Cyberscope

Audit Report
Temple Fi

March 2023

Type	ERC20
Network	ARBITRUM
Address	0x2B33B06724dFE71D2C9a43bF00Ab73D3f2b67D4d
Audited by	© cyberscope

Table of Contents

Table of Contents 1
Review 1
Audit Updates 2
Source Files 2
Analysis 3
MT - Mints Tokens 4
Description 4
Recommendation 5
Diagnostics 5
RSML - Redundant SafeMath Library 6
Description 6
Recommendation 7
L04 - Conformance to Solidity Naming Conventions 7
Description 7
Recommendation 8
L19 - Stable Compiler Version 9
Description 9
Recommendation 10
Functions Analysis 10
Inheritance Graph 13
Flow Graph 13
Summary 13
Disclaimer 16
About Cyberscope 17

Review

Contract Name	TempleToken
Compiler Version	v0.8.18+commit.87f61d96
Optimization	200 runs
Explorer	https://arbiscan.io/address/0x2b33b06724dfe71d2c9a43bf00ab73d3f2b67d4d
Address	0x2b33b06724dfe71d2c9a43bf00ab73d3f2b67d4d
Network	ARBITRUM
Symbol	Temple
Decimals	18
Total Supply	100,000

Audit Updates

Initial Audit	17 Mar 2023

Source Files

Filename	SHA256
Address.sol	0db3442dc03ed78816b9c8588a34fcddc606c284b25ea13dfa3b4a94ebdb1c59
Context.sol	6ee66d1e4693ec63d29393cc2c83594798475ad0eeabcb0951a35780ef003113
ERC20.sol	594305b7f4198c570b0ffd3de60be7d1fec934004c9a868bf3cdd060cd05e2d9
IERC20.sol	7ddf775e2d08dbdbc719ffc0148fb3f6237593cd4a1fc14fa407a2a207c7f790
Ownable.sol	65a10e82e5dc1737e237f2865af3258f947d5a116f2446dfeb12c77e5a55ed9b
SafeERC20.sol	fb39e78e5e2d936716d68ec64389275078b294782e55654d6073983139f185d6
SafeMath.sol	592f8bd5c5998efa4d8e61d95a49428b758fa4174236260b86d442ac245aac1d
TempleToken.sol	1e055ffd16d1c0cad0cda6f705c93cace89e5205279c5b31b6791aff335b2653

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OCTD	Transfers Contract's Tokens	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Unresolved
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

MT - Mints Tokens

Criticality	Critical
Location	TempleToken.sol#L15,19
Status	Unresolved

Description

The contract owner has the authority to mint tokens. The owner may take advantage of it by calling the `mint` and/or `mintFor` functions. As a result, the contract tokens will be highly inflated.

```
function mint(uint256 _amount) public override onlyOwner returns (bool) {
    return mintFor(address(this), _amount);
}

function mintFor(
    address _address,
    uint256 _amount
) public onlyOwner returns (bool) {
    _mint(_address, _amount);
    require(totalSupply() <= maxSupply, "reach max supply");
    return true;
}
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	RSML	Redundant SafeMath Library	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L19	Stable Compiler Version	Unresolved

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	TempleToken.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, and overhead and increases gas consumption unnecessarily.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	TempleToken.sol#L8,15,20,21,29
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
uint256 public constant maxSupply = 1_000_000_000e18
uint256 _amount
address _address
address _to
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	TempleToken.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 ^0.8.17;
```

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.

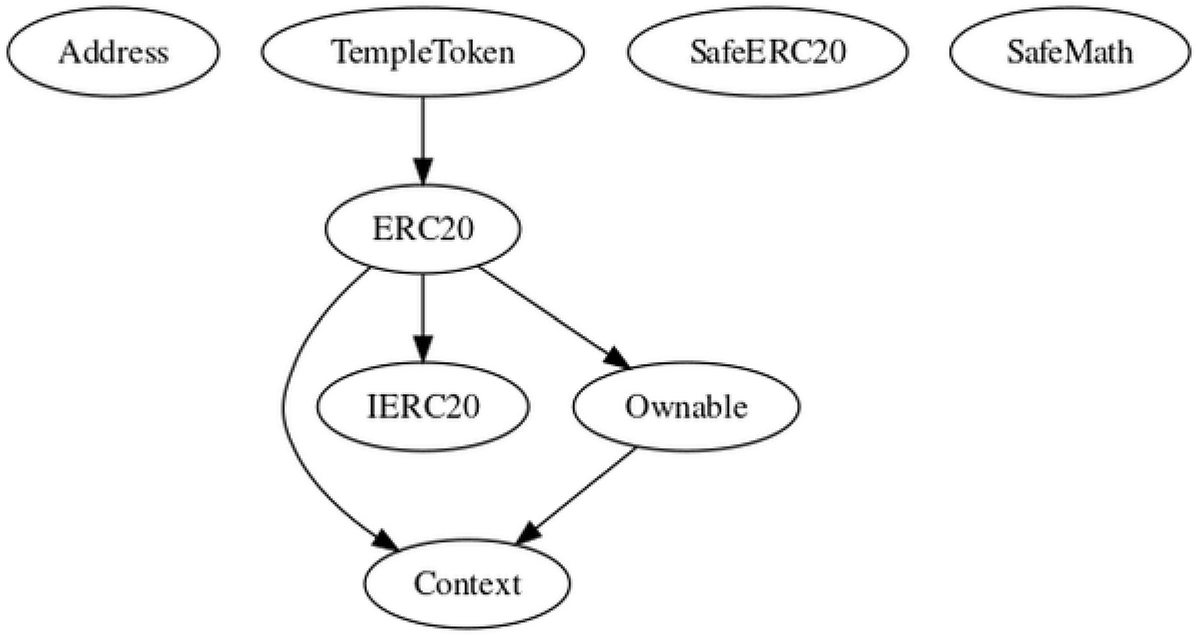
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	_functionCallWithValue	Private	✓	
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
ERC20	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	getOwner	External		-
	name	Public		-
	decimals	Public		-
	symbol	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-

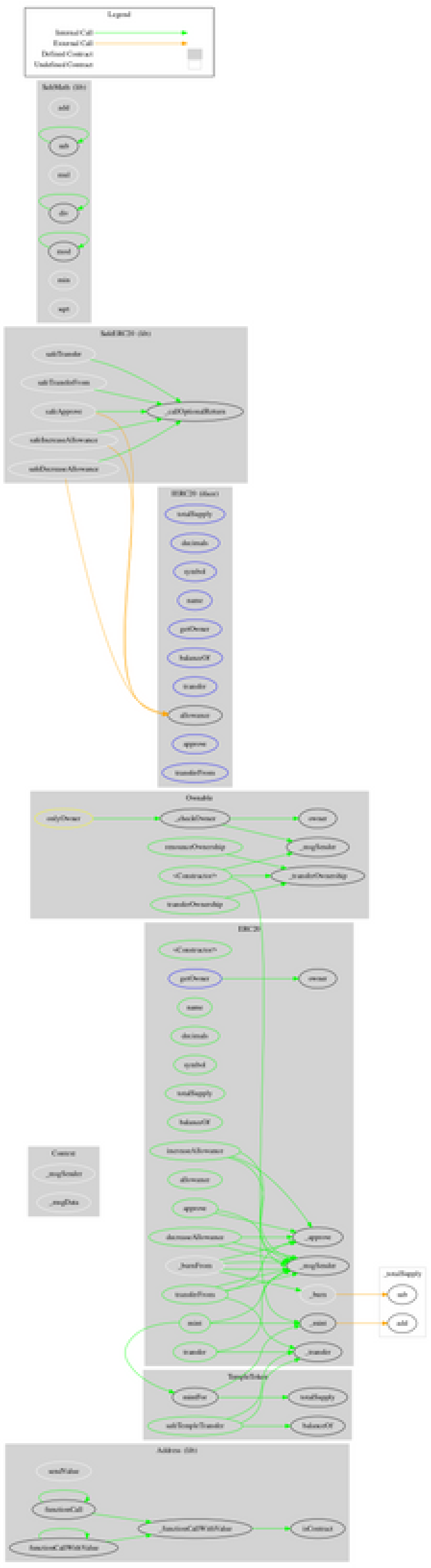
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	mint	Public	✓	onlyOwner
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_burnFrom	Internal	✓	
IERC20	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner

	_transferOwnership	Internal	✓	
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	_callOptionalReturn	Private	✓	
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
	min	Internal		
	sqrt	Internal		
TempleToken	Implementation	ERC20		
		Public	✓	ERC20
	mint	Public	✓	onlyOwner
	mintFor	Public	✓	onlyOwner
	safeTempleTransfer	Public	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
SafeMath (lib)
add
sub
mul
div
mod
min
sqrt
SafeERC20 (lib)
safeTransfer
safeTransferFrom
safeApprove
safeIncreaseAllowance
safeDecreaseAllowance
_callOptionalReturn
IERC20 (iface)
totalSupply
decimals
symbol
name
getOwner
balanceOf
transfer
allowance
approve
transferFrom
Ownable
onlyOwner
_checkOwner
owner
renounceOwnership
_msgSender
<Constructor>
_transferOwnership
transferOwnership
ERC20
increaseAllowance
decreaseAllowance
_burnFrom
mint
_approve
_burn
_mint
_transfer
Context
_msgData
TempleToken
mintFor
safeTempleTransfer
Address (lib)
sendValue
functionCall
functionCallWithValue
isContract

Summary

Temple Fi contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. There are some functions that can be abused by the owner like mint tokens. if the contract owner abuses the mint functionality, the contract will be highly inflated. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

Disclaimer

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