Cyberscope

Audit Report

TempestToken

December 2023

Network	AVAX
Address	0xd24b6e3f5ac53bdec4141a33963357fa8547bbf7
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
●	IRC	Inheritance Redundancy Code	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	TempestToken
Compiler Version	v0.8.18+commit.87f61d96
Optimization	200 runs
Explorer	https://snowtrace.io/address/0xd24b6e3f5ac53bdec4141a33963357fa8547bbf7
Address	0xd24b6e3f5ac53bdec4141a33963357fa8547bbf7
Network	AVAX
Symbol	TMPST
Decimals	18
Total Supply	22,000,000

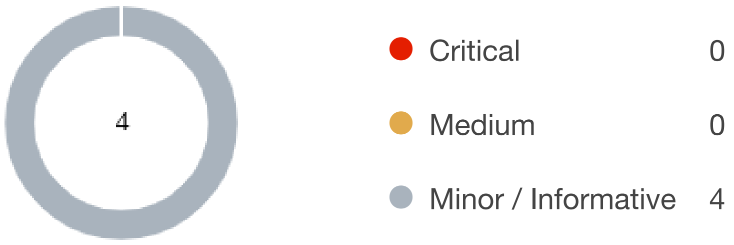
Audit Updates

Initial Audit	14 Dec 2023

Source Files

Filename	SHA256
contracts/TempestToken.sol	c25ddb57a4aaff850cd3f3352447008494aa77d8067291ee234f9bb41cf760b5

Findings Breakdown



Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	0	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	4	0	0	0

IRC - Inheritance Redundancy Code

Criticality	Minor / Informative
Location	contracts/TempestToken.sol#L603,607
Status	Unresolved

Description

The TempestToken contract, inherits from both `ERC20` and `Ownable` contracts. The use of the Ownable contract inherently transfers ownership to the contract creator (msg.sender) during contract deployment. This is a standard feature of the Ownable contract. However, in the `TempestToken` constructor, there is an explicit call to `transferOwnership(msg.sender)`, which is redundant since the ownership is already set to `msg.sender` by the `Ownable` contract during its initialization. Furthermore, the TempestToken contract includes the overridden `transferOwnership` function, which is unnecessary since the Ownable contract already provides an implementation of this function. This redundancy not only increases the complexity of the contract but also potentially raises the gas cost unnecessarily.

```
contract TempestToken is ERC20, Ownable {
    constructor() ERC20("Tempest", "TMPST") {
        uint256 totalSupply = 22_000_000 * 10**uint256(decimals()); //
Total supply is 22 million tokens
        _mint(msg.sender, totalSupply);
        transferOwnership(msg.sender);
    }

    // Function to transfer ownership to a new owner
    function transferOwnership(address newOwner) public override
onlyOwner {
        _transferOwnership(newOwner);
    }
}
```

Recommendation

It is recommended to remove the explicit call to `transferOwnership(msg.sender)` in the constructor of the contract. This action is redundant as the Ownable contract, which

TempestToken inherits, already sets the contract creator as the owner upon deployment. Additionally, it is advisable to eliminate the overridden `transferOwnership` function in the contract. Retaining the original function from the Ownable contract is sufficient and helps in reducing contract complexity and potential errors, while also optimizing for gas efficiency. Simplifying the contract in this manner ensures adherence to best practices in smart contract development, particularly in terms of code efficiency and clarity.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/TempestToken.sol#L502
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.

```
function _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero
address");

        _beforeTokenTransfer(account, address(0), amount);

        uint256 accountBalance = _balances[account];
...
            _totalSupply -= amount;
        }

        emit Transfer(account, address(0), amount);

        _afterTokenTransfer(account, address(0), amount);
    }
```

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.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	contracts/TempestToken.sol#L601
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
uint256 totalSupply = 22_000_000 * 10**uint256(decimals())
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/TempestToken.sol#L7,34,119,200,230,595
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.

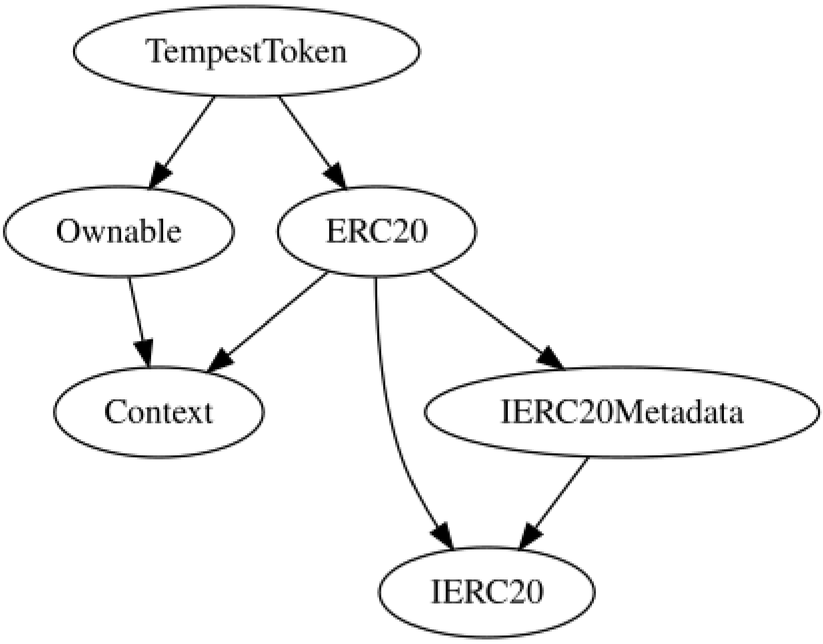
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-

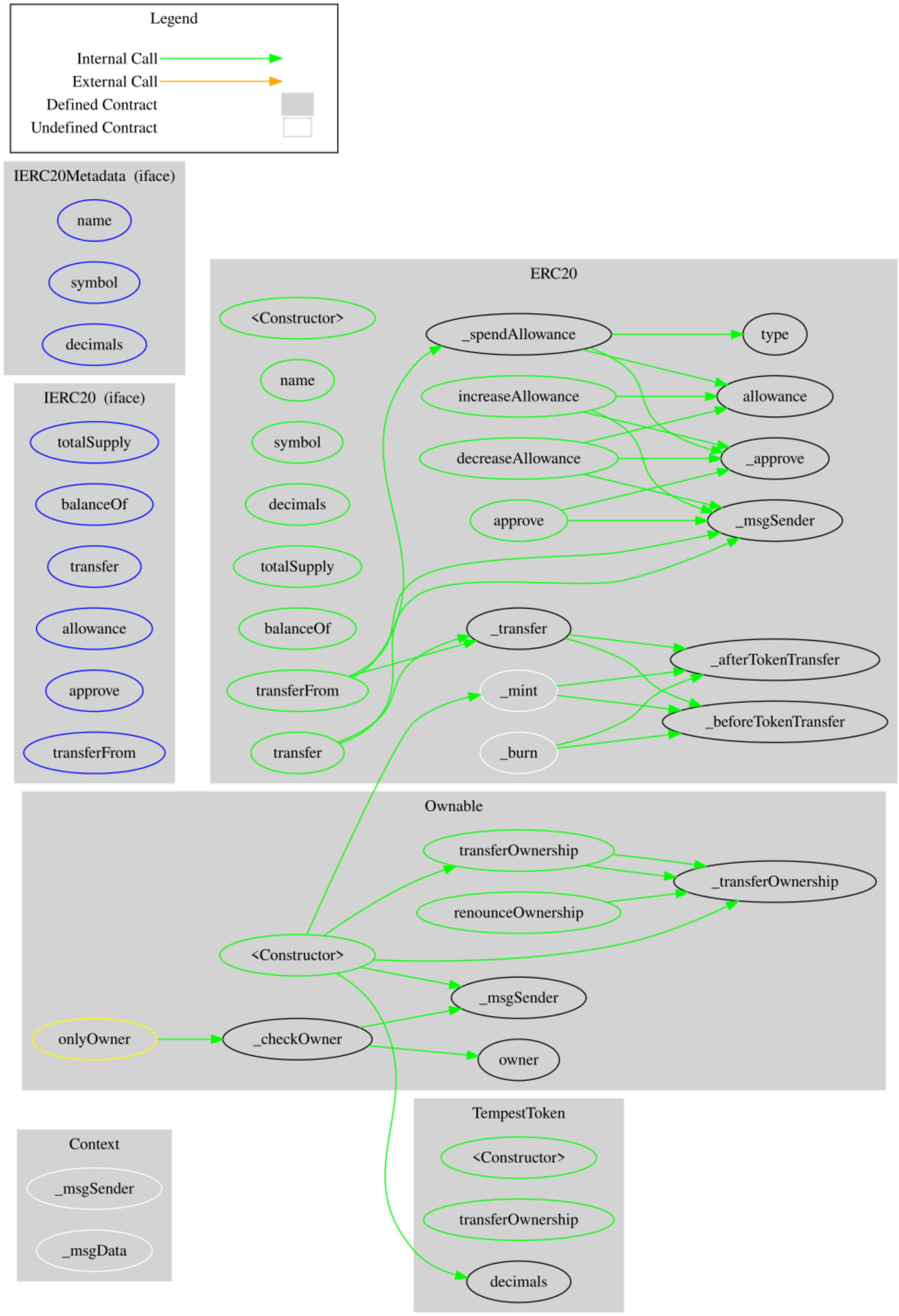
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	

	_burn	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
TempestToken	Implementation	ERC20, Ownable		
		Public	✓	ERC20
	transferOwnership	Public	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transferFrom
transfer
_spendAllowance
increaseAllowance
decreaseAllowance
approve
_transfer
_mint
_burn
type
allowance
_approve
_msgSender
_afterTokenTransfer
_beforeTokenTransfer
Ownable
transferOwnership
renounceOwnership
<Constructor>
onlyOwner
_checkOwner
_transferOwnership
_msgSender
owner
Context
_msgSender
_msgData
TempestToken
<Constructor>
transferOwnership
decimals

Summary

TempestToken contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. TempestToken is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an “endorsement” or “disapproval” of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a 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.

The Cyberscope team

https://www.cyberscope.io