Cyberscope

Audit Report

Cipher

October 2023

Network	MATIC
Address	0xaa404804Ba583c025Fa64c9a276A6127CEB355c6
Audited by	© cyberscope

Table of Contents

Review

Contract Name	PowerfulERC20
Compiler Version	v0.8.4+commit.c7e474f2
Optimization	200 runs
Explorer	https://polygonscan.com/address/0xaa404804ba583c025fa64c9a276a6127ceb355c6
Address	0xaa404804ba583c025fa64c9a276a6127ceb355c6
Network	MATIC
Symbol	CPR
Decimals	2
Total Supply	10,800,000,000

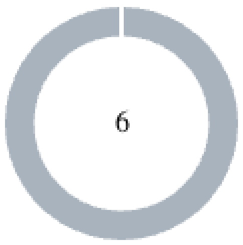
Audit Updates

Initial Audit	11 Oct 2023

Source Files

Filename	SHA256
PowerfulERC20.sol	a402eacca5ac7c72f9b154818e74bce92669f603e27f5ea3a1bef40e9f57c064

Findings Breakdown



●	Critical	0
●	Medium	0
●	Minor / Informative	6

	Severity	Unresolved	Acknowledged	Resolved	Other
●	Critical	0	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	5	0	0	1

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	CCR	Contract Centralization Risk	Renounced
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	PowerfulERC20.sol#L923
Status	Renounced

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

The contract MINTER_ROLE has the authority to mint tokens. The MINTER_ROLE may take advantage of it by calling the `mint` function. During the audit assessment, the max supply is capped and the `_mintingFinished` is false, but the users are able to burn their own tokens and diverse from the max supply. As a result, the MINTER_ROLE will be able to call the mint function up to the equivalent burn amount and the contract tokens will be highly inflated.

```
function _mint(address account, uint256 amount) internal virtual override {
    require(ERC20.totalSupply() + amount <= cap(), "ERC20Capped: cap exceeded");
    super._mint(account, amount);
}
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

Team Update

The contract ownership and authorization roles have been renounced. The information regarding the transactions can be accessed through the following links:

- Ownership Renouncement
- Admin Role Renouncement
- Minter Role Renouncement

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	PowerfulERC20.sol#L524
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.

```
bytes16 private constant alphabet = "0123456789abcdef"
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	PowerfulERC20.sol#L529,554,850,1013,1039,1049,1068,1082,1102,1112,1130,1140,1152
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 toString(uint256 value) internal pure returns (string memory) {
        // Inspired by OraclizeAPI's implementation - MIT licence
        //
https://github.com/oraclize/ethereum-api/blob/b42146b063c7d6ee1358846c198246239e9360
e8/oraclizeAPI_0.4.25.sol

        if (value == 0) {
            return "0";
...
        while (value != 0) {
            digits -= 1;
            buffer[digits] = bytes1(uint8(48 + uint256(value % 10)));
            value /= 10;
        }
        return string(buffer);
    }

...
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	PowerfulERC20.sol#L991,1165
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
            size := extcodesize(account)
        }

assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	PowerfulERC20.sol#L5
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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	PowerfulERC20.sol#L1540
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
IERC20(tokenAddress).transfer(owner(), tokenAmount)
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		

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	✓	
	_beforeTokenTransfer	Internal	✓	
ERC20Decimals	Implementation	ERC20		
		Public	✓	-
	decimals	Public		-

ERC20Mintable	Implementation	ERC20		
	mintingFinished	External		-
	mint	External	✓	canMint
	finishMinting	External	✓	canMint
	_finishMinting	Internal	✓	
Strings	Library			
	toString	Internal		
	toHexString	Internal		
	toHexString	Internal		
IERC165	Interface			
	supportsInterface	External		-
ERC165	Implementation	IERC165		
	supportsInterface	Public		-
IAccessControl	Interface			
	hasRole	External		-
	getRoleAdmin	External		-
	grantRole	External	✓	-
	revokeRole	External	✓	-

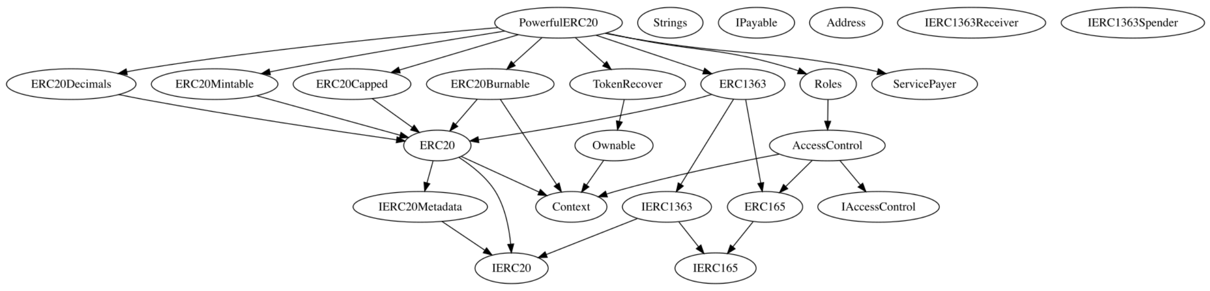
	renounceRole	External	✓	-
AccessControl	Implementation	Context, IAccessControl, ERC165		
	supportsInterface	Public		-
	hasRole	Public		-
	_checkRole	Internal		
	getRoleAdmin	Public		-
	grantRole	Public	✓	onlyRole
	revokeRole	Public	✓	onlyRole
	renounceRole	Public	✓	-
	_setupRole	Internal	✓	
	_setRoleAdmin	Internal	✓	
	_grantRole	Private	✓	
	_revokeRole	Private	✓	
Roles	Implementation	AccessControl		
		Public	✓	-
IPayable	Interface			
	pay	External	Payable	-
ServicePayer	Implementation			
		Public	Payable	-

ERC20Capped	Implementation	ERC20		
		Public	✓	-
	cap	Public		-
	_mint	Internal	✓	
ERC20Burnable	Implementation	Context, ERC20		
	burn	Public	✓	-
	burnFrom	Public	✓	-
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	_verifyCallResult	Private		

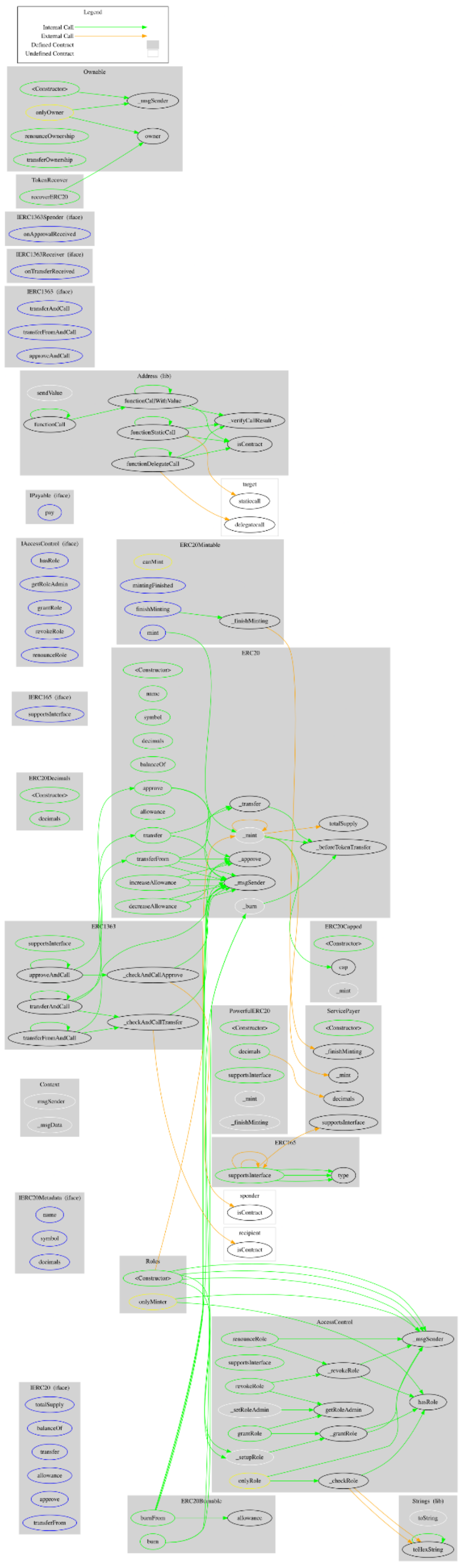
IERC1363	Interface	IERC20, IERC165		
	transferAndCall	External	✓	-
	transferAndCall	External	✓	-
	transferFromAndCall	External	✓	-
	transferFromAndCall	External	✓	-
	approveAndCall	External	✓	-
	approveAndCall	External	✓	-
IERC1363Receiver	Interface			
	onTransferReceived	External	✓	-
IERC1363Spender	Interface			
	onApprovalReceived	External	✓	-
ERC1363	Implementation	ERC20, IERC1363, ERC165		
	supportsInterface	Public		-
	transferAndCall	Public	✓	-
	transferAndCall	Public	✓	-
	transferFromAndCall	Public	✓	-
	transferFromAndCall	Public	✓	-
	approveAndCall	Public	✓	-
	approveAndCall	Public	✓	-

	_checkAndCallTransfer	Internal	✓	
	_checkAndCallApprove	Internal	✓	
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
TokenRecover	Implementation	Ownable		
	recoverERC20	Public	✓	onlyOwner
PowerfulERC20	Implementation	ERC20Decimals, ERC20Capped, ERC20Mintable, ERC20Burnable, ERC1363, TokenRecover, Roles, ServicePayer		
		Public	Payable	ERC20 ERC20Decimals ERC20Capped ServicePayer
	decimals	Public		-
	supportsInterface	Public		-
	_mint	Internal	✓	onlyMinter
	_finishMinting	Internal	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

Cipher contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

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