33     "value": {
34       "contract_name": "ERC20ExecuteExtension",
35       "balance": 0,
36       "contract": {
37         "_executionAdmin": 0,
38         "_executionOperators": [
39           {
40             "key": "ALL_OTHERS",
41             "value": false
42           }
43         ]
44       }
45     }
46   }
47 ]
48
49 After Execution:
50 Input = {
51   baseGasCharge = 1
52   from = 0
53   gasLimit = 0
54   initialGas = 63
55   tokenGasPrice = 254
56   tokenReceiver = 0
57 }
58 This = 0
59 Internal = {
60   __has_assertion_failure = false
61   __has_buf_overflow = false
62   __has_overflow = false
63   __has_returned = false
64   __reverted = false
65   msg = {
66     "gas": 0,
67     "sender": 0,
68     "value": 0
69   }
70 }
71 Other = {
72   block = {
73     "number": 0,
74     "timestamp": 0
75   }
76   gasCharge = 1
77   tokensToCharge = 254
78 }
79 Address_Map = [
80   {
81     "key": "ALL_OTHERS",
82     "value": {
83       "contract_name": "ERC20ExecuteExtension",
84       "balance": 0,
85       "contract": {
86         "_executionAdmin": 0,

```



```


87         "_executionOperators": [
88             {
89                 "key": "ALL_OTHERS",
90                 "value": false
91             }
92         ]
93     }
94 }
95 ]
96 ]

```

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     }

```

```

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     }
29 }
30 Address_Map = [
31     {
32         "key": "ALL_OTHERS",
33         "value": {
34             "contract_name": "ERC20ExecuteExtension",
35             "balance": 0,
36             "contract": {
37                 "_executionAdmin": 0,
38                 "_executionOperators": [
39                     {
40                         "key": "ALL_OTHERS",
41                         "value": false
42                     }
43                 ]
44             }
45         }
46     }
47 ]

```

49 After Execution:

```

50 Input = {
51     baseGasCharge = 14
52     from = 0
53     gasLimit = 232
54     initialGas = 0
55     tokenGasPrice = 6
56     tokenReceiver = 0
57 }
58 This = 0
59 Internal = {
60     __has_assertion_failure = false
61     __has_buf_overflow = false
62     __has_overflow = false
63     __has_returned = false
64     __reverted = false
65     msg = {
66         "gas": 0,
67         "sender": 0,
68         "value": 0

```

```


69     }
70 }
71 Other = {
72     block = {
73         "number": 0,
74         "timestamp": 0
75     }
76     gasCharge = 14
77     tokensToCharge = 84
78 }
79 Address_Map = [
80     {
81         "key": "ALL_OTHERS",
82         "value": {
83             "contract_name": "ERC20ExecuteExtension",
84             "balance": 0,
85             "contract": {
86                 "_executionAdmin": 0,
87                 "_executionOperators": [
88                     {
89                         "key": "ALL_OTHERS",
90                         "value": false
91                     }
92                 ]
93             }
94         }
95     }
96 ]

```

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
268        all gas // TODO use EIP-1930
269    }

```

✓ 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
268         all gas // TODO use EIP-1930
    }
```

✓ 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         }
    }
```

```
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 41

If method completes, integer overflow would not happen.

📅 02, Dec 2019

🕒 4.87 ms

Line 11 in File Admin.sol

```
11 // @CTK NO_OVERFLOW
```

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 42

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

📅 02, Dec 2019

🕒 0.38 ms

Line 12 in File Admin.sol

```
12 // @CTK NO_BUF_OVERFLOW
```

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 43

Method will not encounter an assertion failure.

📅 02, Dec 2019

🕒 0.32 ms

Line 13 in File Admin.sol

```
13 // @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     {
27         _admin = sandAdmin;
27         _executionAdmin = executionAdmin;
28         _mint(beneficiary, 30000000000000000000000000000000);
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     {
26         _admin = sandAdmin;
27         _executionAdmin = executionAdmin;
28         _mint(beneficiary, 30000000000000000000000000000000);
29     }
```

✓ The code meets the specification.


```

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.

```

1 Counter Example:
2 Before Execution:
3   Input = {
4     amount = 128
5     to = 2
6   }
7   This = 0
8   Internal = {
9     __has_assertion_failure = false
10    __has_buf_overflow = false
11    __has_overflow = false
12    __has_returned = false
13    __reverted = false
14    msg = {
15      "gas": 0,
16      "sender": 0,
17      "value": 0
18    }
19  }
20  Other = {
21    __return = false
22    block = {
23      "number": 0,
24      "timestamp": 0
25    }
26  }
27  Address_Map = [
28    {
29      "key": 0,
30      "value": {
31        "contract_name": "ERC20BaseToken",
32        "balance": 0,
33        "contract": {
34          "_totalSupply": 0,
35          "_balances": [
36            {
37              "key": "ALL_OTHERS",
38              "value": 128
39            }
40          ],
41          "_allowances": [
42            {
43              "key": "ALL_OTHERS",
44              "value": [
45                {
46                  "key": "ALL_OTHERS",
47                  "value": 128
48                }
49              ]
50            }
51          ],
52          "_superOperators": [
53            {
54              "key": "ALL_OTHERS",
55              "value": false
56            }
57          ],

```

```

58         "_admin": 0
59     }
60 }
61 },
62 {
63     "key": "ALL_OTHERS",
64     "value": "EmptyAddress"
65 }
66 ]
67
68 After Execution:
69     Input = {
70         amount = 128
71         to = 2
72     }
73     This = 0
74     Internal = {
75         __has_assertion_failure = false
76         __has_buf_overflow = false
77         __has_overflow = true
78         __has_returned = true
79         __reverted = false
80         msg = {
81             "gas": 0,
82             "sender": 0,
83             "value": 0
84         }
85     }
86     Other = {
87         __return = true
88         block = {
89             "number": 0,
90             "timestamp": 0
91         }
92     }
93     Address_Map = [
94     {
95         "key": 0,
96         "value": {
97             "contract_name": "ERC20BaseToken",
98             "balance": 0,
99             "contract": {
100                 "_totalSupply": 0,
101                 "_balances": [
102                 {
103                     "key": 0,
104                     "value": 0
105                 },
106                 {
107                     "key": 2,
108                     "value": 0
109                 },
110                 {
111                     "key": "ALL_OTHERS",
112                     "value": 128
113                 }
114             ],
115             "_allowances": [

```

```


116     {
117         "key": "ALL_OTHERS",
118         "value": [
119             {
120                 "key": "ALL_OTHERS",
121                 "value": 128
122             }
123         ]
124     },
125 ],
126 "_superOperators": [
127     {
128         "key": "ALL_OTHERS",
129         "value": false
130     }
131 ],
132 "_admin": 0
133 }
134 }
135 },
136 {
137     "key": "ALL_OTHERS",
138     "value": "EmptyAddress"
139 }
140 ]

```

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://
123                 // github.com/ethereum/EIPs/issues/717)
124                 require(currentAllowance >= amount, "Not enough funds allowed");
125                 _allowances[from][msg.sender] = currentAllowance - amount;
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
```

```

13     __has_returned = false
14     __reverted = false
15     msg = {
16         "gas": 0,
17         "sender": 128,
18         "value": 0
19     }
20 }
21 Other = {
22     __return = false
23     block = {
24         "number": 0,
25         "timestamp": 0
26     }
27 }
28 Address_Map = [
29     {
30         "key": 0,
31         "value": {
32             "contract_name": "ERC20BaseToken",
33             "balance": 0,
34             "contract": {
35                 "_totalSupply": 0,
36                 "_balances": [
37                     {
38                         "key": 2,
39                         "value": 4
40                     },
41                     {
42                         "key": 0,
43                         "value": 128
44                     },
45                     {
46                         "key": 4,
47                         "value": 4
48                     },
49                     {
50                         "key": 1,
51                         "value": 128
52                     },
53                     {
54                         "key": "ALL_OTHERS",
55                         "value": 46
56                     }
57                 ],
58                 "_allowances": [
59                     {
60                         "key": 0,
61                         "value": [
62                             {
63                                 "key": 0,
64                                 "value": 2
65                             },
66                             {
67                                 "key": 128,
68                                 "value": 174
69                             },
70                             {

```

```

71         "key": "ALL_OTHERS",
72         "value": 0
73     }
74 ]
75 },
76 {
77     "key": "ALL_OTHERS",
78     "value": [
79         {
80             "key": "ALL_OTHERS",
81             "value": 46
82         }
83     ]
84 }
85 ],
86 "_superOperators": [
87     {
88         "key": "ALL_OTHERS",
89         "value": false
90     }
91 ],
92 "_admin": 0
93 }
94 }
95 },
96 {
97     "key": "ALL_OTHERS",
98     "value": "EmptyAddress"
99 }
100 ]

```

102 After Execution:

```

103     Input = {
104         amount = 128
105         from = 0
106         to = 1
107     }
108     This = 0
109     Internal = {
110         __has_assertion_failure = false
111         __has_buf_overflow = false
112         __has_overflow = true
113         __has_returned = true
114         __reverted = false
115         msg = {
116             "gas": 0,
117             "sender": 128,
118             "value": 0
119         }
120     }
121     Other = {
122         __return = true
123         block = {
124             "number": 0,
125             "timestamp": 0
126         }
127     }
128     Address_Map = [

```

```

129 {
130   "key": 0,
131   "value": {
132     "contract_name": "ERC20BaseToken",
133     "balance": 0,
134     "contract": {
135       "_totalSupply": 0,
136       "_balances": [
137         {
138           "key": 2,
139           "value": 4
140         },
141         {
142           "key": 0,
143           "value": 0
144         },
145         {
146           "key": 4,
147           "value": 4
148         },
149         {
150           "key": 1,
151           "value": 0
152         },
153         {
154           "key": "ALL_OTHERS",
155           "value": 46
156         }
157       ],
158       "_allowances": [
159         {
160           "key": 0,
161           "value": [
162             {
163               "key": 0,
164               "value": 2
165             },
166             {
167               "key": 128,
168               "value": 46
169             },
170             {
171               "key": "ALL_OTHERS",
172               "value": 0
173             }
174           ]
175         },
176         {
177           "key": "ALL_OTHERS",
178           "value": [
179             {
180               "key": "ALL_OTHERS",
181               "value": 46
182             }
183           ]
184         }
185       ],
186       "_superOperators": [


```

```
187         {
188             "key": "ALL_OTHERS",
189             "value": false
190         }
191     ],
192     "_admin": 0
193 }
194 }
195 },
196 {
197     "key": "ALL_OTHERS",
198     "value": "EmptyAddress"
199 }
200 ]
```

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://
123             // github.com/ethereum/EIPs/issues/717)
124             require(currentAllowance >= amount, "Not enough funds allowed");
125             _allowances[from][msg.sender] = currentAllowance - amount;
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

97 //CTK NO_ASF

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://
123             // github.com/ethereum/EIPs/issues/717)
124             require(currentAllowance >= amount, "Not enough funds allowed");
125             _allowances[from][msg.sender] = currentAllowance - amount;
126         }
127     }
128     _transfer(from, to, amount);
129     return true;
130 }

```

✓ The code meets the specification.

Formal Verification Request 90

transferFrom_require

📅 02, Dec 2019

🕒 39.52 ms

Line 98-103 in File ERC20BaseToken.sol

```

98 /*CTK transferFrom_require
99 @tag assume_completion
100 @post to != address(0)
101 @post _balances[from] >= amount
102 @post (msg.sender != from && !_superOperators[msg.sender] && _allowances[from][
103         msg.sender] != (2**256) - 1) -> _allowances[from][msg.sender] >= amount
104 */

```

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://
123             // github.com/ethereum/EIPs/issues/717)
124             require(currentAllowance >= amount, "Not enough funds allowed");
125             _allowances[from][msg.sender] = currentAllowance - amount;
126         }
127     }
128     _transfer(from, to, amount);
129     return true;
130 }

```

✓ 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

134 //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.

```
1 Counter Example:
2 Before Execution:
3   Input = {
4     amount = 254
5   }
6   This = 0
7   Internal = {
8     __has_assertion_failure = false
9     __has_buf_overflow = false
10    __has_overflow = false
11    __has_returned = false
12    __reverted = false
13    msg = {
14      "gas": 0,
15      "sender": 0,
16      "value": 0
17    }
18  }
19  Other = {
20    __return = false
21    block = {
22      "number": 0,
23      "timestamp": 0
24    }
25  }
26  Address_Map = [
27    {
28      "key": 0,
29      "value": {
30        "contract_name": "ERC20BaseToken",
31        "balance": 0,
32        "contract": {
33          "_totalSupply": 0,
34          "_balances": [
35            {
36              "key": 0,
37              "value": 255
38            },
39            {
40              "key": "ALL_OTHERS",
41              "value": 254
42            }
43          ],
44          "_allowances": [
45            {
46              "key": "ALL_OTHERS",
47              "value": [
48                {
49                  "key": "ALL_OTHERS",
```



```

50         "value": 254
51     }
52 ]
53 }
54 ],
55 "_superOperators": [
56 {
57     "key": "ALL_OTHERS",
58     "value": false
59 }
60 ],
61 "_admin": 0
62 }
63 }
64 },
65 {
66     "key": "ALL_OTHERS",
67     "value": "EmptyAddress"
68 }
69 ]
70
71 After Execution:
72 Input = {
73     amount = 254
74 }
75 This = 0
76 Internal = {
77     __has_assertion_failure = false
78     __has_buf_overflow = false
79     __has_overflow = true
80     __has_returned = true
81     __reverted = false
82     msg = {
83         "gas": 0,
84         "sender": 0,
85         "value": 0
86     }
87 }
88 Other = {
89     __return = true
90     block = {
91         "number": 0,
92         "timestamp": 0
93     }
94 }
95 Address_Map = [
96 {
97     "key": 0,
98     "value": {
99         "contract_name": "ERC20BaseToken",
100         "balance": 0,
101         "contract": {
102             "_totalSupply": 2,
103             "_balances": [
104                 {
105                     "key": 0,
106                     "value": 1
107                 },

```

```


108     {
109         "key": "ALL_OTHERS",
110         "value": 254
111     }
112 ],
113 "_allowances": [
114     {
115         "key": "ALL_OTHERS",
116         "value": [
117             {
118                 "key": "ALL_OTHERS",
119                 "value": 254
120             }
121         ]
122     }
123 ],
124 "_superOperators": [
125     {
126         "key": "ALL_OTHERS",
127         "value": false
128     }
129 ],
130 "_admin": 0
131 }
132 }
133 },
134 {
135     "key": "ALL_OTHERS",
136     "value": "EmptyAddress"
137 }
138 ]

```

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

```

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 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.

```

1 Counter Example:
2 Before Execution:
3     Input = {
4         amount = 127
5         owner = 2
6     }
7     This = 0
8     Internal = {
9         __has_assertion_failure = false
10        __has_buf_overflow = false
11        __has_overflow = false
12        __has_returned = false
13        __reverted = false
14        msg = {
15            "gas": 0,
16            "sender": 0,
17            "value": 0
18        }
19    }
20    Other = {
21        __return = false
22        block = {
23            "number": 0,
24            "timestamp": 0
25        }
26    }
27    Address_Map = [
28        {
29            "key": 0,
30            "value": {
31                "contract_name": "ERC20BaseToken",

```

```

32     "balance": 0,
33     "contract": {
34         "_totalSupply": 0,
35         "_balances": [
36             {
37                 "key": 66,
38                 "value": 0
39             },
40             {
41                 "key": 0,
42                 "value": 1
43             },
44             {
45                 "key": 4,
46                 "value": 0
47             },
48             {
49                 "key": 2,
50                 "value": 234
51             },
52             {
53                 "key": 128,
54                 "value": 0
55             },
56             {
57                 "key": "ALL_OTHERS",
58                 "value": 64
59             }
60         ],
61         "_allowances": [
62             {
63                 "key": 0,
64                 "value": [
65                     {
66                         "key": 0,
67                         "value": 0
68                     },
69                     {
70                         "key": "ALL_OTHERS",
71                         "value": 64
72                     }
73                 ]
74             },
75             {
76                 "key": 1,
77                 "value": [
78                     {
79                         "key": 0,
80                         "value": 1
81                     },
82                     {
83                         "key": "ALL_OTHERS",
84                         "value": 64
85                     }
86                 ]
87             },
88             {
89                 "key": "ALL_OTHERS",

```

```

90         "value": [
91             {
92                 "key": "ALL_OTHERS",
93                 "value": 148
94             }
95         ]
96     },
97 ],
98 "_superOperators": [
99     {
100         "key": 0,
101         "value": true
102     },
103     {
104         "key": 2,
105         "value": true
106     },
107     {
108         "key": "ALL_OTHERS",
109         "value": false
110     }
111 ],
112 "_admin": 0
113 }
114 }
115 },
116 {
117     "key": "ALL_OTHERS",
118     "value": "EmptyAddress"
119 }
120 ]
121

```

122 After Execution:

```

123     Input = {
124         amount = 127
125         owner = 2
126     }
127     This = 0
128     Internal = {
129         __has_assertion_failure = false
130         __has_buf_overflow = false
131         __has_overflow = true
132         __has_returned = true
133         __reverted = false
134         msg = {
135             "gas": 0,
136             "sender": 0,
137             "value": 0
138         }
139     }
140     Other = {
141         __return = true
142         block = {
143             "number": 0,
144             "timestamp": 0
145         }
146     }
147     Address_Map = [

```

```

148 {
149   "key": 0,
150   "value": {
151     "contract_name": "ERC20BaseToken",
152     "balance": 0,
153     "contract": {
154       "_totalSupply": 129,
155       "_balances": [
156         {
157           "key": 0,
158           "value": 1
159         },
160         {
161           "key": 66,
162           "value": 0
163         },
164         {
165           "key": 4,
166           "value": 0
167         },
168         {
169           "key": 2,
170           "value": 107
171         },
172         {
173           "key": 128,
174           "value": 0
175         },
176         {
177           "key": "ALL_OTHERS",
178           "value": 64
179         }
180       ],
181       "_allowances": [
182         {
183           "key": 0,
184           "value": [
185             {
186               "key": 0,
187               "value": 0
188             },
189             {
190               "key": "ALL_OTHERS",
191               "value": 64
192             }
193           ]
194         },
195         {
196           "key": 1,
197           "value": [
198             {
199               "key": 0,
200               "value": 1
201             },
202             {
203               "key": "ALL_OTHERS",
204               "value": 64
205             }


```

```
206     ]
207   },
208   {
209     "key": "ALL_OTHERS",
210     "value": [
211       {
212         "key": "ALL_OTHERS",
213         "value": 148
214       }
215     ]
216   }
217 ],
218 "_superOperators": [
219   {
220     "key": 0,
221     "value": true
222   },
223   {
224     "key": 2,
225     "value": true
226   },
227   {
228     "key": "ALL_OTHERS",
229     "value": false
230   }
231 ],
232 "_admin": 0
233 }
234 }
235 },
236 {
237   "key": "ALL_OTHERS",
238   "value": "EmptyAddress"
239 }
240 ]
```

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]  
167         ][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     )
260     public
261     returns (bool success)
262     {
263         require(
264             msg.sender == owner || !_superOperators[msg.sender],
265             "msg.sender != owner && !superOperator"
266         );
267         _addAllowanceIfNeeded(owner, spender, amountNeeded);
268         return true;
269     }


```

✓ 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);

```


```
291     }
292   }
293 }
```

✓ 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

```

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 111

If method completes, integer overflow would not happen.

 02, Dec 2019

 0.69 ms

Line 295 in File ERC20BaseToken.sol

```

295     //@CTK NO_OVERFLOW

```

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 112

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

 02, Dec 2019

 0.6 ms

Line 296 in File ERC20BaseToken.sol

```

296     //@CTK NO_BUF_OVERFLOW

```

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 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),

```



```

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 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;

```

```

341     _balances[to] += amount;
342     emit Transfer(from, to, amount);
343 }

```

✗ This code violates the specification.

```

1 Counter Example:
2 Before Execution:
3   Input = {
4     amount = 192
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
13    __has_returned = false
14    __reverted = false
15    msg = {
16      "gas": 0,
17      "sender": 0,
18      "value": 0
19    }
20  }
21  Other = {
22    block = {
23      "number": 0,
24      "timestamp": 0
25    }
26  }
27  Address_Map = [
28    {
29      "key": 0,
30      "value": {
31        "contract_name": "ERC20BaseToken",
32        "balance": 0,
33        "contract": {
34          "_totalSupply": 0,
35          "_balances": [
36            {
37              "key": 0,
38              "value": 192
39            },
40            {
41              "key": 1,
42              "value": 64
43            },
44            {
45              "key": "ALL_OTHERS",
46              "value": 255
47            }
48          ],
49          "_allowances": [
50            {
51              "key": "ALL_OTHERS",
52              "value": [
53

```

```

54         "key": "ALL_OTHERS",
55         "value": 255
56     }
57 ]
58 }
59 ],
60 "_superOperators": [
61     {
62         "key": "ALL_OTHERS",
63         "value": false
64     }
65 ],
66 "_admin": 0
67 }
68 }
69 },
70 {
71     "key": "ALL_OTHERS",
72     "value": "EmptyAddress"
73 }
74 ]
75
76 After Execution:
77 Input = {
78     amount = 192
79     from = 0
80     to = 1
81 }
82 This = 0
83 Internal = {
84     __has_assertion_failure = false
85     __has_buf_overflow = false
86     __has_overflow = true
87     __has_returned = false
88     __reverted = false
89     msg = {
90         "gas": 0,
91         "sender": 0,
92         "value": 0
93     }
94 }
95 Other = {
96     block = {
97         "number": 0,
98         "timestamp": 0
99     }
100     currentBalance = 192
101 }
102 Address_Map = [
103     {
104         "key": 0,
105         "value": {
106             "contract_name": "ERC20BaseToken",
107             "balance": 0,
108             "contract": {
109                 "_totalSupply": 0,
110                 "_balances": [

```

```


112         "key": 0,
113         "value": 0
114     },
115     {
116         "key": 1,
117         "value": 0
118     },
119     {
120         "key": "ALL_OTHERS",
121         "value": 255
122     }
123 ],
124 "_allowances": [
125     {
126         "key": "ALL_OTHERS",
127         "value": [
128             {
129                 "key": "ALL_OTHERS",
130                 "value": 255
131             }
132         ]
133     }
134 ],
135 "_superOperators": [
136     {
137         "key": "ALL_OTHERS",
138         "value": false
139     }
140 ],
141 "_admin": 0
142 }
143 }
144 },
145 {
146     "key": "ALL_OTHERS",
147     "value": "EmptyAddress"
148 }
149 ]

```

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;

```


```
342     emit Transfer(from, to, amount);
343 }
```

✓ 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

```
322  //@CTK NO_ASF
```

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 119

_transfer_require

 02, Dec 2019

 11.0 ms

Line 323-327 in File ERC20BaseToken.sol

```
323  /*@CTK _transfer_require
324      @tag assume_completion
325      @post to != address(0)
326      @post _balances[from] >= amount
327  */
```

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 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:

```

3   Input = {
4       amount = 224
5       to = 2
6   }
7   This = 0
8   Internal = {
9       __has_assertion_failure = false
10      __has_buf_overflow = false
11      __has_overflow = false
12      __has_returned = false
13      __reverted = false
14      msg = {
15          "gas": 0,
16          "sender": 0,
17          "value": 0
18      }
19  }
20  Other = {
21      block = {
22          "number": 0,
23          "timestamp": 0
24      }
25  }
26  Address_Map = [
27      {
28          "key": 0,
29          "value": {
30              "contract_name": "ERC20BaseToken",
31              "balance": 0,
32              "contract": {
33                  "_totalSupply": 0,
34                  "_balances": [
35                      {
36                          "key": 2,
37                          "value": 34
38                      },
39                      {
40                          "key": "ALL_OTHERS",
41                          "value": 2
42                      }
43                  ],
44                  "_allowances": [
45                      {
46                          "key": "ALL_OTHERS",
47                          "value": [
48                              {
49                                  "key": "ALL_OTHERS",
50                                  "value": 2
51                              }
52                          ]
53                      }
54                  ],
55                  "_superOperators": [
56                      {
57                          "key": "ALL_OTHERS",
58                          "value": false
59                      }
60                  ],

```

```

61         "_admin": 0
62     }
63 }
64 },
65 {
66     "key": "ALL_OTHERS",
67     "value": "EmptyAddress"
68 }
69 ]
70
71 After Execution:
72     Input = {
73         amount = 224
74         to = 2
75     }
76     This = 0
77     Internal = {
78         __has_assertion_failure = false
79         __has_buf_overflow = false
80         __has_overflow = true
81         __has_returned = false
82         __reverted = false
83         msg = {
84             "gas": 0,
85             "sender": 0,
86             "value": 0
87         }
88     }
89     Other = {
90         block = {
91             "number": 0,
92             "timestamp": 0
93         }
94         currentTotalSupply = 0
95         newTotalSupply = 224
96     }
97     Address_Map = [
98     {
99         "key": 0,
100         "value": {
101             "contract_name": "ERC20BaseToken",
102             "balance": 0,
103             "contract": {
104                 "_totalSupply": 224,
105                 "_balances": [
106                 {
107                     "key": "ALL_OTHERS",
108                     "value": 2
109                 }
110             ],
111             "_allowances": [
112             {
113                 "key": "ALL_OTHERS",
114                 "value": [
115                 {
116                     "key": "ALL_OTHERS",
117                     "value": 2
118                 }

```




```
119         ]
120     }
121 ],
122     "_superOperators": [
123     {
124         "key": "ALL_OTHERS",
125         "value": false
126     }
127 ],
128     "_admin": 0
129 }
130 }
131 },
132 {
133     "key": "ALL_OTHERS",
134     "value": "EmptyAddress"
135 }
136 ]
```

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://
401                 // github.com/ethereum/EIPs/issues/717)
402                 _allowances[from][msg.sender] = currentAllowance - amount;
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.

```

1 Counter Example:
2 Before Execution:
3   Input = {
4     amount = 124
5     from = 128
6   }
7   This = 0
8   Internal = {
9     __has_assertion_failure = false
10    __has_buf_overflow = false
11    __has_overflow = false
12    __has_returned = false
13    __reverted = false
14    msg = {
15      "gas": 0,
16      "sender": 0,
17      "value": 0
18    }
19  }
20  Other = {
21    block = {
22      "number": 0,
23      "timestamp": 0
24    }
25  }
26  Address_Map = [
27    {
28      "key": 0,
29      "value": {
30        "contract_name": "ERC20BaseToken",
31        "balance": 0,
32        "contract": {
33          "_totalSupply": 0,
34          "_balances": [
35            {
36              "key": 0,
37              "value": 1
38            },
39            {
40              "key": 128,
41              "value": 208
42            },
43            {
44              "key": "ALL_OTHERS",
45              "value": 131
46            }
47          ],
48          "_allowances": [
49            {
50              "key": 0,
51              "value": [
52                {
53                  "key": 0,
54                  "value": 0
55                },
56                {
57                  "key": "ALL_OTHERS",
58                  "value": 131

```

```

59         }
60     ],
61 },
62 {
63     "key": 128,
64     "value": [
65         {
66             "key": 0,
67             "value": 255
68         },
69         {
70             "key": "ALL_OTHERS",
71             "value": 124
72         }
73     ]
74 },
75 {
76     "key": "ALL_OTHERS",
77     "value": [
78         {
79             "key": "ALL_OTHERS",
80             "value": 131
81         }
82     ]
83 }
84 ],
85 "_superOperators": [
86     {
87         "key": "ALL_OTHERS",
88         "value": false
89     }
90 ],
91 "_admin": 0
92 }
93 }
94 },
95 {
96     "key": "ALL_OTHERS",
97     "value": "EmptyAddress"
98 }
99 ]

```

After Execution:

```

102 Input = {
103     amount = 124
104     from = 128
105 }
106 This = 0
107 Internal = {
108     __has_assertion_failure = false
109     __has_buf_overflow = false
110     __has_overflow = true
111     __has_returned = false
112     __reverted = false
113     msg = {
114         "gas": 0,
115         "sender": 0,
116         "value": 0

```

```

117     }
118 }
119 Other = {
120     block = {
121         "number": 0,
122         "timestamp": 0
123     }
124     currentBalance = 208
125 }
126 Address_Map = [
127     {
128         "key": 0,
129         "value": {
130             "contract_name": "ERC20BaseToken",
131             "balance": 0,
132             "contract": {
133                 "_totalSupply": 132,
134                 "_balances": [
135                     {
136                         "key": 0,
137                         "value": 1
138                     },
139                     {
140                         "key": 128,
141                         "value": 84
142                     },
143                     {
144                         "key": "ALL_OTHERS",
145                         "value": 131
146                     }
147                 ],
148                 "_allowances": [
149                     {
150                         "key": 0,
151                         "value": [
152                             {
153                                 "key": 0,
154                                 "value": 0
155                             },
156                             {
157                                 "key": "ALL_OTHERS",
158                                 "value": 131
159                             }
160                         ]
161                     },
162                     {
163                         "key": 128,
164                         "value": [
165                             {
166                                 "key": 0,
167                                 "value": 255
168                             },
169                             {
170                                 "key": "ALL_OTHERS",
171                                 "value": 124
172                             }
173                         ]
174                     }

```

```


175     {
176         "key": "ALL_OTHERS",
177         "value": [
178             {
179                 "key": "ALL_OTHERS",
180                 "value": 131
181             }
182         ]
183     },
184 ],
185 "_superOperators": [
186     {
187         "key": "ALL_OTHERS",
188         "value": false
189     }
190 ],
191 "_admin": 0
192 }
193 }
194 },
195 {
196     "key": "ALL_OTHERS",
197     "value": "EmptyAddress"
198 }
199 ]

```

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://
401             // github.com/ethereum/EIPs/issues/717)
402             _allowances[from][msg.sender] = currentAllowance - amount;
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 128

Method will not encounter an assertion failure.

 02, Dec 2019

 7.56 ms

Line 375 in File ERC20BaseToken.sol

```
375  //@CTK NO_ASF
```

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 129

`__burn__require__identity`

 02, Dec 2019

 46.14 ms

Line 376-381 in File ERC20BaseToken.sol

```
376  /*@CTK __burn__require__identity
377      @tag assume_completion
378      @post amount > 0
379      @post _balances[from] >= amount
```



```

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

```

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://
401                 // github.com/ethereum/EIPs/issues/717)
402                 _allowances[from][msg.sender] = currentAllowance - amount;
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] [
387             msg.sender] >= amount
388         @post (msg.sender != from && !_superOperators[msg.sender] && _allowances[from] [
389             msg.sender] != (2**256) - 1) -> __post._allowances[from][msg.sender] ==
390             _allowances[from][msg.sender] - amount
391         @post __post._balances[from] == _balances[from] - amount
392         @post __post._totalSupply == _totalSupply - amount
393     */

```

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 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
```

```

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 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;
```

```
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.

Source Code with CertiK Labels

File ERC20ExecuteExtension.sol

```

1  pragma solidity 0.5.9;
2
3
4  contract ERC20ExecuteExtension {
5
6      /// @dev _executionAdmin != _admin so that this super power can be disabled
        independently
7      address internal _executionAdmin;
8
9      event ExecutionAdminAdminChanged(address oldAdmin, address newAdmin);
10
11     /// @notice give the address responsible for adding execution rights.
12     /// @return address of the execution administrator.
13     ///@CTK NO_OVERFLOW
14     ///@CTK NO_BUF_OVERFLOW
15     ///@CTK NO_ASF
16     /*@CTK getExecutionAdmin
17     @post __return == _executionAdmin
18     */
19     function getExecutionAdmin() external view returns (address) {
20         return _executionAdmin;
21     }
22
23     /// @notice change the execution administrator to be `newAdmin`.
24     /// @param newAdmin address of the new administrator.
25     ///@CTK NO_OVERFLOW
26     ///@CTK NO_BUF_OVERFLOW
27     ///@CTK NO_ASF
28     /*@CTK changeExecutionAdmin_require
29     @tag assume_completion
30     @post msg.sender == _executionAdmin
31     */
32     /*@CTK changeExecutionAdmin_change
33     @tag assume_completion
34     @pre msg.sender == _executionAdmin
35     @post __post._executionAdmin == newAdmin
36     */
37     function changeExecutionAdmin(address newAdmin) external {
38         require(msg.sender == _executionAdmin, "only executionAdmin can change
39             executionAdmin");
40         emit ExecutionAdminAdminChanged(_executionAdmin, newAdmin);
41         _executionAdmin = newAdmin;
42     }
43
44     mapping(address => bool) internal _executionOperators;
45     event ExecutionOperator(address executionOperator, bool enabled);
46
47     /// @notice set `executionOperator` as executionOperator: `enabled`.
48     /// @param executionOperator address that will be given/removed executionOperator
49     right.
50     /// @param enabled set whether the executionOperator is enabled or disabled.
51     ///@CTK NO_OVERFLOW
52     ///@CTK NO_BUF_OVERFLOW
53     ///@CTK NO_ASF

```

```

52  /*@CTK setExecutionOperator_require
53      @tag assume_completion
54      @post msg.sender == _executionAdmin
55  */
56  /*@CTK setExecutionOperator_change
57      @tag assume_completion
58      @pre msg.sender == _executionAdmin
59      @post __post._executionOperators[executionOperator] == enabled
60  */
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  }
69
70  /// @notice check whether address `who` is given executionOperator rights.
71  /// @param who The address to query.
72  /// @return whether the address has executionOperator rights.
73  /*@CTK NO_OVERFLOW
74  /*@CTK NO_BUF_OVERFLOW
75  /*@CTK NO_ASF
76  /*@CTK isExecutionOperator
77      @post __return == _executionOperators[who]
78  */
79  function isExecutionOperator(address who) public view returns (bool) {
80      return _executionOperators[who];
81  }
82
83  /// @notice execute on behalf of the contract.
84  /// @param to destination address fo the call.
85  /// @param gasLimit exact amount of gas to be passed to the call.
86  /// @param data the bytes sent to the destination address.
87  /// @return success whether the execution was successful.
88  /// @return returnData data resulting from the execution.
89  /*@FIXME NO_OVERFLOW
90  /*@FIXME NO_BUF_OVERFLOW
91  /*@FIXME NO_ASF
92  /*@CTK executeWithSpecificGas_require
93      @tag assume_completion
94      @post _executionOperators[msg.sender] == true
95  */
96  function executeWithSpecificGas(address to, uint256 gasLimit, bytes calldata data)
97      external returns (bool success, bytes memory returnData) {
98      require(_executionOperators[msg.sender], "only execution operators allowed to
99          execute on SAND behalf");
100      (success, returnData) = to.call.gas(gasLimit)(data);
101      assert(gasleft() > gasLimit / 63); // not enough gas provided, assert to throw
102          all gas // TODO use EIP-1930
103  }
104
105  /// @notice approve a specific amount of token for `from` and execute on behalf of
106  the contract.
107  /// @param from address of which token will be transfered.
108  /// @param to destination address fo the call.

```

```

105  /// @param amount number of tokens allowed that can be transfer by the code at `to
106  /// @param gasLimit exact amount of gas to be passed to the call.
107  /// @param data the bytes sent to the destination address.
108  /// @return success whether the execution was successful.
109  /// @return returnData data resulting from the execution.
110  //@CTK NO_OVERFLOW
111  //@CTK NO_BUF_OVERFLOW
112  //@CTK NO_ASF
113  /*@CTK approveAndExecuteWithSpecificGas_require
114   @tag assume_completion
115   @post _executionOperators[msg.sender] == true
116  */
117  function approveAndExecuteWithSpecificGas(
118      address from,
119      address to,
120      uint256 amount,
121      uint256 gasLimit,
122      bytes calldata data
123  ) external returns (bool success, bytes memory returnData) {
124      require(_executionOperators[msg.sender], "only execution operators allowed to
125          execute on SAND behalf");
126      return _approveAndExecuteWithSpecificGas(from, to, amount, gasLimit, data);
127  }
128  /// @dev the reason for this function is that charging for gas here is more gas-
129  /// @notice approve a specific amount of token for `from` and execute on behalf of
130  /// @param from address of which token will be transfered.
131  /// @param to destination address fo the call.
132  /// @param amount number of tokens allowed that can be transfer by the code at `to
133  /// @param gasLimit exact amount of gas to be passed to the call.
134  /// @param tokenGasPrice price in token for the gas to be charged.
135  /// @param baseGasCharge amount of gas charged on top of the gas used for the call
136  /// @param tokenReceiver recipient address of the token charged for the gas used.
137  /// @param data the bytes sent to the destination address.
138  /// @return success whether the execution was successful.
139  /// @return returnData data resulting from the execution.
140  //@CTK NO_OVERFLOW
141  //@CTK NO_BUF_OVERFLOW
142  //@CTK NO_ASF
143  /*@CTK approveAndExecuteWithSpecificGasAndChargeForIt_require
144   @tag assume_completion
145   @post _executionOperators[msg.sender] == true
146  */
147  function approveAndExecuteWithSpecificGasAndChargeForIt(
148      address from,
149      address to,
150      uint256 amount,
151      uint256 gasLimit,
152      uint256 tokenGasPrice,
153      uint256 baseGasCharge,
154      address tokenReceiver,
155      bytes calldata data
156  ) external returns (bool success, bytes memory returnData) {

```

```

157     uint256 initialGas = gasleft();
158     require(_executionOperators[msg.sender], "only execution operators allowed to
        execute on SAND behalf");
159     (success, returnData) = _approveAndExecuteWithSpecificGas(from, to, amount,
        gasLimit, data);
160     if (tokenGasPrice > 0) {
161         _charge(from, gasLimit, tokenGasPrice, initialGas, baseGasCharge,
            tokenReceiver);
162     }
163 }
164
165 /// @notice transfer 1amount1 token from `from` to `to` and charge the gas
    required to perform that transfer.
166 /// @param from address of which token will be transfered.
167 /// @param to destination address fo the call.
168 /// @param amount number of tokens allowed that can be transfer by the code at `to`
    .
169 /// @param gasLimit exact amount of gas to be passed to the call.
170 /// @param tokenGasPrice price in token for the gas to be charged.
171 /// @param baseGasCharge amount of gas charged on top of the gas used for the call
    .
172 /// @param tokenReceiver recipient address of the token charged for the gas used.
173 /// @return whether the transfer was successful.
174 //@CTK NO_OVERFLOW
175 //@CTK NO_BUF_OVERFLOW
176 //@CTK NO_ASF
177 /*@CTK transferAndChargeForGas_require
    @tag assume_completion
178 @post _executionOperators[msg.sender] == true
179 */
181 /*@CTK transferAndChargeForGas_return
    @tag assume_completion
182 @pre _executionOperators[msg.sender] == true
183 @post __return == true
184 */
185
186 function transferAndChargeForGas(
187     address from,
188     address to,
189     uint256 amount,
190     uint256 gasLimit,
191     uint256 tokenGasPrice,
192     uint256 baseGasCharge,
193     address tokenReceiver
194 ) external returns (bool) {
195     uint256 initialGas = gasleft();
196     require(_executionOperators[msg.sender], "only execution operators allowed to
        perfrom transfer and charge");
197     _transfer(from, to, amount);
198     if (tokenGasPrice > 0) {
199         _charge(from, gasLimit, tokenGasPrice, initialGas, baseGasCharge,
            tokenReceiver);
200     }
201     return true;
202 }
203
204 //@CTK NO_OVERFLOW
205 //@CTK NO_BUF_OVERFLOW
206 //@CTK FAIL NO_ASF

```

```

207  /*@CTK FAIL "_charge gas left lower than limit"
208      @tag assume_completion
209      @let uint256 gasCharge = initialGas - 1 + baseGasCharge
210      @post gasCharge * tokenGasPrice / gasCharge == tokenGasPrice
211  */
212  /*@CTK FAIL "_charge gas left higher than limit"
213      @tag assume_completion
214      @let uint256 gasCharge = gasLimit + baseGasCharge
215      @post gasCharge * tokenGasPrice / gasCharge == tokenGasPrice
216  */
217  function _charge(
218      address from,
219      uint256 gasLimit,
220      uint256 tokenGasPrice,
221      uint256 initialGas,
222      uint256 baseGasCharge,
223      address tokenReceiver
224  ) internal {
225      uint256 gasCharge;
226      gasCharge = initialGas - gasleft();
227      if(gasCharge > gasLimit) {
228          gasCharge = gasLimit;
229      }
230      gasCharge += baseGasCharge;
231      uint256 tokensToCharge = gasCharge * tokenGasPrice;
232      require(tokensToCharge / gasCharge == tokenGasPrice, "overflow");
233      _transfer(from, tokenReceiver, tokensToCharge);
234  }
235
236  /*@CTK NO_OVERFLOW
237  /*@CTK NO_BUF_OVERFLOW
238  /*@CTK NO_ASF
239  function _approveAndExecuteWithSpecificGas(
240      address from,
241      address to,
242      uint256 amount,
243      uint256 gasLimit,
244      bytes memory data
245  ) internal returns (bool success, bytes memory returnData) {
246
247      if (amount > 0) {
248          _addAllowanceIfNeeded(from, to, amount);
249      }
250      (success, returnData) = to.call.gas(gasLimit)(data);
251      assert(gasleft() > gasLimit / 63); // not enough gas provided, assert to throw
        all gas // TODO use EIP-1930
252  }
253
254  function _transfer(address from, address to, uint256 amount) internal;
255  function _addAllowanceIfNeeded(address owner, address spender, uint256
        amountNeeded) internal;
256  }

```

File Admin.sol

```

1  pragma solidity ^0.5.2;
2
3  contract Admin {
4

```

```

5  address internal _admin;
6
7  event AdminChanged(address oldAdmin, address newAdmin);
8
9  /// @notice gives the current administrator of this contract.
10 /// @return the current administrator of this contract.
11 ///@CTK NO_OVERFLOW
12 ///@CTK NO_BUF_OVERFLOW
13 ///@CTK NO_ASF
14 /*@CTK getAdmin
15   @tag assume_completion
16   @post __return == _admin
17 */
18 function getAdmin() external view returns (address) {
19     return _admin;
20 }
21
22 /// @notice change the administrator to be `newAdmin`.
23 /// @param newAdmin address of the new administrator.
24 ///@CTK NO_OVERFLOW
25 ///@CTK NO_BUF_OVERFLOW
26 ///@CTK NO_ASF
27 /*@CTK changeAdmin_requirement
28   @tag assume_completion
29   @post msg.sender == _admin
30 */
31 /*@CTK changeAdmin_change
32   @tag assume_completion
33   @pre msg.sender == _admin
34   @post __post._admin == newAdmin
35 */
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 }
41
42 modifier onlyAdmin() {
43     require (msg.sender == _admin, "only admin allowed");
44     _;
45 }
46
47 }
```

File SuperOperators.sol

```

1  pragma solidity ^0.5.2;
2
3  //import "../Admin.sol";
4  import "../sandbox-private-contracts/contracts_common/src/BaseWithStorage/Admin.sol";
5
6  contract SuperOperators is Admin {
7
8      mapping(address => bool) internal _superOperators;
9
10     event SuperOperator(address superOperator, bool enabled);
11
12     /// @notice Enable or disable the ability of `superOperator` to transfer tokens of
        all (superOperator rights).
```

```

13  /// @param superOperator address that will be given/removed superOperator right.
14  /// @param enabled set whether the superOperator is enabled or disabled.
15  //@CTK NO_OVERFLOW
16  //@CTK NO_BUF_OVERFLOW
17  //@CTK NO_ASF
18  /*@CTK setSuperOperator_admin
19      @tag assume_completion
20      @inv msg.sender == _admin
21  */
22  /*@CTK setSuperOperator_change
23      @tag assume_completion
24      @pre msg.sender == _admin
25      @post __post._superOperators[superOperator] == enabled
26  */
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  }
35
36  /// @notice check whether address `who` is given superOperator rights.
37  /// @param who The address to query.
38  /// @return whether the address has superOperator rights.
39  //@CTK NO_OVERFLOW
40  //@CTK NO_BUF_OVERFLOW
41  //@CTK NO_ASF
42  /*@CTK isSuperOperator
43      @tag assume_completion
44      @post __return == _superOperators[who]
45  */
46  function isSuperOperator(address who) public view returns (bool) {
47      return _superOperators[who];
48  }
49  }

```

File Sand.sol

```

1  pragma solidity 0.5.9;
2
3  import "../sandbox-private-contracts/src/Sand/erc20/ERC20ExecuteExtension.sol";
4  import "../sandbox-private-contracts/src/Sand/erc20/ERC20BaseToken.sol";
5  import "../sandbox-private-contracts/src/Sand/erc20/ERC20BasicApproveExtension.sol";
6
7  contract Sand is ERC20ExecuteExtension, ERC20BasicApproveExtension, ERC20BaseToken {
8
9      //@CTK NO_OVERFLOW
10     //@CTK NO_BUF_OVERFLOW
11     //@CTK NO_ASF
12     /*@CTK Sand_require
13         @tag assume_completion
14         @post beneficiary != address(0)
15         @post _totalSupply + 30000000000000000000000000000000 > _totalSupply
16     */
17     /*@CTK Sand_change
18         @tag assume_completion
19         @pre beneficiary != address(0)

```

File ERC20BaseToken.sol

page 109


```

16  */
17  function totalSupply() public view returns (uint256) {
18      return _totalSupply;
19  }
20
21  /// @notice Gets the balance of `owner`.
22  /// @param owner The address to query the balance of.
23  /// @return The amount owned by `owner`.
24  //@CTK NO_OVERFLOW
25  //@CTK NO_BUF_OVERFLOW
26  //@CTK NO_ASF
27  /*@CTK balanceOf
28      @post __return == _balances[owner]
29  */
30  function balanceOf(address owner) public view returns (uint256) {
31      return _balances[owner];
32  }
33
34  /// @notice gets allowance of `spender` for `owner`'s tokens.
35  /// @param owner address whose token is allowed.
36  /// @param spender address allowed to transfer.
37  /// @return the amount of token `spender` is allowed to transfer on behalf of `
38  owner`.
39  //@CTK NO_OVERFLOW
40  //@CTK NO_BUF_OVERFLOW
41  //@CTK NO_ASF
42  /*@CTK allowance
43      @post __return == _allowances[owner][spender]
44  */
45  function allowance(address owner, address spender)
46      public
47      view
48      returns (uint256)
49  {
50      return _allowances[owner][spender];
51  }
52
53  /// @notice returns the number of decimals for that token.
54  /// @return the number of decimals.
55  /*@CTK decimals
56      @post __return == 18
57  */
58  function decimals() public view returns (uint8) {
59      return uint8(18);
60  }
61
62  /// @notice Transfer `amount` tokens to `to`.
63  /// @param to the recipient address of the tokens transferred.
64  /// @param amount the number of tokens transferred.
65  /// @return true if success.
66  //@CTK FAIL_NO_OVERFLOW
67  //@CTK NO_BUF_OVERFLOW
68  //@CTK NO_ASF
69  /*@CTK transfer_require
70      @tag assume_completion
71      @post to != address(0)
72      @post _balances[msg.sender] >= amount
73  */

```

```

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  */
82  function transfer(address to, uint256 amount)
83      public
84      returns (bool)
85  {
86      _transfer(msg.sender, to, amount);
87      return true;
88  }
89
90  /// @notice Transfer `amount` tokens from `from` to `to`.
91  /// @param from whose token it is transferring from.
92  /// @param to the recipient address of the tokens transferred.
93  /// @param amount the number of tokens transferred.
94  /// @return true if success.
95  //@CTK FAIL NO_OVERFLOW
96  //@CTK NO_BUF_OVERFLOW
97  //@CTK NO_ASF
98  /*@CTK transferFrom_require
99      @tag assume_completion
100      @post to != address(0)
101      @post _balances[from] >= amount
102      @post (msg.sender != from && !_superOperators[msg.sender] && _allowances[from][
103          msg.sender] != (2**256) - 1) -> _allowances[from][msg.sender] >= amount
104  */
105  /*@CTK transferFrom_change
106      @tag assume_completion
107      @pre to != address(0)
108      @pre _balances[from] >= amount
109      @pre (msg.sender != from && !_superOperators[msg.sender] && _allowances[from][msg
110          .sender] != (2**256) - 1) -> _allowances[from][msg.sender] >= amount
111      @pre from != to
112      @post (msg.sender != from && !_superOperators[msg.sender] && _allowances[from][
113          msg.sender] != (2**256) - 1) -> __post._allowances[from][msg.sender] ==
114          _allowances[from][msg.sender] - amount
115      @post __post._balances[from] == _balances[from] - amount
116      @post __post._balances[to] == _balances[to] + amount
117      @post __return == true
118  */
119  function transferFrom(address from, address to, uint256 amount)
120      public
121      returns (bool)
122  {
123      if (msg.sender != from && !_superOperators[msg.sender]) {
124          uint256 currentAllowance = _allowances[from][msg.sender];
125          if (currentAllowance != (2**256) - 1) {
126              // save gas when allowance is maximal by not reducing it (see https://
127                  github.com/ethereum/EIPs/issues/717)
128              require(currentAllowance >= amount, "Not enough funds allowed");
129              _allowances[from][msg.sender] = currentAllowance - amount;
130          }
131      }
132  }

```

```

126     }
127     _transfer(from, to, amount);
128     return true;
129 }
130
131 /// @notice burn `amount` tokens.
132 /// @param amount the number of tokens to burn.
133 /// @return true if success.
134 //@CTK FAIL NO_OVERFLOW
135 //@CTK NO_BUF_OVERFLOW
136 //@CTK NO_ASF
137 /*@CTK burn_require
138   @tag assume_completion
139   @post amount > 0
140   @post _balances[msg.sender] >= amount
141 */
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 */
150 function burn(uint256 amount) external returns (bool) {
151     _burn(msg.sender, amount);
152     return true;
153 }
154
155 /// @notice burn `amount` tokens from `owner`.
156 /// @param owner address whose token is to burn.
157 /// @param amount the number of token to burn.
158 /// @return true if success.
159 //@CTK FAIL NO_OVERFLOW
160 //@CTK NO_BUF_OVERFLOW
161 //@CTK NO_ASF
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]
167     [msg.sender] >= amount
168 */
169 /*@CTK burnFor_change
170   @tag assume_completion
171   @pre amount > 0
172   @pre _balances[owner] >= amount
173   @pre (msg.sender != owner && !_superOperators[msg.sender]) -> _allowances[owner] [
174     msg.sender] >= amount
175   @post (msg.sender != owner && !_superOperators[msg.sender] && _allowances[owner] [
176     msg.sender] != (2*256) - 1) -> __post._allowances[owner][msg.sender] ==
177     _allowances[owner][msg.sender] - amount
178   @post __post._balances[owner] == _balances[owner] - amount
179   @post __post._totalSupply == _totalSupply - amount
180 */
181 function burnFor(address owner, uint256 amount) external returns (bool) {
182     _burn(owner, amount);
183     return true;
184 }

```

```

180 }
181
182 /// @notice approve `spender` to transfer `amount` tokens.
183 /// @param spender address to be given rights to transfer.
184 /// @param amount the number of tokens allowed.
185 /// @return true if success.
186 //@FIXME NO_OVERFLOW
187 //@FIXME NO_BUF_OVERFLOW
188 //@FIXME NO_ASF
189 /*@FIXME approve_require
190   @tag assume_completion
191   @post msg.sender != address(0)
192   @post spender != address(0)
193 */
194 /*@FIXME approve_change
195   @tag assume_completion
196   @pre msg.sender != address(0)
197   @pre spender != address(0)
198   @post __post._allowances[msg.sender][spender] == amount
199 */
200 function approve(address spender, uint256 amount)
201     public
202     returns (bool success)
203 {
204     _approveFor(msg.sender, spender, amount);
205     return true;
206 }
207
208 /// @notice approve `spender` to transfer `amount` tokens from `owner`.
209 /// @param owner address whose token is allowed.
210 /// @param spender address to be given rights to transfer.
211 /// @param amount the number of tokens allowed.
212 /// @return true if success.
213 //@FIXME NO_OVERFLOW
214 //@FIXME NO_BUF_OVERFLOW
215 //@FIXME NO_ASF
216 /*@FIXME approveFor_require
217   @tag assume_completion
218   @post msg.sender == owner || _superOperators[msg.sender]
219   @post owner != address(0)
220   @post spender != address(0)
221 */
222 /*@FIXME approveFor_change
223   @tag assume_completion
224   @pre msg.sender == owner || _superOperators[msg.sender]
225   @pre owner != address(0)
226   @pre spender != address(0)
227   @post __post._allowances[owner][spender] == amount
228   @post __return == true
229 */
230 function approveFor(address owner, address spender, uint256 amount)
231     public
232     returns (bool success)
233 {
234     require(
235         msg.sender == owner || _superOperators[msg.sender],
236         "msg.sender != owner && !superOperator"
237     );

```

```

238     _approveFor(owner, spender, amount);
239     return true;
240 }
241
242 // @FIXME NO_OVERFLOW
243 // @CTK NO_BUF_OVERFLOW
244 // @CTK NO_ASF
245 /* @CTK addAllowanceIfNeeded_require
246     @tag assume_completion
247     @post msg.sender == owner || !_superOperators[msg.sender]
248     @post (amountNeeded > 0 && !_superOperators[spender] && _allowances[owner][
249         spender] < amountNeeded) -> owner != address(0)
250     @post (amountNeeded > 0 && !_superOperators[spender] && _allowances[owner][
251         spender] < amountNeeded) -> spender != address(0)
252 */
253 /* @CTK addAllowanceIfNeeded_change
254     @tag assume_completion
255     @pre msg.sender == owner || !_superOperators[msg.sender]
256     @pre (amountNeeded > 0 && !_superOperators[spender] && _allowances[owner][spender]
257         < amountNeeded) -> owner != address(0)
258     @pre (amountNeeded > 0 && !_superOperators[spender] && _allowances[owner][spender]
259         < amountNeeded) -> spender != address(0)
260     @post (amountNeeded > 0 && !_superOperators[spender] && _allowances[owner][
261         spender] < amountNeeded) -> __post._allowances[owner][spender] ==
262         amountNeeded
263 */
264 function addAllowanceIfNeeded(address owner, address spender, uint256 amountNeeded
265 )
266     public
267     returns (bool success)
268 {
269     require(
270         msg.sender == owner || !_superOperators[msg.sender],
271         "msg.sender != owner && !superOperator"
272     );
273     _addAllowanceIfNeeded(owner, spender, amountNeeded);
274     return true;
275 }
276
277 // @CTK NO_OVERFLOW
278 // @CTK NO_BUF_OVERFLOW
279 // @CTK NO_ASF
280 /* @CTK _addAllowanceIfNeeded_require
281     @tag assume_completion
282     @post (amountNeeded > 0 && !_superOperators[spender] && _allowances[owner][
283         spender] < amountNeeded) -> owner != address(0)
284     @post (amountNeeded > 0 && !_superOperators[spender] && _allowances[owner][
285         spender] < amountNeeded) -> spender != address(0)
286 */
287 /* @CTK _addAllowanceIfNeeded_change
288     @tag assume_completion
289     @pre (amountNeeded > 0 && !_superOperators[spender] && _allowances[owner][spender]
290         < amountNeeded) -> owner != address(0)
291     @pre (amountNeeded > 0 && !_superOperators[spender] && _allowances[owner][spender]
292         < amountNeeded) -> spender != address(0)
293     @post (amountNeeded > 0 && !_superOperators[spender] && _allowances[owner][
294         spender] < amountNeeded) -> __post._allowances[owner][spender] ==
295         amountNeeded

```

```

283  */
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  }
294
295  //@CTK NO_OVERFLOW
296  //@CTK NO_BUF_OVERFLOW
297  //@CTK NO_ASF
298  /*@CTK _approveFor_require
299      @tag assume_completion
300      @post owner != address(0)
301      @post spender != address(0)
302  */
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  */
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  }
319
320  //@CTK FAIL NO_OVERFLOW
321  //@CTK NO_BUF_OVERFLOW
322  //@CTK NO_ASF
323  /*@CTK _transfer_require
324      @tag assume_completion
325      @post to != address(0)
326      @post _balances[from] >= amount
327  */
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  */
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 }
344
345 //@CTK FAIL NO_OVERFLOW
346 //@CTK NO_BUF_OVERFLOW
347 //@CTK NO_ASF
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 */
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 */
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 }
372
373 //@CTK FAIL NO_OVERFLOW
374 //@CTK NO_BUF_OVERFLOW
375 //@CTK NO_ASF
376 /*@CTK _burn_require_identity
377   @tag assume_completion
378   @post amount > 0
379   @post _balances[from] >= amount
380   @post (msg.sender != from && !_superOperators[msg.sender]) -> _allowances[from][
381     msg.sender] >= amount
382 */
383 /*@CTK _burn_change
384   @tag assume_completion
385   @pre amount > 0
386   @pre _balances[from] >= amount
387   @pre (msg.sender != from && !_superOperators[msg.sender]) -> _allowances[from][
388     msg.sender] >= amount
389   @post (msg.sender != from && !_superOperators[msg.sender] && _allowances[from][
390     msg.sender] != (2**256) - 1) -> __post._allowances[from][msg.sender] ==
391     _allowances[from][msg.sender] - amount
392   @post __post._balances[from] == _balances[from] - amount
393   @post __post._totalSupply == _totalSupply - amount
394 */
395 function _burn(address from, uint256 amount) internal {
396     require(amount > 0, "cannot burn 0 tokens");
397     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://
401         // github.com/ethereum/EIPs/issues/717)
402         _allowances[from][msg.sender] = currentAllowance - amount;
403     }
404 }
405
406 uint256 currentBalance = _balances[from];
407 require(currentBalance >= amount, "Not enough funds");
408 _balances[from] = currentBalance - amount;
409 _totalSupply -= amount;
410 emit Transfer(from, address(0), amount);
411 }

```

File ERC20BasicApproveExtension.sol

```

1  pragma solidity 0.5.9;
2
3  contract ERC20BasicApproveExtension {
4
5      /// @notice approve `target` to spend `amount` and call it with data.
6      /// @param target address to be given rights to transfer and destination of the
7      /// call.
8      /// @param amount the number of tokens allowed.
9      /// @param data bytes for the call.
10     /// @return data of the call.
11     ///@CTK NO_OVERFLOW
12     ///@CTK NO_BUF_OVERFLOW
13     ///@CTK NO_ASF
14     function approveAndCall(
15         address target,
16         uint256 amount,
17         bytes calldata data
18     ) external payable returns (bytes memory) {
19         require(
20             doFirstParamEqualsAddress(data, msg.sender),
21             "first param != sender"
22         );
23         _approveFor(msg.sender, target, amount);
24
25         // solium-disable-next-line security/no-call-value
26         (bool success, bytes memory returnData) = target.call.value(msg.value)(data);
27         require(success, string(returnData));
28         return returnData;
29     }
30
31     /// @notice temporarily approve `target` to spend `amount` and call it with data.
32     /// Previous approvals remains unchanged.
33     /// @param target destination of the call, allowed to spend the amount specified
34     /// @param amount the number of tokens allowed to spend.
35     /// @param data bytes for the call.
36     /// @return data of the call.
37     ///@CTK NO_OVERFLOW

```



```

36 // @CTK NO_BUF_OVERFLOW
37 // @CTK NO_ASF
38 function paidCall(
39     address target,
40     uint256 amount,
41     bytes calldata data
42 ) external payable returns (bytes memory) {
43     require(
44         doFirstParamEqualsAddress(data, msg.sender),
45         "first param != sender"
46     );
47
48     if (amount > 0) {
49         _addAllowanceIfNeeded(msg.sender, target, amount);
50     }
51
52     // solium-disable-next-line security/no-call-value
53     (bool success, bytes memory returnData) = target.call.value(msg.value)(data);
54     require(success, string(returnData));
55
56     return returnData;
57 }
58
59 function _approveFor(address owner, address target, uint256 amount) internal;
60 function _addAllowanceIfNeeded(address owner, address spender, uint256
    amountNeeded) internal;
61
62 // @CTK NO_OVERFLOW
63 // @CTK NO_BUF_OVERFLOW
64 // @CTK NO_ASF
65 /* @CTK doFirstParamEqualsAddress_require
66     @tag assume_completion
67     @post (data.length < (36 + 32)) -> !__return
68 */
69 function doFirstParamEqualsAddress(bytes memory data, address _address)
70     internal
71     pure
72     returns (bool)
73 {
74     if (data.length < (36 + 32)) {
75         return false;
76     }
77     uint256 value;
78     /* @CTK "Load 32 bytes from mem[data+36]"
79     @var uint value
80     @post value == uint256(_address)
81 */
82     assembly {
83         value := mload(add(data, 36))
84     }
85     return value == uint256(_address);
86 }
87 }

```

