



Advanced Manual **Smart Contract Audit**

October 31, 2022

 [CoinsultAudits](#)

 info@coinsult.net

 coinsult.net

Audit requested by



Snowflake

0xE0f463832295ADf63eB6CA053413a3f9cd8bf685

Table of Contents

1. Audit Summary

- 1.1 Audit scope
- 1.2 Tokenomics
- 1.3 Source Code

2. Disclaimer

3. Global Overview

- 3.1 Informational issues
- 3.2 Low-risk issues
- 3.3 Medium-risk issues
- 3.4 High-risk issues

4. Vulnerabilities Findings

5. Contract Privileges

- 5.1 Maximum Fee Limit Check
- 5.2 Contract Pausability Check
- 5.3 Max Transaction Amount Check
- 5.4 Exclude From Fees Check
- 5.5 Ability to Mint Check
- 5.6 Ability to Blacklist Check
- 5.7 Owner Privileges Check

6. Notes

- 6.1 Notes by Coinsult
- 6.2 Notes by Snowflake

7. Contract Snapshot

8. Website Review

9. Certificate of Proof

Audit Summary

Project Name	Snowflake
Website	https://www.snowflake.exchange/
Blockchain	Polygon Chain
Smart Contract Language	Solidity
Contract Address	0xE0f463832295ADf63eB6CA053413a3f9cd8bf685
Audit Method	Static Analysis, Manual Review
Date of Audit	31 October 2022

This audit report has been prepared by Coinsult's experts at the request of the client. In this audit, the results of the static analysis and the manual code review will be presented. The purpose of the audit is to see if the functions work as intended, and to identify potential security issues within the smart contract.

The information in this report should be used to understand the risks associated with the smart contract. This report can be used as a guide for the development team on how the contract could possibly be improved by remediating the issues that were identified.

Audit Scope





Source Code

Coinsult was commissioned by Snowflake to perform an audit based on the following code:

<https://polygonscan.com/address/0xE0f463832295ADf63eB6CA053413a3f9cd8bf685#code>

Note that we only audited the code available to us on this URL at the time of the audit. If the URL is not from any block explorer (main net), it may be subject to change. Always check the contract address on this audit report and compare it to the token you are doing research for.

Tokenomics

Rank	Address	Quantity (Token)	Percentage
1	 0x334b922baac8e0406583fb8a548913786addde04	60,000,000	30.0000%
2	0xb4596ed10d22822302dd8f9a184ffa7a1314d5	50,000,000	25.0000%
3	 0xabc22e1a7944e626859fdb2ed8efcecd2f908681	40,000,000	20.0000%
4	 0x4fa66f42320a16d112b8876d650368b964567e9a	20,000,000	10.0000%
5	 0xbbbb2ab6a7fa7a719e2647953728c5484e30b31d	20,000,000	10.0000%

Audit Method

Coinsult's manual smart contract audit is an extensive methodical examination and analysis of the smart contract's code that is used to interact with the blockchain. This process is conducted to discover errors, issues and security vulnerabilities in the code in order to suggest improvements and ways to fix them.

➔ Automated Vulnerability Check

Coinsult uses software that checks for common vulnerability issues within smart contracts. We use automated tools that scan the contract for security vulnerabilities such as integer-overflow, integer-underflow, out-of-gas-situations, unchecked transfers, etc.

➔ Manual Code Review

Coinsult's manual code review involves a human looking at source code, line by line, to find vulnerabilities. Manual code review helps to clarify the context of coding decisions. Automated tools are faster but they cannot take the developer's intentions and general business logic into consideration.

➔ Used Tools

- Slither: Solidity static analysis framework
- Remix: IDE Developer Tool
- CWE: Common Weakness Enumeration
- SWC: Smart Contract Weakness Classification and Test Cases
- DEX: Testnet Blockchains

Risk Classification

Coinsult uses certain vulnerability levels, these indicate how bad a certain issue is. The higher the risk, the more strictly it is recommended to correct the error before using the contract.

Vulnerability Level	Description
● Informational	Does not compromise the functionality of the contract in any way
● Low-Risk	Won't cause any problems, but can be adjusted for improvement
● Medium-Risk	Will likely cause problems and it is recommended to adjust
● High-Risk	Will definitely cause problems, this needs to be adjusted

Coinsult has four statuses that are used for each risk level. Below we explain them briefly.

Risk Status	Description
Total	Total amount of issues within this category
Pending	Risks that have yet to be addressed by the team
Acknowledged	The team is aware of the risks but does not resolve them
Resolved	The team has resolved and remedied the risk

Disclaimer

This audit report has been prepared by Coinsult's experts at the request of the client. In this audit, the results of the static analysis and the manual code review will be presented. The purpose of the audit is to see if the functions work as intended, and to identify potential security issues within the smart contract.

The information in this report should be used to understand the risks associated with the smart contract. This report can be used as a guide for the development team on how the contract could possibly be improved by remediating the issues that were identified.

Coinsult is not responsible if a project turns out to be a scam, rug-pull or honeypot. We only provide a detailed analysis for your own research.

Coinsult is not responsible for any financial losses. Nothing in this contract audit is financial advice, please do your own research.

The information provided in this audit is for informational purposes only and should not be considered investment advice. Coinsult does not endorse, recommend, support or suggest to invest in any project.

Coinsult can not be held responsible for when a project turns out to be a rug-pull, honeypot or scam.

Global Overview

Manual Code Review

In this audit report we will highlight the following issues:

Vulnerability Level	Total	Pending	Acknowledged	Resolved
● Informational	0	0	0	0
● Low-Risk	2	0	2	0
● Medium-Risk	0	0	0	0
● High-Risk	0	0	0	0

Centralization Risks

Coinsult checked the following privileges:

Contract Privilege	Description
Owner can mint?	● Owner can mint new tokens
Owner can blacklist?	● Owner cannot blacklist addresses
Owner can set fees > 25%?	● Owner cannot set the sell fee to 25% or higher
Owner can exclude from fees?	● Owner cannot exclude from fees
Owner can pause trading?	● Owner cannot pause the contract
Owner can set Max TX amount?	● Owner cannot set max transaction amount

More owner privileges are listed later in the report.

Error Code	Description
SWC-104	CWE-252: Unchecked Return Value

● **Low-Risk:** Could be fixed, will not bring problems.

Unchecked transfer

The return value of an external transfer/transferFrom call is not checked.

```
function _transfer(address sender, address recipient, uint256 amount) internal override {  
    ERC20._transfer(sender, recipient, amount);  
    _moveDelegates(sender, recipient, amount);  
}
```

Recommendation

Use SafeERC20, or ensure that the transfer/transferFrom return value is checked.

Exploit scenario

```
contract Token {  
    function transferFrom(address _from, address _to, uint256 _value) public returns (bool success);  
}  
contract MyBank{  
    mapping(address => uint) balances;  
    Token token;  
    function deposit(uint amount) public{  
        token.transferFrom(msg.sender, address(this), amount);  
        balances[msg.sender] += amount;  
    }  
}
```

Several tokens do not revert in case of failure and return false. If one of these tokens is used in MyBank, deposit will not revert if the transfer fails, and an attacker can call deposit for free..

Error Code	Description
CS: 071	Using safemath in Solidity 0.8.0+

● **Low-Risk:** Could be fixed, will not bring problems.

Using safemath in Solidity 0.8.0+

SafeMath is generally not needed starting with Solidity 0.8, since the compiler now has built in overflow checking.

```
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            uint256 c = a + b;
            if (c < a) return (false, 0);
            return (true, c);
        }
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, with an overflow flag.
```

Recommendation

Check if you really need SafeMath and consider removing it.

Maximum Fee Limit Check

Error Code	Description
CEN-01	Centralization: Operator Fee Manipulation

Coinsult tests if the owner of the smart contract can set the transfer, buy or sell fee to 25% or more. It is bad practice to set the fees to 25% or more, because owners can prevent healthy trading or even stop trading when the fees are set too high.


Type of fee	Description
Transfer fee	● Owner cannot set the transfer fee to 25% or higher
Buy fee	● Owner cannot set the buy fee to 25% or higher
Sell fee	● Owner cannot set the sell fee to 25% or higher

Type of fee	Description
Max transfer fee	0%
Max buy fee	0%
Max sell fee	0%

Contract Pausability Check

Error Code	Description
CEN-02	Centralization: Operator Pausability


Coinsult tests if the owner of the smart contract has the ability to pause the contract. If this is the case, users can no longer interact with the smart contract; users can no longer trade the token.

Privilege Check	Description
Can owner pause the contract?	 Owner cannot pause the contract

Max Transaction Amount Check

Error Code	Description
CEN-03	Centralization: Operator Transaction Manipulation

Coinsult tests if the owner of the smart contract can set the maximum amount of a transaction. If the transaction exceeds this limit, the transaction will revert. Owners could prevent normal transactions to take place if they abuse this function.

Privilege Check	Description
Can owner set max tx amount?	 Owner cannot set max transaction amount

Exclude From Fees Check

Error Code	Description
CEN-04	Centralization: Operator Exclusion

Coinsult tests if the owner of the smart contract can exclude addresses from paying tax fees. If the owner of the smart contract can exclude from fees, they could set high tax fees and exclude themselves from fees and benefit from 0% trading fees. However, some smart contracts require this function to exclude routers, dex, cex or other contracts / wallets from fees.

Privilege Check	Description
Can owner exclude from fees?	● Owner cannot exclude from fees

Ability To Mint Check

Error Code	Description
CEN-05	Centralization: Operator Increase Supply

Coinsult tests if the owner of the smart contract can mint new tokens. If the contract contains a mint function, we refer to the token's total supply as non-fixed, allowing the token owner to "mint" more tokens whenever they want.

A mint function in the smart contract allows minting tokens at a later stage. A method to disable minting can also be added to stop the minting process irreversibly.

Minting tokens is done by sending a transaction that creates new tokens inside of the token smart contract. With the help of the smart contract function, an unlimited number of tokens can be created without spending additional energy or money.

Privilege Check	Description
Can owner mint?	● Owner can mint new tokens

Function

```
function mintToFarm(address _farm) external onlyOwner {
    require(!doOnce, "only can mint once");
    doOnce = true;
    __mint(_farm, MAX_SUPPLY * 5 / 10);
}
```

Ability To Blacklist Check

Error Code	Description
CEN-06	Centralization: Operator Dissallows Wallets

Coinsult tests if the owner of the smart contract can blacklist accounts from interacting with the smart contract. Blacklisting methods allow the contract owner to enter wallet addresses which are not allowed to interact with the smart contract.

This method can be abused by token owners to prevent certain / all holders from trading the token. However, blacklists might be good for tokens that want to rule out certain addresses from interacting with a smart contract.

Privilege Check	Description
Can owner blacklist?	● Owner cannot blacklist addresses

Other Owner Privileges Check

Error Code	Description
CEN-100	Centralization: Operator Privileges

Coinsult lists all important contract methods which the owner can interact with.

- ⚠ Owner can mint to farm wallet, but only once
- ⚠ Contract contains delegation calls for voting / checkpoints

Notes

Notes by Snowflake

No notes provided by the team.

Notes by Coinsult

Contract contains

TokenVesting public communityLockToken;

TokenVesting public treasuryLockToken;

TokenVesting public teamLockLockToken;

TokenVesting public angelLockToken;

TokenVesting public advisorLockToken;

Which are unverified and not audited by Coinsult

Contract Snapshot

This is how the constructor of the contract looked at the time of auditing the smart contract.

```
contract Token is Ownable, ERC20{

    TokenVesting public communityLockToken;
    TokenVesting public treasuryLockToken;
    TokenVesting public teamLockToken;
    TokenVesting public angelLockToken;
    TokenVesting public advisorLockToken;

    bool doOnce;
    uint256 public constant MAX_SUPPLY = 400e6 * 1e18;

    constructor(string memory name_, string memory symbol_) Ownable() ERC20(name_,symbol_){

        treasuryLockToken = new TokenVesting(msg.sender, block.timestamp + 180 days, 0 days, 300 days);
        teamLockToken = new TokenVesting(msg.sender, block.timestamp + 360 days, 0 days, 300 days);
        communityLockToken = new TokenVesting(msg.sender, block.timestamp + 90 days, 0 days, 120 days);
        angelLockToken = new TokenVesting(msg.sender, block.timestamp + 360 days, 0 days, 150 days);
        advisorLockToken = new TokenVesting(msg.sender, block.timestamp + 180 days, 0 days, 0 days);

        __mint(msg.sender, MAX_SUPPLY * 25 / 200);
        __mint(address(treasuryLockToken), MAX_SUPPLY * 15 / 100);
        __mint(address(teamLockToken), MAX_SUPPLY * 5 / 100);
        __mint(address(communityLockToken), MAX_SUPPLY * 10 / 100);
        __mint(address(angelLockToken), MAX_SUPPLY * 5 / 100);
        __mint(address(advisorLockToken), MAX_SUPPLY * 5 / 200);

    }

    function mintToFarm(address _farm) external onlyOwner {
        require(!doOnce, "only can mint once");
        doOnce = true;
        __mint(_farm, MAX_SUPPLY * 5 / 10);
    }

    function __mint(address _to, uint256 _amount) internal {
        __mint(_to, _amount);
        __moveDelegates(address(0), _to, _amount);
    }
}
```

Certificate of Proof

● Not KYC verified by Coinsult

Snowflake

Audited by Coinsult.net




Date: 31 October 2022

✓ Advanced Manual Smart Contract Audit

End of report

Smart Contract Audit

 CoinsultAudits

 info@coinsult.net

 coinsult.net

Request your smart contract audit / KYC

t.me/coinsult_tg