Cyberscope

Audit Report

Diamond Token

January 2025

Network	SEPOLIA
Address	0x7E204a9Da7A9E8096eE27940F911b5e256C75c20
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
●	UPA	Unexcluded Pinksale Address	Unresolved
●	FOV	Fee Overwriting Value	Unresolved
●	RED	Redundant Event Declaration	Unresolved
●	RISD	Redundant Initial Supply Declaration	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved

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	Diamond_Token
Compiler Version	v0.8.24+commit.e11b9ed9
Optimization	200 runs
Explorer	https://sepolia.etherscan.io/address/0x7e204a9da7a9e8096ee27940f911b5e256c75c20
Address	0x7E204a9Da7A9E8096eE27940F911b5e256C75c20
Network	SEPOLIA
Symbol	DIT
Decimals	18
Total Supply	100,000,000
Badge Eligibility	Must Fix UPA

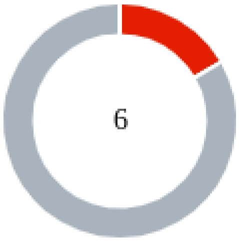
Audit Updates

Initial Audit	3 Jan 2025

Source Files

Filename	SHA256
Diamon_Token_BEP20.sol	38364221cc87bcc30bec0df08c4533747c70c98252c3f1734adebf511cf47eb7

Findings Breakdown



Critical	1
Medium	0
Minor / Informative	5

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

UPA - Unexcluded Pinksale Address

Criticality	Critical
Location	Diamon_Token_BEP20.sol#L66
Status	Unresolved

Description

The contract incorporates operational restrictions on transactions, which can hinder seamless interaction with decentralized applications (dApps) such as launchpads, presales, lockers, or staking platforms. In scenarios where an external contract, such as a launchpad factory, needs to integrate with the contract, it should be exempt from the limitations to ensure uninterrupted service and functionality. Failure to provide such exemptions can block the successful process and operation of services reliant on this contract.

```
    function _update(
        address sender,
        address recipient,
        uint256 amount
    ) internal override {
        if (isExcludedFromFee[sender] || isExcludedFromFee[recipient]) {
            super._update(sender, recipient, amount);
            return;
        }
```

Recommendation

It is advisable to modify the contract by incorporating functionality that enables the exclusion of designated addresses from transactional restrictions. This enhancement will allow specific addresses, such as those associated with decentralized applications (dApps) and service platforms, to operate without being hindered by the standard constraints imposed on other users. Implementing this feature will ensure smoother integration and functionality with external systems, thereby expanding the contract's versatility and effectiveness in diverse operational environments.

FOV - Fee Overwriting Value

Criticality	Minor / Informative
Location	Diamon_Token_BEP20.sol#L66
Status	Unresolved

Description

The contract is designed to apply a transaction fee by calculating a `feeAmount` using the `transferFeePercentage` variable. However, when a transaction involves a swap, the contract calculates a new fee based on the `liquidityFeePercentage` and applies this fee instead, overwriting the previously calculated `feeAmount`. This logic does not take into account the original transfer fee and results in only the new liquidity fee being applied. This behaviour creates inconsistencies in how fees are applied between regular transactions and swap transactions, potentially causing confusion or a mismatch with the expected fee structure.

```
    function _update(
        address sender,
        address recipient,
        uint256 amount
    ) internal override {
        ...
        uint256 feeAmount = (amount * transferFeePercentage) /
10000;
        uint256 amountAfterFee = amount - feeAmount;
        if (sender == v2PairAddress || recipient ==
v2PairAddress) {
            // the user is trying to swap the tokens
            uint256 LiquidityFeeAmount = (amount *
liquidityFeePercentage) /
                10000;
            uint256 amountAfterFee_2 = amount -
LiquidityFeeAmount;
            super._update(sender, feeCollector,
LiquidityFeeAmount);
            super._update(sender, recipient, amountAfterFee_2);
            return;
        }
        super._update(sender, feeCollector, feeAmount); //
Transfer fee to the fee collector
        super._update(sender, recipient, amountAfterFee); //
Transfer the remaining tokens to the recipient
    }
```

Recommendation

It is recommended to implement a generalised fee calculation mechanism that can dynamically account for all relevant fees, ensuring consistency across all transaction types. The team should consider the intended functionality of the contract, whether a uniform fee structure is desired or if dynamic fees based on the transaction type or destination are appropriate. If dynamic fees are intended, the team should ensure the logic is explicitly clear, thoroughly documented, to prevent unintended overwriting of fees and to maintain transparency for users.

RED - Redundant Event Declaration

Criticality	Minor / Informative
Location	Diamon_Token_BEP20.sol#L15
Status	Unresolved

Description

The contract uses events that are not emitted within the contract's functions. As a result, these declared events are redundant and serve no purpose within the contract's current implementation.

```
    event TransferFeeTaken(
        address indexed from,
        address indexed to,
        uint256 amount
    );
```

Recommendation

To optimize contract performance and efficiency, it is advisable to regularly review and refactor the codebase, removing the unused event declarations. This proactive approach not only streamlines the contract, reducing deployment and execution costs but also enhances readability and maintainability.

RISD - Redundant Initial Supply Declaration

Criticality	Minor / Informative
Location	Diamon_Token_BEP20.sol#L23,33
Status	Unresolved

Description

The contract declares an `initialSupply_` constant with a value of `100,000,000 * 10^18` tokens. However, within the constructor, it also accepts `initialSupply` as a parameter, which is subsequently used to mint tokens. This dual declaration introduces redundancy and may lead to confusion about the intended initial supply of tokens. Furthermore, it increases the risk of inconsistencies if the declared constant `initialSupply_` and the parameter `initialSupply` do not align, potentially impacting the predictability of the contract's behaviour.

```
    uint256 public constant initialSupply_ = 100_000_000 * 10**18; // 
100,000,000 tokens with 18 decimals

    constructor(uint256 initialSupply)
        ERC20("Diamond_Token", "DIT")
        Ownable(ownerWallet)
    {
        ...
        _mint(msg.sender, initialSupply * 10**18);
    }
```

Recommendation

It is recommended to streamline the contract design by removing the redundant `initialSupply_` declaration if the initial supply is meant to be dynamic and determined via the constructor parameter. Alternatively, if the initial supply is intended to be a fixed value, the constructor parameter should be eliminated, and the constant `initialSupply_` should be used directly in the `_mint` function. This approach would reduce redundancy, improve code clarity, and ensure consistency in the contract's implementation.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	Diamon_Token_BEP20.sol#L12
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.

```
address public ownerWallet =
0x1d64FD1e4eB9Df7C75Ad4B4DAe6A23aa8C4B5fe8
```

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 behavior, 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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Diamon_Token_BEP20.sol#L8,60
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.

```
contract Diamond_Token is ERC20, Ownable {
    uint256 public transferFeePercentage = 300; // Fee
percentage in basis points (e.g., 100 = 1%)
    uint256 public liquidityFeePercentage = 150; // Fee
percentage in basis points (e.g., 100 = 1%)
    address public feeCollector =
0xd841972Ac48461517f561CB6785E2f1CBe37Ea07; // Address to
receive the fee
    address public ownerWallet =
0x1d64FD1e4eB9Df7C75Ad4B4DAe6A23aa8C4B5fe8; // Owner's wallet
    address public v2PairAddress;
...
            super._update(sender, recipient, amountAfterFee_2);
            return;
        }
        super._update(sender, feeCollector, feeAmount); //
Transfer fee to the fee collector
        super._update(sender, recipient, amountAfterFee); //
Transfer the remaining tokens to the recipient
    }
}

...
```

Recommendation

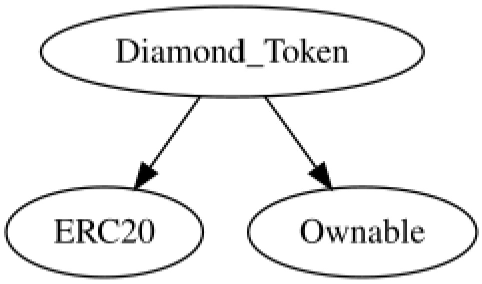
By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation https://docs.soliditylang.org/en/stable/style-guide.html#naming-conventions.

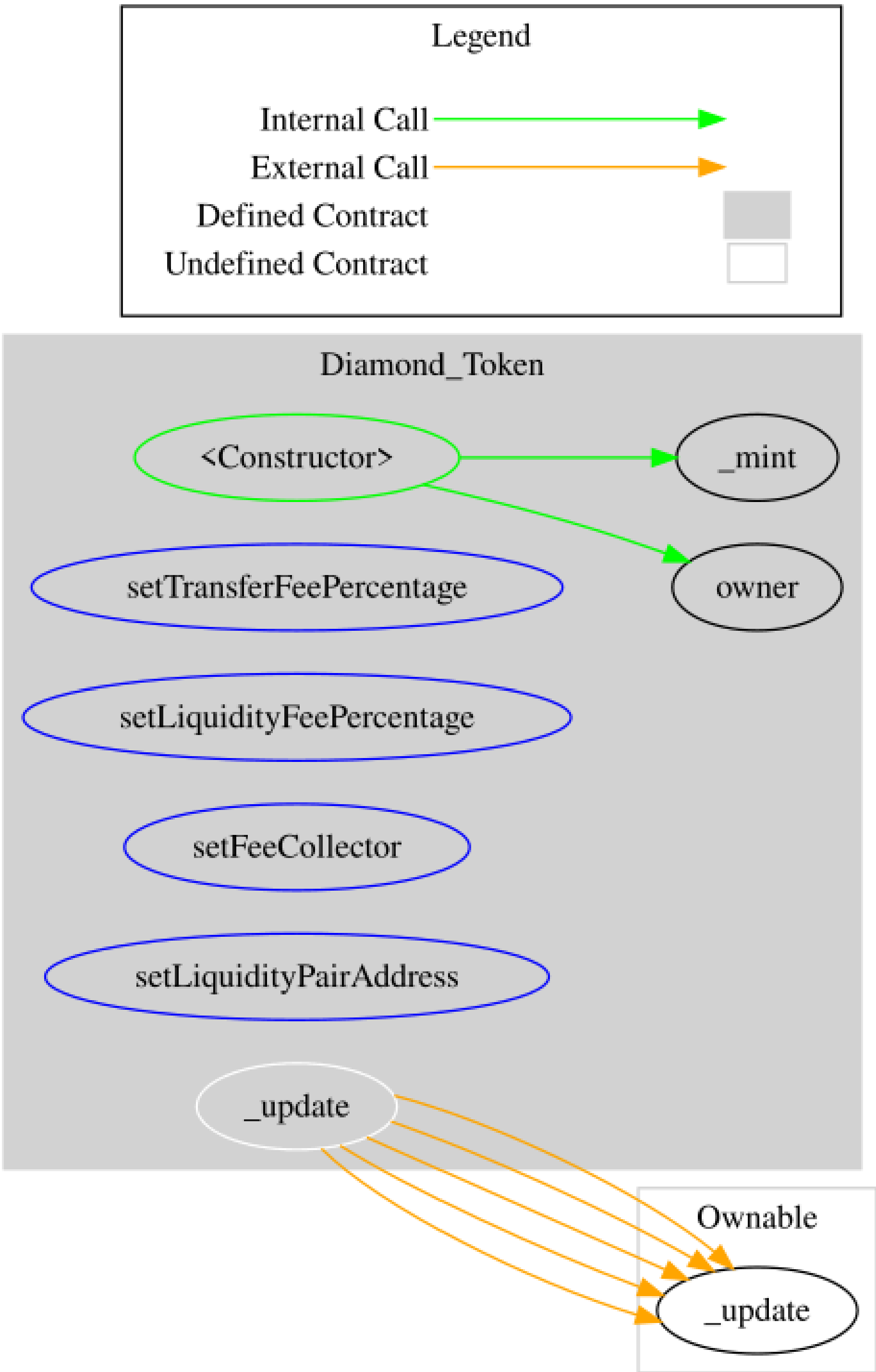
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Diamond_Token	Implementation	ERC20, Ownable		
		Public	✓	ERC20 Ownable
	setTransferFeePercentage	External	✓	onlyOwner
	setLiquidityFeePercentage	External	✓	onlyOwner
	setFeeCollector	External	✓	onlyOwner
	setLiquidityPairAddress	External	✓	onlyOwner
	_update	Internal	✓	

Inheritance Graph



Flow Graph



Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Diamond_Token
<Constructor>
_mint
owner
setTransferFeePercentage
setLiquidityFeePercentage
setFeeCollector
setLiquidityPairAddress
_update
Ownable
_update

Summary

Diamond Token contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Diamond Token is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error but 1 critical issue. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions. There is also a limit of max 5% fees.

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