Cyberscope

Audit Report

SportPoint

July 2024

Network BSC

Address 0x0cF453DC7EA21ef8FdfFC1B14Cb848E1e3884Be5

Audited by © cyberscope

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	BEP20Token
Compiler Version	v0.8.0+commit.c7dfd78e
Optimization	200 runs
Explorer	https://bscscan.com/address/0x0cf453dc7ea21ef8fdffc1b14cb848e1e3884be5
Address	0x0cf453dc7ea21ef8fdffc1b14cb848e1e3884be5
Network	BSC
Symbol	POINT
Decimals	18
Total Supply	1,000,000,000

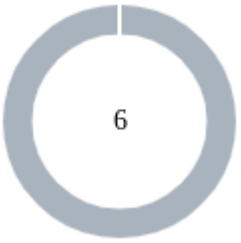
Audit Updates

Initial Audit	31 Jul 2024

Source Files

Filename	SHA256
BEP20Token.sol	026d6abd0bf6fdbe7a45ffe4d29fc38ed10d7c6d3482619aed90628d28e91a1b

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	6	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	CR	Code Repetition	Unresolved
	IVD	Inefficient Variable Definition	Unresolved
	MI	Meta-Transaction Incompatibility	Unresolved
	MEM	Missing Error Messages	Unresolved
	L02	State Variables could be Declared Constant	Unresolved
	L19	Stable Compiler Version	Unresolved

CR - Code Repetition

Criticality	Minor / Informative
Location	BEP20Token.sol#L54,L72
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimise code repetition where possible. In this case, both `transfer` and `transferFrom` functions implement the same transfer mechanism.

```
  function transfer(address receiver, uint256 numTokens) public
override returns (bool) {
        require(numTokens <= balances[msg.sender]);
        balances[msg.sender] -= numTokens;
        balances[receiver] += numTokens;
        emit Transfer(msg.sender, receiver, numTokens);
        return true;
    }
```

```
  function transferFrom(address owner, address buyer, uint256
numTokens) public override returns (bool) {
        require(numTokens <= balances[owner]);
        require(numTokens <= allowed[owner][msg.sender]);

        balances[owner] -= numTokens;
        allowed[owner][msg.sender] -= numTokens;
        balances[buyer] += numTokens;
        emit Transfer(owner, buyer, numTokens);
        return true;
    }
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

IVD - Inefficient Variable Definition

Criticality	Minor / Informative
Location	BEP20Token.sol#L23
Status	Unresolved

Description

The contract declares state variables inefficiently or includes unnecessary variables. This practice can lead to increased gas consumption and decrease the overall efficiency of the smart contract.

For instance, a state variable is declared for the decimal precision of the contract.

```
uint8 private _decimals = 18;
```

This can be optimised by having the `decimals()` function return the desired precision directly, eliminating the need for the state variable.

Recommendation

The team is suggested to review and optimise the declaration of state variables in the contract. Unnecessary variables should be removed and the remaining variables should be declared in a way that minimises gas consumption. This can improve the overall efficiency of the contract and reduce costs for users that interact with it.

MI - Meta-Transaction Incompatibility

Criticality	Minor / Informative
Location	BEP20Token.sol#L55,L56,L58,L63,L64,L74,L77
Status	Unresolved

Description

The contract is currently designed in a manner that does not accommodate meta-transactions. Meta-transactions are a unique type of blockchain transaction that allows users to delegate transaction submission, processing, and fee payment to a third party.

Consider the case where a user signs a message to transfer their token, submits it to `Contract A` and `Contract A` submits it to the network. In this case, `Contract A` will appear as the sender of the original message and not `User A`. This discrepancy can potentially lead to inconsistencies within the contract.

In this contract, the `msg.sender` variable is used to identify the caller of a method. However, in the context of meta-transactions, `msg.sender` will refer to the intermediate party (`Contract A`) rather than the original caller (`User A`).

```
 function transfer(address receiver, uint256 numTokens) public override
returns (bool) {
        ...
        balances[msg.sender] -= numTokens;
        ...
        return true;
    }
```

Recommendation

This is an informative finding. To support meta-transactions, the team should replace the use of `msg.sender` with a function that returns the initial caller. For standard operations, `msg.sender` remains suitable. For more information see the link: https://docs.openzeppelin.com/contracts/3.x/api/gsn#GSNRecipient-_msgSender--

MEM - Missing Error Messages

Criticality	Minor / Informative
Location	BEP20Token.sol#L51,69,70
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.

```
require(numTokens <= balances[msg.sender])
require(numTokens <= balances[owner])
require(numTokens <= allowed[owner][msg.sender])
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	BEP20Token.sol#L17,L18,L19
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
string private _name = "SportPoint"
string private _symbol = "POINT"
uint8 private _decimals = 18
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behaviour, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	BEP20Token.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;
```

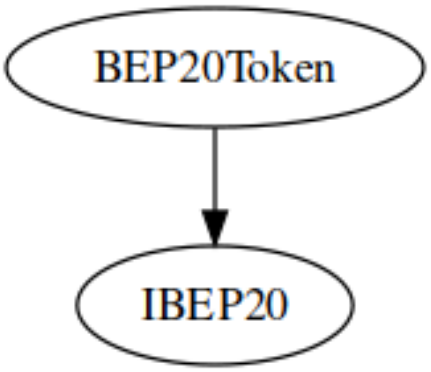
Recommendation

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**	
IBEP20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
BEP20Token	Implementation	IBEP20		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	approve	Public	✓	-
	allowance	Public		-
	transferFrom	Public	✓	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
BEP20Token
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transfer
approve
allowance
transferFrom
IBEP20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom

Summary

SportPoint is an interesting project that has a friendly and growing community. This audit investigated security issues, business logic concerns and potential improvements. The Smart Contract analysis reported no compiler error or critical issues. Only minor recommendations were identified to optimise the smart contract’s performance.

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

cyberscope.io