expelee

A Secure Place For Web3

SMART CONTRACT AUDIT OF

GET SCHIFFY GOLD

Contract Address

0x194460B46315D4C61F52E5100560E62EB958D182

Audit Summary

Expelee team has performed a line-by-line manual analysis and automated review of the smart contract. The smart contract was analysed mainly for common smart contract vulnerabilities, exploits, and manipulation hacks. According to the smart contract audit:

Audit Result: **PASSED**

Ownership: **NOT RENOUNCED**

KYC Verification: Done

Audit Date: 28/06/2022

Audit Team: **EXPELEE**

Be aware that smart contracts deployed on the blockchain aren't resistant to internal exploit, external vulnerability, or hack. For a detailed understanding of risk severity, source code vulnerability, functional hack, and audit disclaimer, kindly refer to the audit.

DISCLAMER

All the content provided in this document is for general information only and should not be used as financial advice or a reason to buy any investment. Team provides no guarantees against the sale of team tokens or the removal of liquidity by the project audited in this document.

Always Do your own research and protect yourselves from being scammed. The Expelee team has audited this project for general information and only expresses their opinion based on similar projects and checks from popular diagnostic tools.

Under no circumstances did Expelee receive a payment to manipulate those results or change the awarding badge that we will be adding in our website. Always Do your own research and protect yourselves from scams.

This document should not be presented as a reason to buy or not buy any particular token. The Expelee team disclaims any liability for the resulting losses.

Contract Review

Contract Name	GetSchiffyGold
Compiler Version	v0.8.7+commit.e28d00a7
Optimization	Yes with 200 runs
License	Unlicense
Explorer	https://bscscan.com/address/0x194460B46315D4C61F52E5100560E62EB958D182#code
Symbol	GOLD
Decimals	18
Total Supply	1000,000,000,000
Domain	https://getschiffy.com/en

Project Review

Token Name: GOLD

Web Site: https://getschiffy.com/en

Twitter: https://twitter.com/getschiffy

Telegram: https://t.me/getschiffy

Contract Address:

0x194460B46315D4C61F52E5100560E62EB958D182

Platform: Binance Smart Chain

Token Type: BEP 20

Language: SOLIDITY

Audit Methodology

The scope of this report is to audit the smart contract source code. We have scanned the contract and reviewed the project for common vulnerabilities, exploits, hacks, and back-doors. Below is the list of commonly known smart contract vulnerabilities, exploits, and hacks:

Category

Smart Contract Vulnerabilities	- Unhandled Exceptions - Transaction Order Dependency - Integer Overflow - Unrestricted Action - Incorrect Inheritance Order - Typographical Errors - Requirement Violation
Source Code Review	- Gas Limit and Loops - Deployment Consistency - Repository Consistency - Data Consistency - Token Supply Manipulation
Functional Assessment	- Operations Trail & Event Generation - Assets Manipulation - Liquidity Access

Vulnerability Checklist

№	Description.	Result
1	Compiler warnings.	Passed
2	Race conditions and Re-entrancy. Cross-function raceconditions.	Passed
3	Possible delays in data delivery.	Passed
4	Oracle calls.	Passed
5	Front running.	Passed
6	Timestamp dependence.	Passed
7	Integer Overflow and Underflow.	Passed
8	DoS with Revert.	Passed
9	DoS with block gas limit.	Passed
10	Methods execution permissions.	Passed
11	Economy model.	Passed
12	The impact of the exchange rate on the logic.	Passed
13	Private user data leaks.	Passed
14	Malicious Event log.	Passed
15	Scoping and Declarations.	Passed
16	Uninitialized storage pointers.	Passed
17	Arithmetic accuracy.	Passed
18	Design Logic.	Passed
19	Cross-function race conditions.	Passed
20	Safe Zeppelin module.	Passed
21	Fallback function security.	Passed

Manual Audit

🟢 **Low-Risk**

🟢 4 low-risk code issues found

🟡 **Medium-Risk**

🟡 1 medium-risk code issues found

🔴 **High-Risk**

🔴 0 high-risk code issues found

Low-Risk

1) Contract contains Reentrancy vulnuerabilities

```
- uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this),_uniswapV2R State variables
written after the call(s):
- setExcludes() (#687)
- _isExcludedFromFee[owner()] = true (#693)
- _isExcludedFromFee[address(this)] = true (#694)
- _isExcludedFromFee[_playToEarnWallet] = true (#695)
- setExcludes() (#687)
- _isExcludedFromMaxTx[owner()] = true (#697)
- _isExcludedFromMaxTx[address(this)] = true (#698)
- _isExcludedFromMaxTx[_playToEarnWallet] = true (#699)
- uniswapV2Router = _uniswapV2Router (#685)
```

Recommendation

Apply the check-effects-interaction pattern

2) Missing events arithmethics

Detect missing events for critical arithmetic parameters .

```
function excludeFromFee(address account) public onlyOwner {
    _isExcludedFromFee[account] = true;
}

function includeInFee(address account) public onlyOwner {
    _isExcludedFromFee[account] = false;
}

function setTaxFeePercent(uint256 taxFee) external onlyOwner() {
    _taxFee = taxFee;
}

function setPlayToEarnFeePercent(uint256 liquidityFee) external onlyOwner() {
    _playToEarn = liquidityFee;
}

function setMaxTxPercent(uint256 maxTxPercent) external onlyOwner() {
    _maxTxAmount = _tTotal.mul(maxTxPercent).div(10**2);
}
```

Recommendation

Emit an event for critical parameter changes.

3) Costly Operations inside a Loop

Costly operations inside a loop might waste gas so, optimizations are justified

```
function includeInReward(address account) external onlyOwner() {
    require(_isExcluded[account], "Account is already excluded"); for (uint256
    i = 0; i &lt; _excluded.length; i++) {

        if (_excluded[i] == account) {

            _excluded[i] = _excluded[_excluded.length - 1];

            _tOwned[account] = 0;

            _isExcluded[account] = false;

            _excluded.pop(); break;

        }
    }

}
```

Recommendation

Use a local variable to hold the loop computation result.

4) Avoid relying on block.timestamp

block.timestamp can be manipulated by miners.

```
function lock(uint256 time) public virtual onlyOwner {

    _previousOwner = _owner;

    _owner = address(0);

    _lockTime = block.timestamp + time;

    emit OwnershipTransferred(_owner, address(0));

}


//Unlocks the contract for owner when _lockTime is exceeds function
unlock() public virtual {

    require(_previousOwner == msg.sender, "You don't have permission to unlock");
    require(block.timestamp &gt; _lockTime , "Contract is locked until 7 days"); emit
    OwnershipTransferred(_owner, _previousOwner);

    _owner = _previousOwner;

}
```

Recommendation

Do not use block.timestamp, now or blockhash as a source of randomness

Medium-Risk

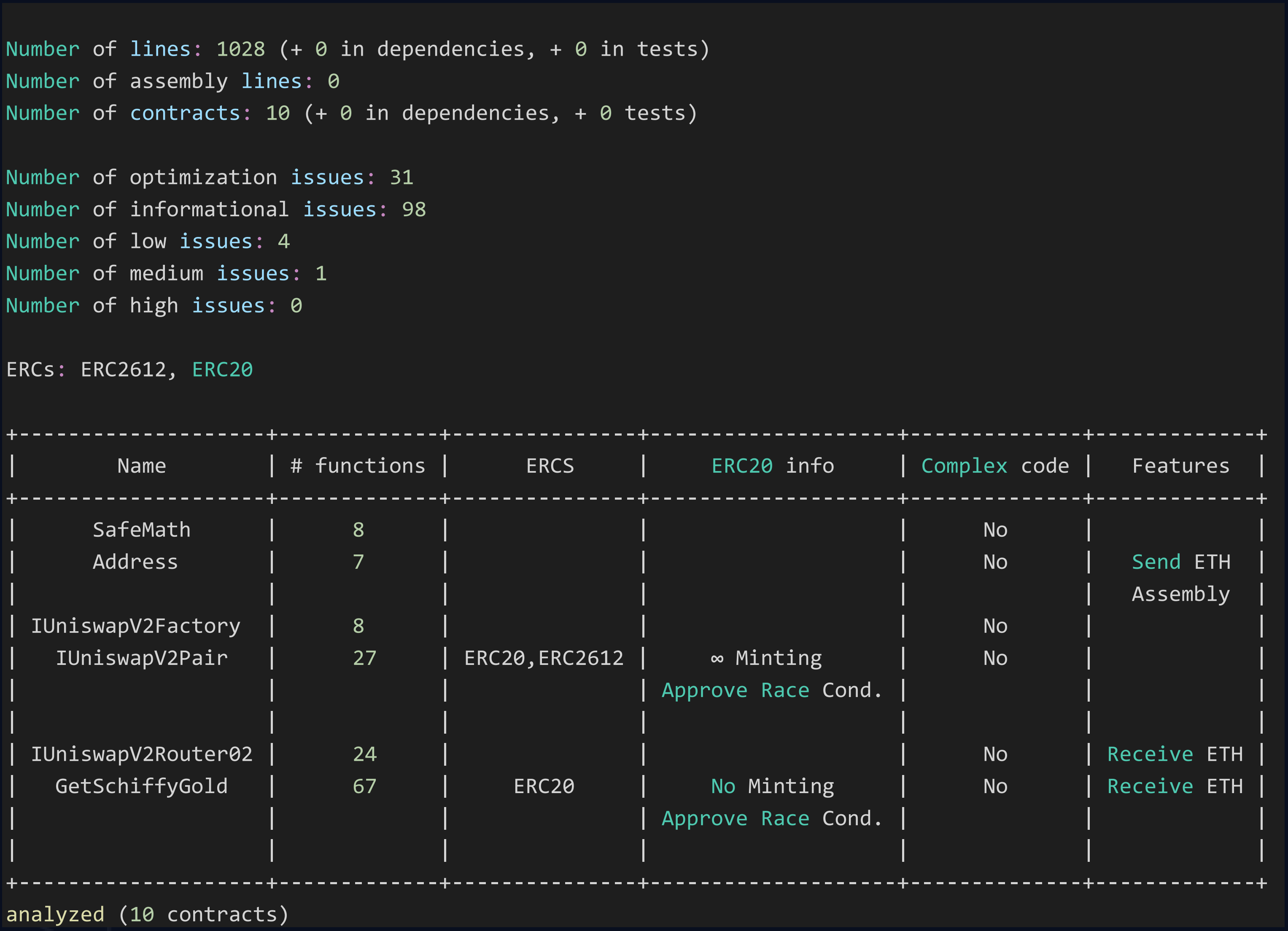
1) No check on Lock time

```
//Locks the contract for owner for the amount of time provided function
lock(uint256 time) public virtual onlyOwner {

    _previousOwner = _owner;

    _owner = address(0);

    _lockTime = block.timestamp + time;

    emit OwnershipTransferred(_owner, address(0));

}
```

Recommendation

Since you rennounce the ownership of the contract for an X amount of time, you should check this value (x) to be sure you won't unintentionally lock it for too long / infinity and you won't be able to get ownership of the contract back.

Manual Audit Summary



```
Number of lines: 1028 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 10 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 31
Number of informational issues: 98
Number of low issues: 4
Number of medium issues: 1
Number of high issues: 0

ERCs: ERC2612, ERC20

+--------------------+-------------+---------------+--------------------+--------------+-------------+
|        Name        | # functions |      ERCS     |     ERC20 info     | Complex code |   Features  |
+--------------------+-------------+---------------+--------------------+--------------+-------------+
|      SafeMath      |      8      |               |                    |      No      |             |
|      Address       |      7      |               |                    |      No      |   Send ETH  |
|                    |             |               |                    |              |   Assembly  |
| IUniswapV2Factory  |      8      |               |                    |      No      |             |
|   IUniswapV2Pair   |      27     | ERC20,ERC2612 |     ∞ Minting      |      No      |             |
|                    |             |               | Approve Race Cond. |              |             |
|                    |             |               |                    |              |             |
| IUniswapV2Router02 |      24     |               |                    |      No      | Receive ETH |
|   GetSchiffyGold   |      67     |     ERC20     |     No Minting     |      No      | Receive ETH |
|                    |             |               | Approve Race Cond. |              |             |
|                    |             |               |                    |              |             |
+--------------------+-------------+---------------+--------------------+--------------+-------------+
analyzed (10 contracts)
```

Important Points To Consider

✓ The owner cannot mint tokens after Initial

✓ The owner cannot stop Trading.

✓ Verified contract source

✗ Token is not sellable (not a honeypot) at this time

✗ Ownership NOT renounced or source does contain an ownership functionality

✗ Source does contain a max transaction count

✗ Source does contain a fee modifier

✓ Owner/creator wallet contains less than 10% of circulating token supply (0%)

✓ All other holders possess less than 10% of circulating token supply

About Expelee

Expelee is a community driven organisation dedicated to fostering an anti-rug movement. We're here to keep investment safe from fraudsters. We've encountered several rug pulls and know how it feels to be duped, which is why we don't want anybody else to go through the same experience. We are here to raise awareness through our services so that the future of cryptocurrency can be rug-free.

The auditing process focuses to the following considerations with collaboration of an expert team:

- Functionality test of the Smart Contract to determine if proper logic has been followed throughout the whole process.
- Manually detailed examination of the code line by line by experts.
- Live test by multiple clients using Test net.
- Analysing failure preparations to check how the Smart
- Contract performs in case of any bugs and vulnerabilities.
- Checking whether all the libraries used in the code are on the latest version.
- Analysing the security of the on-chain data.

Social Media

/Expelee

/ExpeleeOfficial

/expelee-co