Cyberscope

A TAC Security Company

Audit Report

The Second Best

July 2025

Network	BSC
Address	0xBf006b86C09aa50C15793680e27f41F00e13c214
Audited by	© cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	MEM	Missing Error Messages	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
● Critical	Highly Likely / High Impact
● Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
● Minor / Informative	Unlikely / Low to no Impact

Review

Contract Name	StandardToken
Compiler Version	v0.8.20+commit.a1b79de6
Optimization	200 runs
Explorer	https://bscscan.com/address/0xbf006b86c09aa50c15793680e27f41f00e13c214
Address	0xbf006b86c09aa50c15793680e27f41f00e13c214
Network	BSC
Symbol	2ND
Decimals	18
Total Supply	21.000.001

Audit Updates

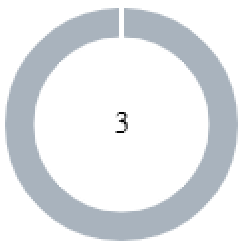
Initial Audit	30 Jul 2025

Source Files

Filename	SHA256
contracts/StandardToken.sol	7a8412f412766a54b9a8a9733d981ef4d84490ccae89be8f120c51bc7535802e
@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol	2d3d7dc6e116cb8ebb8517208141cb3d0950b337a285f15f8476ec3df29d824e
@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol	db92fc1b515decad3a783b1422190877d2d70b907c6e36fb0998d9465aee42db

@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol	a2c4e5c274a586f145d278293ae33198cd8f412ab7e6d26f2394c8949b32b24b
@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol	2d9e57d2a4b0775334be2968019c1939377d45b69e8b724fe6bb80af47e28419
@openzeppelin/contracts/token/ERC20/IERC20.sol	7ebde70853ccafcf1876900dad458f46eb9444d591d39bfc58e952e2582f5587

Findings Breakdown



🟥	Critical	0
🟧	Medium	0
●	Minor / Informative	3

	Severity	Unresolved	Acknowledged	Resolved	Other
🟥	Critical	0	0	0	0
🟧	Medium	0	0	0	0
●	Minor / Informative	3	0	0	0

MEM - Missing Error Messages

Criticality	Minor / Informative
Location	contracts/StandardToken.sol#L179,229,261
Status	Unresolved

Description

The contract is missing error messages. Specifically, there are no error messages to accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

The contract includes functionality that performs value subtraction. If the subtracted value exceeds the original value, the function will revert. However, the contract lacks error messages to inform users of the reason for the revert.

```
_approve(sender, _msgSender(), _allowances[sender][_msgSender()] -
amount);
_approve(_msgSender(), spender, _allowances[_msgSender()][spender] -
subtractedValue);
...
_balances[sender] = _balances[sender] - amount;
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/StandardToken.sol#L296,338
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
_burn(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: burn from the zero address");
    _beforeTokenTransfer(account, address(0), amount);
    _balances[account] = _balances[account] - amount;
    _totalSupply = _totalSupply - amount;
    emit Transfer(account, address(0), amount);
}

function _setupDecimals(uint8 decimals_) internal virtual {
    _decimals = decimals_;
}
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/StandardToken.sol#L2
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.0;
```

Recommendation

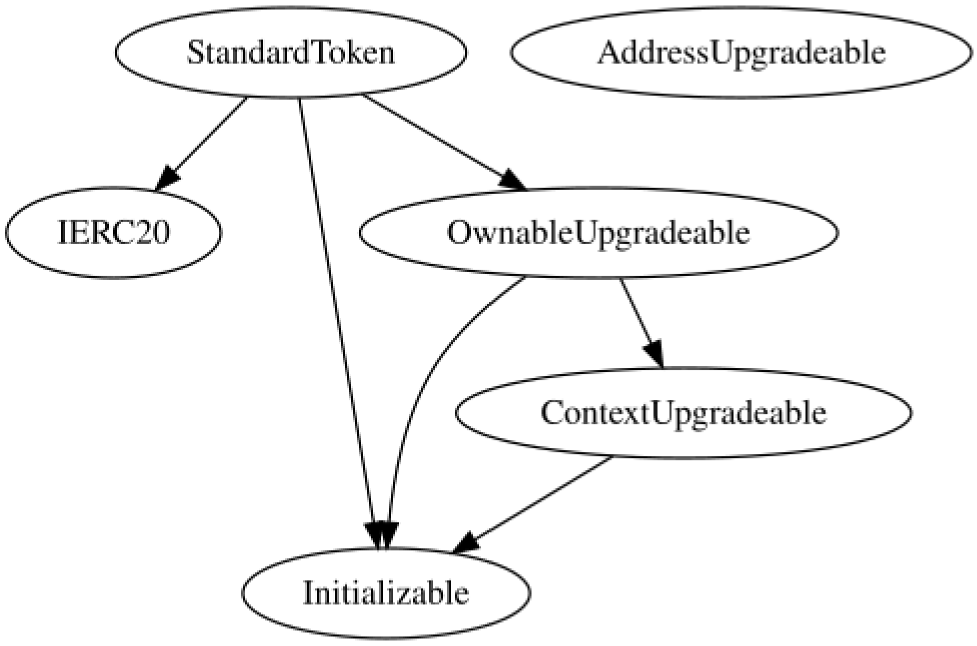
The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

Functions Analysis

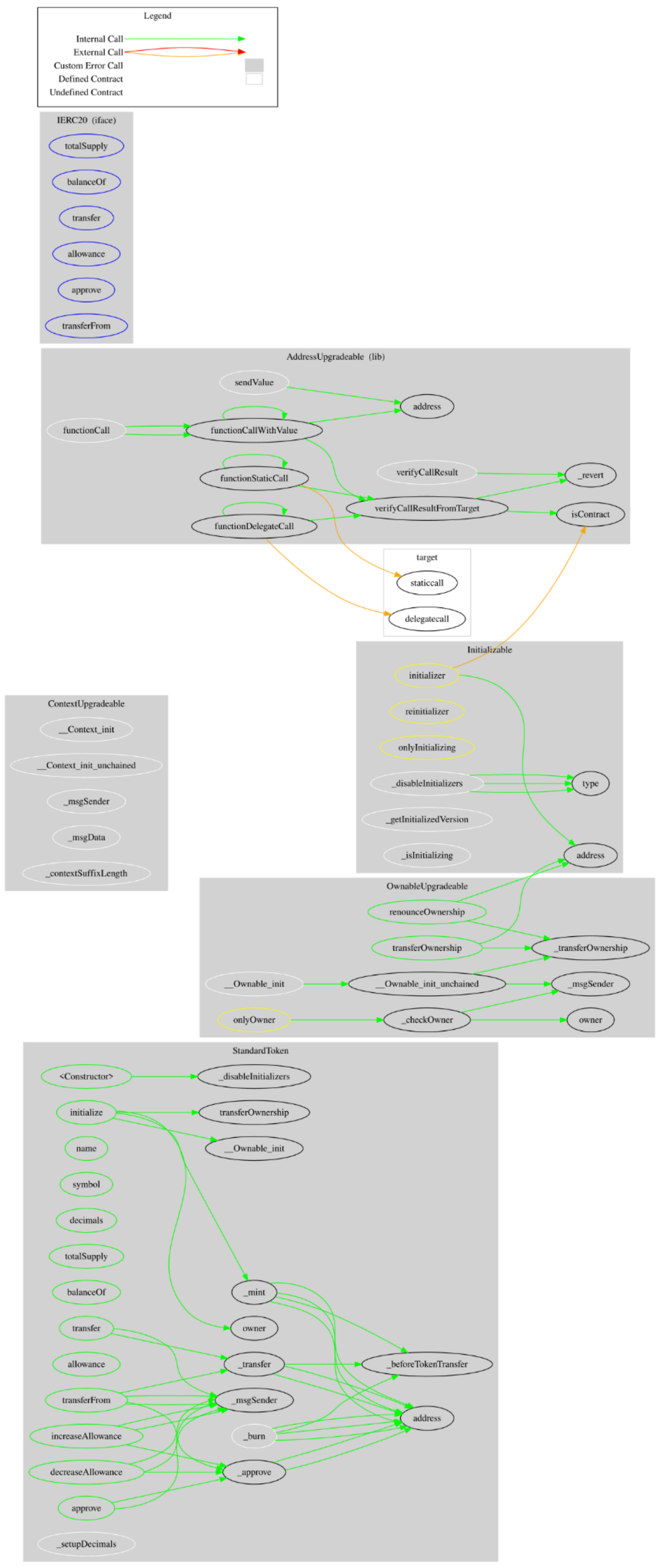
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
StandardToken	Implementation	IERC20, Initializable, OwnableUpgradeable		
		Public	✓	-
	initialize	Public	✓	initializer
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_setupDecimals	Internal	✓	

	_beforeTokenTransfer	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
AddressUpgradeable (lib)
sendValue
address
functionCall
functionCallWithValue
functionStaticCall
verifyCallResult
_revert
verifyCallResultFromTarget
isContract
functionDelegateCall
target
staticcall
delegatecall
Initializable
initializer
reinitializer
onlyInitializing
_disableInitializers
type
_getInitializedVersion
_isInitializing
address
ContextUpgradeable
__Context_init
__Context_init_unchained
_msgSender
_msgData
_contextSuffixLength
OwnableUpgradeable
renounceOwnership
transferOwnership
_transferOwnership
__Ownable_init
__Ownable_init_unchained
_msgSender
onlyOwner
_checkOwner
owner
StandardToken
<Constructor>
_disableInitializers
initialize
transferOwnership
name
__Ownable_init
symbol
decimals
totalSupply
balanceOf
_mint
transfer
owner
allowance
_transfer
_beforeTokenTransfer
transferFrom
_msgSender
increaseAllowance
address
_burn
decreaseAllowance
_approve
approve
_setupDecimals

Summary

The Second Best contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. The Second Best is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues.

Disclaimer

About Cyberscope

Cyberscope is a TAC blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

A **TAC Security** Company

The Cyberscope team

cyberscope.io