Cyberscope

Audit Report

Cashpepe

April 2024

Network	BSC
Address	0xb18BB833953F684F51f36f90067c3D863f2d8B06
Audited by	© cyberscope

Table of Contents

Table of Contents 1
Review 2
Audit Updates 2
Source Files 2
Findings Breakdown 3
Diagnostics 4
MT - Mints Tokens 5
Description 5
Recommendation 5
L09 - Dead Code Elimination 6
Description 6
Recommendation 6
L13 - Divide before Multiply Operation 8
Description 8
Recommendation 8
L17 - Usage of Solidity Assembly 9
Description 9
Recommendation 9
L18 - Multiple Pragma Directives 10
Description 10
Recommendation 10
L19 - Stable Compiler Version 11
Description 11
Recommendation 11
L20 - Succeeded Transfer Check 12
Description 12
Recommendation 12
Functions Analysis 13
Inheritance Graph 14
Flow Graph 15
Summary 16
Disclaimer 17
About Cyberscope 18

Review

Contract Name	PowerfulBEP20
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://bscscan.com/address/0xb18bb833953f684f51f36f90067c3d863f2d8b06
Address	0xb18bb833953f684f51f36f90067c3d863f2d8b06
Network	BSC
Symbol	CASHPEPE
Decimals	18
Total Supply	420,690,000,000,000

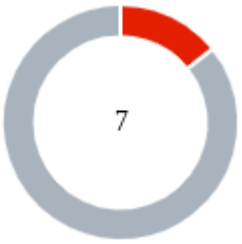
Audit Updates

Initial Audit	16 Apr 2024

Source Files

Filename	SHA256
PowerfulBEP20.sol	ebf7a6bec17290c777b01a573ae27d820963bd9822f60e693fb45db9a7da1ab1

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	6	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Critical	MT	Mints Tokens	Unresolved
Minor / Informative	L09	Dead Code Elimination	Unresolved
Minor / Informative	L13	Divide before Multiply Operation	Unresolved
Minor / Informative	L17	Usage of Solidity Assembly	Unresolved
Minor / Informative	L18	Multiple Pragma Directives	Unresolved
Minor / Informative	L19	Stable Compiler Version	Unresolved
Minor / Informative	L20	Succeeded Transfer Check	Unresolved

MT - Mints Tokens

Criticality	Critical
Location	PowerfulBEP20.sol#L1615
Status	Unresolved

Description

The `MINTER_ROLE` address has the authority to mint tokens. The `MINTER_ROLE` account may take advantage of it by calling the `mint` function. As a result, the contract tokens will be highly inflated.

```
function mint(address account, uint256 amount) external canMint
{
    _mint(account, amount);
}
```

Recommendation

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

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the MINTER_ROLE, which will eliminate the threats but it is non-reversible.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	PowerfulBEP20.sol#L649,674,684,703,713,730,740,755,765,780,804,816,1572,1579,1587,1598,1608,1692,1705,1741,1752,1794,1805,1843,1856,1886,1908,1915,1923,1932,1958,1983,1990,2021,2231,2240
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 sendValue(address payable recipient, uint256 amount)
internal {
        require(address(this).balance >= amount, "Address:
insufficient balance");

        (bool success, ) = recipient.call{value: amount}("");
        require(success, "Address: unable to send value, recipient
may have reverted");
    }

function functionCall(address target, bytes memory data) internal
returns (bytes memory) {
        return functionCallWithValue(target, data, 0, "Address:
low-level call failed");
    }

...
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	PowerfulBEP20.sol#L1654,1657,1669,1673,1674,1675,1676,1677,1678,1684
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
prod0 := div(prod0, twos)
result = prod0 * inverse
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	PowerfulBEP20.sol#L821,1170,1201,1615,1964
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 {
                let returndata_size := mload(returndata)
                revert(add(32, returndata), returndata_size)
            }

assembly {
                    revert(add(32, reason), mload(reason))
                }

assembly {
                let mm := mulmod(x, y, not(0))
                prod0 := mul(x, y)
                prod1 := sub(sub(mm, prod0), lt(mm, prod0))
            }

...
```

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.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	PowerfulBEP20.sol#L8,89,118,145,511,551,589,836,864,893,978,1016,1048,1214,1297,1319,1336,1356,1382,1400,1466,1557,1899,1945,2031,2279,2299,2319
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```
pragma solidity ^0.8.0;
pragma solidity ^0.8.1;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	PowerfulBEP20.sol#L8,89,118,145,511,551,589,836,864,893,978,1016,1048,1214,1297,1319,1336,1356,1382,1400,1466,1557,1899,1945,2031,2279,2299,2319
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;
pragma solidity ^0.8.1;
```

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	PowerfulBEP20.sol#L1311
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)
```

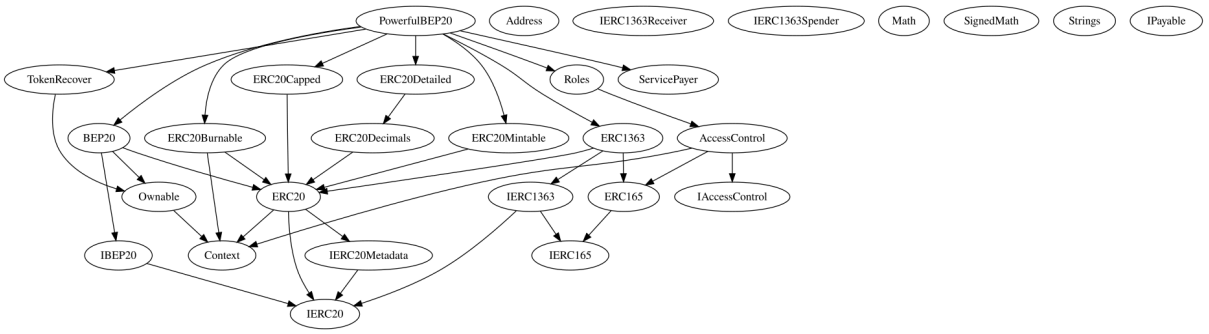
Recommendation

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.

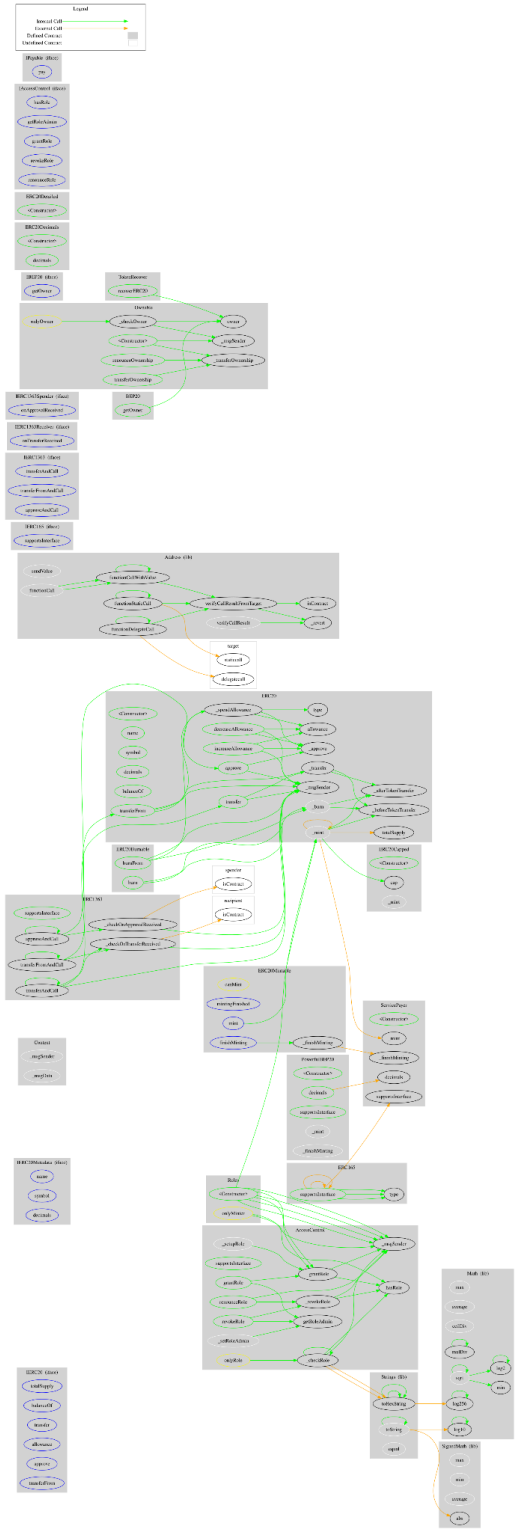
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
PowerfulBEP20	Implementation	BEP20, ERC20Detailed, ERC20Capped, ERC20Mintable, ERC20Burnable, ERC1363, TokenRecover, Roles, ServicePayer		
		Public	Payable	ERC20Detailed ERC20Capped ServicePayer
	decimals	Public		-
	supportsInterface	Public		-
	_mint	Internal	✓	onlyMinter
	_finishMinting	Internal	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

Cashpepe 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, then 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

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