Cyberscope

Audit Report
Stelnar

April 2024

Network	BSC Testnet
Address	0x042440b38f3ee8fdaebdff162558cce265cb0dde
Audited by	© cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	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
●	ITD	Incorrect Tax Deduction	Unresolved
●	PAMAR	Pair Address Max Amount Restriction	Unresolved
●	EPC	Existing Pair Creation	Unresolved
●	MVN	Misleading Variable Naming	Unresolved
●	MTEE	Missing Transfer Event Emission	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	Stelnar
Compiler Version	v0.8.23+commit.f704f362
Optimization	200 runs
Explorer	https://testnet.bscscan.com/address/0x042440b38f3ee8fdaebdff162558cce265cb0dde
Address	0x042440b38f3ee8fdaebdff162558cce265cb0dde
Network	BSC_TESTNET
Symbol	STL
Decimals	18
Total Supply	10,000,000,000
Badge Eligibility	Must Fix Criticals

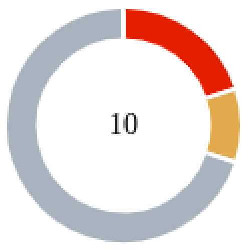
Audit Updates

Initial Audit	16 Apr 2024

Source Files

Filename	SHA256
Stelnar.sol	071311708dd23e8d92abf7a7ab27a93054c7cf7b1eec6d7a77d7ccb4dd5c3575

Findings Breakdown



●	Critical	2
●	Medium	1
●	Minor / Informative	7

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	2	0	0	0
●	Medium	1	0	0	0
●	Minor / Informative	7	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	Stelnar.sol#L218
Status	Unresolved

Description

The contract owner has the authority to limit the sales for all users excluding the owner. The owner may take advantage of it by setting the `_automatedMarketMakerPairs` of the pair address to false. As a result, the contract may operate as a honeypot since will allow only one sale per 5 blocks.

```
if (_transferDelay) {
    if (to != _uniswapV2Router &&
!_automatedMarketMakerPairs[to]) {
        require(
            _holderLastTransferTimestamp[tx.origin] <
                block.number - 4 &&
                _holderLastTransferTimestamp[to] < block.number
- 4,
            "_transfer:: Transfer Delay enabled.  Try again
later."
        );
        _holderLastTransferTimestamp[tx.origin] = block.number;
        _holderLastTransferTimestamp[to] = block.number;
        emit TransferDelay(to, block.number, block.number + 5);
    }
}
```

Additionally, the contract owner has the authority to stop transactions, as described in detail in section `PAMAR`. As a result, the contract might operate as a honeypot.

Recommendation

The contract could embody a check for not allowing setting the `_automatedMarketMakerPairs` of the pair address to false. The team should carefully manage the private keys of the owner'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 ownership, which will eliminate the threats but it is non-reversible.

ITD - Incorrect Tax Deduction

Criticality	Critical
Location	Stelnar.sol#L260,279
Status	Unresolved

Description

The contract is currently deducting the tax value from the `value` variable before it reduces the sender's balance (`fromBalance`). Consequently, the value variable becomes the new lower amount post-tax deduction, which leads to a scenario where the balance of the from address is decreased by the amount minus the tax, rather than the original total amount that was supposed to be charged. This misplacement of the tax deduction step results in an incorrect transaction amount being reflected in the from balance, thereby not accurately representing the actual transaction cost to the sender as initially intended by the contract's design.

```
function _transfer(address from, address to, uint256 value) internal {
        ...
                        _taxAmount = (value * _sTax) / 100;
                        value -= _taxAmount;
                    }
                }
            }

        _transferUpdate(from, to, value, _taxAmount);
    }

    function _transferUpdate(
        address from,
        address to,
        uint256 value,
        uint256 feeAmount
    ) internal {
        uint256 fromBalance = _balances[from];
        if (fromBalance < value) {
            revert ERC20InsufficientBalance(from, fromBalance, value);
        }
        _balances[from] = fromBalance - value;
        _balances[to] += value;

        if (feeAmount > 0) {
            _balances[_taxAddress] += feeAmount;
        }

        emit Transfer(from, to, value);
    }
```

Recommendation

It is recommended to modify the contract so that the deduction from the `fromBalance` occurs before the tax is deducted from the value. This change will ensure that the full transaction amount, including the tax, is correctly deducted from the sender's balance, reflecting a more accurate financial transaction. Adjusting the order of operations to first subtract the total amount from the sender's balance and then apply the tax deduction to the transaction value will prevent discrepancies in balance calculations and maintain the integrity of the financial records in the blockchain ledger. This correction will enhance the contract's reliability and fairness, aligning it with standard accounting practices.

PAMAR - Pair Address Max Amount Restriction

Criticality	Medium
Location	Stelnar.sol#L248
Status	Unresolved

Description

The contract includes an if statement designed to prevent any single wallet from accumulating more tokens than the `_maxBuyAmount` limit by reverting transactions that would exceed this threshold. However, during sales transactions where the to address defaults to the pair address, the transactions are inadvertently halted if the pair address is removed from the `_isExcludedMaxTransactionAmount` mapping. This behavior could severely disrupt the contract's functionality by preventing the sale transactions and inadvertently turn the contract into a honeypot.

```
else if (_isSTaxEnabled && _automatedMarketMakerPairs[to]) {
    if (
        !_isExcludedMaxTransactionAmount[from] ||
        !_isExcludedMaxTransactionAmount[to]
    ) {
        require(
            value <= _maxBuyAmount,
            "Transfer amount exceeds the maxTxAmount."
        );
    }
```

Recommendation

It is recommended to introduce a check that will allow the pair address to hold tokens beyond the `_maxBuyAmount` limit or prevent the removal of the pair address from the `_isExcludedMaxTransactionAmount` mapping. This exception should be specifically coded to recognize and permit pair addresses (or any predefined liquidity pool addresses) to exceed this limit to ensure continuous operation of sales transactions without interruption. Implementing this change will prevent the unintended halting of token sales due to the `_maxBuyAmount` limit and maintain the contract's liquidity and operational integrity. This is crucial for avoiding potential security risks and ensuring that the ecosystem functions as intended without unintended token lock-ups.

EPC - Existing Pair Creation

Criticality	Minor / Informative
Location	Stelnar.sol#L474
Status	Unresolved

Description

The contract contains a function that does not handle the scenario where a pair already exists prior to its execution. If a pair for the given tokens has already been established, the `createPair` function will revert and not proceed with the creation of a new pair. As a result, if a pair has been previously set up before the function is invoked, the contract will encounter an error when trying to call the `createPair` function. This will prevent the successful execution, essentially leading the function to revert.

```
    function createPair(
         address _address
    ) external onlyOwner returns (address pair) {
         pair = IUniswapV2Factory(address(factory)).createPair(
             address(this),
             _address
         );
         _automatedMarketMakerPairs[pair] = true;
    }
```

Recommendation

To mitigate the risks associated with attempting to create an already existing pair, it is recommended to implement a check to determine whether the pair already exists before proceeding to create a new pair. This can be achieved by utilizing the getPair function of the Factory contract to retrieve the address of the pair contract for the specified tokens. If the address returned by the getPair function is the zero address, it indicates that the pair does not exist, and the contract can proceed with the createPair function. Conversely, if a non-zero address is returned, it indicates that the pair already exists, and the createPair function will revert.

MVN - Misleading Variable Naming

Criticality	Minor / Informative
Location	Stelnar.sol#L233,250
Status	Unresolved

Description

The contract is employing the `_maxBuyAmount` variable to limit transaction amounts in both buy and sell scenarios, contrary to what the variable name suggests. This variable is checked against the transaction value in conditions that identify either a buying or a selling event. The use of `_maxBuyAmount` for both types of transactions can lead to confusion and misinterpretation of the code, as users could expect that such a specifically named variable would be exclusive to buy transactions only.

```
// when Buy
if (_isBTaxEnabled && _automatedMarketMakerPairs[from]) {
        require(
            value <= _maxBuyAmount,
            "Transfer amount exceeds the maxTxAmount."
        );
    }
...
// when sell
else if (_isSTaxEnabled && _automatedMarketMakerPairs[to]) {
        require(
            value <= _maxBuyAmount,
            "Transfer amount exceeds the maxTxAmount."
        );
    }
```

Recommendation

It is recommended to rename the variable to reflect the actual usage across different transaction types. A more neutral and descriptive name like `_maxTransactionAmount` would eliminate any ambiguity concerning the variable's purpose and application. This change will enhance the clarity and readability of the contract, thereby reducing the potential for errors and misunderstandings in the management and auditing of the contract.

Ensuring precise and intuitive naming conventions is crucial for maintaining best practices in smart contract development.

MTEE - Missing Transfer Event Emission

Criticality	Minor / Informative
Location	Stelnar.sol#L282
Status	Unresolved

Description

The contract is a missing transfer event emission when fees are transferred to the contract address as part of the transfer process. This omission can lead to a lack of visibility into fee transactions and hinder the ability of decentralized applications (DApps) like blockchain explorers to accurately track and analyze these transactions.

```
if (feeAmount > 0) {
    _balances[_taxAddress] += feeAmount;
}
```

Recommendation

To address this issue, it is recommended to emit a transfer event after transferring the taxed amount to the contract address. The event should include relevant information such as the sender, recipient (contract address), and the amount transferred.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	Stelnar.sol#L464,469
Status	Unresolved

Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
   function excludeFromFees(address _address) external
onlyOwner {
        _isExcludedFromFee[_address] = true;
        emit ExcludedFromFess(_address);
    }

    function includeInFees(address _address) external onlyOwner
{
        _isExcludedFromFee[_address] = false;
        emit IncludeInFees(_address);
    }
```

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Stelnar.sol#L115,116,117,118,120,121,123,124,125,126,127,129,130,131,171,397,404,426,440,446,452,457,463,499,504,522,523
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.

```
mapping(address => bool) _isExcludedMaxTransactionAmount;
mapping(address => uint256) _holderLastTransferTimestamp;
mapping(address => bool) _isExcludedFromFee;
mapping(address => bool) _automatedMarketMakerPairs;
address _uniswapV2Router;
address _taxAddress;
uint256 _bTax;
uint256 _sTax;
uint256 _totalSupply;
uint256 _maxBuyAmount;
uint256 _maxSellAmount;
bool _transferDelay;
bool _isBTaxEnabled;
bool _isSTaxEnabled;
```

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	Stelnar.sol#L317
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 _burn(address account, uint256 value) internal {
    if (account == address(0)) {
      revert ERC20InvalidSender(address(0));
    }
    _update(account, address(0), value);
  }
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	Stelnar.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 ^0.8.19;
```

Recommendation

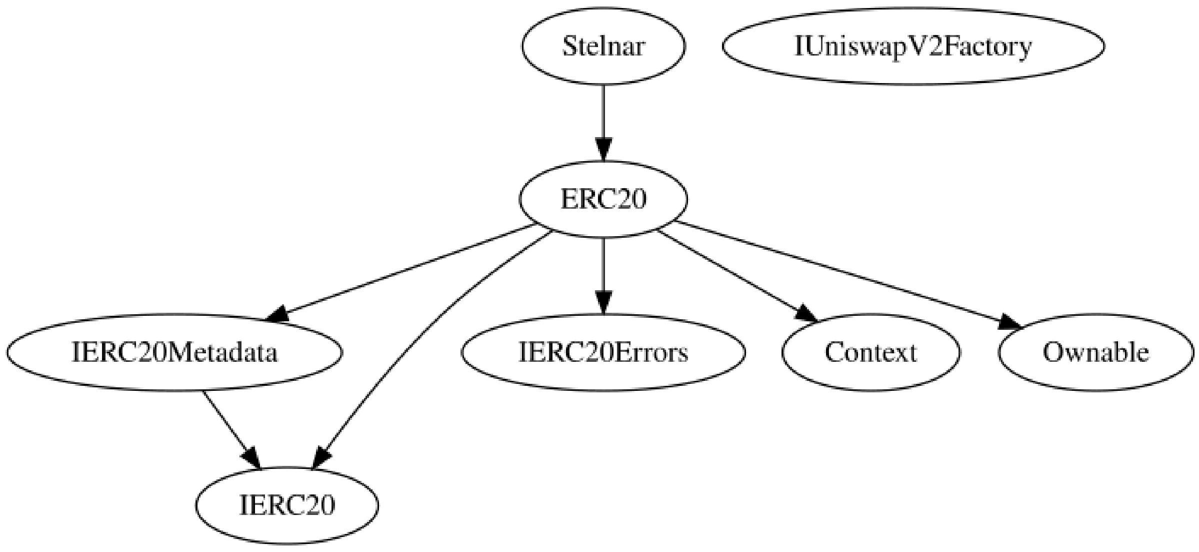
The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest 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

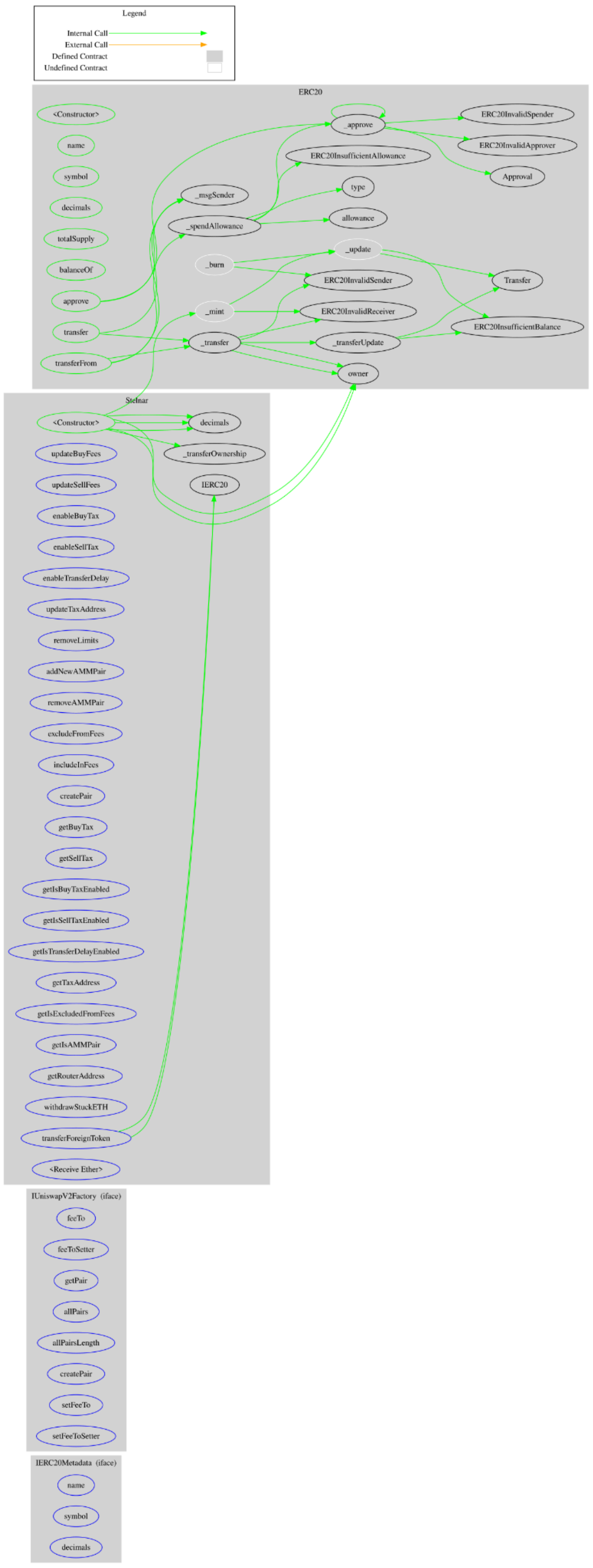
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Stelnar	Implementation	ERC20		
		Public	✓	ERC20
	updateBuyFees	External	✓	onlyOwner
	updateSellFees	External	✓	onlyOwner
	enableBuyTax	External	✓	onlyOwner
	enableSellTax	External	✓	onlyOwner
	enableTransferDelay	External	✓	onlyOwner
	updateTaxAddress	External	✓	onlyOwner
	removeLimits	External	✓	onlyOwner
	addNewAMMPair	External	✓	onlyOwner
	removeAMMPair	External	✓	onlyOwner
	excludeFromFees	External	✓	onlyOwner
	includeInFees	External	✓	onlyOwner
	createPair	External	✓	onlyOwner
	getBuyTax	External		-
	getSellTax	External		-
	getIsBuyTaxEnabled	External		-
	getIsSellTaxEnabled	External		-
	getIsTransferDelayEnabled	External		-

	getTaxAddress	External		-
	getIsExcludedFromFees	External		-
	getIsAMMPair	External		-
	getRouterAddress	External		-
	withdrawStuckETH	External	✓	onlyOwner
	transferForeignToken	External	✓	onlyOwner
		External	Payable	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
approve
transfer
transferFrom
_approve
ERC20InvalidSpender
ERC20InvalidApprover
Approval
ERC20InsufficientAllowance
_msgSender
_spendAllowance
type
allowance
_update
_burn
ERC20InvalidSender
Transfer
_mint
ERC20InvalidReceiver
ERC20InsufficientBalance
_transfer
_transferUpdate
owner
Stelnar
<Constructor>
decimals
updateBuyFees
_transferOwnership
updateSellFees
IERC20
enableBuyTax
enableSellTax
enableTransferDelay
updateTaxAddress
removeLimits
addNewAMMPair
removeAMMPair
excludeFromFees
includeInFees
createPair
getBuyTax
getSellTax
getIsBuyTaxEnabled
getIsSellTaxEnabled
getIsTransferDelayEnabled
getTaxAddress
getIsExcludedFromFees
getIsAMMPair
getRouterAddress
withdrawStuckETH
transferForeignToken
<Receive Ether>
IUniswapV2Factory (iface)
feeTo
feeToSetter
getPair
allPairs
allPairsLength
createPair
setFeeTo
setFeeToSetter
IERC20Metadata (iface)
name
symbol
decimals

Summary

Stelnar 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 stop transactions. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of max 20% fees.

Disclaimer

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