RugFreeCoins Audit

Hodl4Gold Token

Smart Contract Security Audit

December 29, 2021

Contents

Audit details	1
Disclaimer	2
Background	3
About the project	4
Target market and the concept	8
Potential to grow with score points	10
Total Points	10
Contract details	11
Contract code function details	12
Contract description table	13
Security issue checking status	26
Owner privileges	29
Audit conclusion	35

Audit details

Audited project

Hodl4Gold Token

Contract Address

0xE8c4bEce93084D649fB630886b5332942b643BB9

Client contact

Hodl4Gold Team

Blockchain

Binance smart chain

Project website

https://hold.rugfreecoins.com/

Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Rugfreecoins and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (Rugfreecoins) owe no duty of care towards you or any other person, nor does Rugfreecoins make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and Rugfreecoins hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, Rugfreecoins hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Rugfreecoins, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.

Background

Rugfreecoins was commissioned by Hodl4Gold Token to perform an audit of the smart contract.

https://bscscan.com/address/0xE8c4bEce93084D649fB630886b5332942b643BB9

The focus of this audit is to verify that the smart contract is secure, resilient and working according to the specifications.

The information in this report should be used to understand the risk exposure of the smart contract, project feasibility, long term sustainability and as a guide to improve the security posture of the smart contract by remediating the issues that were identified.

About the project

Hodl4Gold is a token built on the Binance Smart Chain that is with an innovative investment use case the main purpose of which is to seek out constant revenue sources, which in turn, powers reward combined with BUSD auto distribution, lottery system and auto burn. The projects is having 4 main use cases of NFT minting Dapp and the marketplace, lottery platform, play to earn game and reflection launchpad with the DEX platform. Each transaction, purchase and sale incur 20% fee.

Features

- The **BUSD rewards** will be distributed among every holder proportional to how many tokens each individual holds in values of **13% when buying and selling**.

- The additional component included under the sustainability section is a **liquidity fee of 3% from buying and selling**, which is a redistribution mechanism that ensures the trading pool always has sufficient liquidity.

- **The lottery fee is 1% when buying and selling** is what allows Hodl4Gold to become the most commonly known and recognized lottery token in the crypto sphere. In order for a cryptocurrency to grow and gain traction, especially in the Altcoin market, it must have a 'use-case', which only usually comes with the promise of a better future. The Hodl4Gold team is motivated by the idea that the coin will have a use-case from day one! The gambling industry has been around for centuries, and there will be an ever-growing crowd of 'gamblers' and players in the crypto sphere, as cryptocurrencies slowly become the staple in terms of money transactions around the world. In order to support this transition, Hodl4Gold wishes to establish itself as the most competitive and well-known lottery token in the industry, all the while progressively growing a following.

  With Hodl4Gold, the chances of winning are relative to how many tokens you hold, which means that all holders are incentivized to buy more tokens in the long term if they wish to increase their chances of winning the lottery.

  And later the platform will be enhanced to buying a ticket to entry with H4G. The H4G is sent to the burn wallet and contributes to the daily volume. Users can win by having from 2 Numbers to 6 Numbers correct in sequence. All funds not distributed compounds to the next week's Lotto jackpot growing the Jackpot to a Life-Changing Pot in BNB.

- The **sustainability fee of 1% when buying and selling for marketing and dev** is what allows Hodl4Gold to hold the aforementioned promise. Tokens will be swapped into BNB and will be sent to a marketing wallet per transaction. This way, Hodl4Gold will have enough funds to promote the coin and spend for future development without selling tokens as the traditional way.

- 1% of fee will be charged when buying and selling for the team expenses.

- Hodl4Gold has the burn strategy that a **1% fee in each transaction when selling** is getting charged that benefits and rewards those who invest long-term. This feature slowly reduces supply making each Kira Doge more and more valuable.

Tokenomics

20% fee when buying & selling

- 13% of trade goes to holders pockets in BUSD.
- 3% of trade goes to the liquidity pool.
- 1% of trade goes to the burn wallet.
- 1% of trade goes to development and marketing.
- 1% of trade goes to the team wallet.
- 1% trade goes to the lottery pool.

Roadmap


Phase 1
Develop Contract
Establishment of all the social networks
Marketing campaign
First audit
Publish website, roadmap and whitepaper
Marketing campaign
Private sale/Seed round
Lottery platform development
NFT Minting Dapp and Marketplace development
Testing Dapps
Pre-sale on Pinksale
Token launch on PancakeSwap
Lottery platform launch
NFT Minting Dapp and Marketplace launch
BUSD rewards dashboard launch
Listing Coingecko & Coinmarketcap
Weekly lottery drawing
Kick-off DEX development
Kick-off the reflection launchpad development


Phase 2
Release the native token DEX platform
Influencer marketing
Extensive marketing campaign Partnerships
Reflection launchpad beta version release
Start development of the NFT Game
Start release of first Game NFT's on marketplace

Phase 3

- More extensive marketing campaigns across the world
- Onboard more buyers and sellers to the project
- Complete DEX platform release
- Complete reflection launchpad release
- Complete game platform release

Target market and the concept

Target market

- Anyone who's interested in the Crypto space with long-term investment plans.
- Anyone who's ready to earn a passive income in BUSD by holding tokens.
- Anyone who's interested in trading tokens.
- Anyone who is ready to hold and be eligible to win in the daily lottery
- Anyone who is ready to hold a large portion of tokens and be eligible to get a high chance of winning in the lottery.
- Anyone who's interested in collecting NFTs or trading NFTs.
- Anyone who wants mint any design through Hodl4Gold minting Dapp platform.
- Anyone who's interested in taking part with the future plans of the Hodl4Gold token.
- Anyone who wants to be part with the Hodl4Gold NFT marketplace with buying and selling.
- Anyone who's interested in taking part with Hodl4Gold reflection launchpad.
- Anyone who's interested in taking part with Hodl4Gold play to earn game and win rewards.
- Anyone who's interested in making financial transactions with any other party using BUSD, BNB or Hodl4Gold as the currency.

Core concept

The Hodl4Gold reward system

13% of each transaction when buying and selling get converted to BUSD and is split amongst all holders. Holders will be eligible to receive tokens every one hour and rewards are proportional to how many tokens each individual holds.

Sustainable mechanism

The **sustainability fee of 1% when buying and selling for marketing and dev and 1% fee for the team** is what allows Hodl4Gold to promote the token and use funds to further the development of the platform. Tokens will be swapped into BNB and will be sent to a marketing wallet per transaction. This way, Hodl4Gold will have access to the funds without selling tokens as the traditional way, which will enable them to consume funds without hurting the project.

The liquidity fee of 3%, which is a redistribution mechanism that ensures the trading pool always has sufficient liquidity.

Hodl4Gold has the burn strategy that **1% fee in each transaction when selling** is getting charged that benefits and rewards those who invest long-term. This feature slowly reduces supply making each Kira Doge more and more valuable.

Lottery pool & lottery platform

The concept encourages,

- Investors, to hold the token for a long time, which makes them believe in the project and keep the hopes high of expecting to win a huge prize at once.
- Investors buy more and more since the chance of winning is higher.
- The concept is revolutionary and certainly can get the attraction of new investors as the project progresses along.
- Project market price and market cap can keep stable if everything goes according to the plan since keeping tokens will seem more profitable than selling.

Weekly Lottery drawing

Hodl4Gold lottery is with a strong use case specifically targeting the gambling industry aiming for any long-term believers and holders to give a chance to be eligible for the weekly lottery and win. The most unique core part of the Hodl4Gold is that the chances of winning are relative to how many tokens investors hold, which means that all holders are incentivized to buy more tokens in the long term if they wish to increase their chances of winning the lottery.

How Chances of Winning are Calculated

Chances of winning will be calculated in indirect proportion to how many tokens each holder has. This means that having more tokens does increase your chances of winning, but not in a linear fashion. Instead, a logarithmic function will be used to convert the proportion of holdings that each investor has and calculate their chances of winning accordingly. This will lower the discrepancy in the probability of winning between a whale and a small investor while keeping our largest investors at an advantage.

No. of tokens	% chance of winning	Log transformation	% of winning (log transformation)
15	0.50	1.17609	0.36784
07	0.23	0.84510	0.26432
05	0.17	0.69897	0.21861
03	0.10	0.477120	0.14923
Total		**3.19728**	**1**

Lottery Drawing Dapp

The lottery platform will be visible in an interface, where all contract holders are visible with their wallet IDs and the number of tokens they hold. Holders can connect wallets and check the probability of winning against the rest of the holders.

Winners will be chosen on a random draw, live on video chat. The winners will be populated on the web with wallet IDs and the amounts they won.

Potential to grow with score points

1.	Project efficiency	10/10
2.	Project uniqueness	10/10
3	Information quality	10/10
4	Service quality	10/10
5	System quality	10/10
6	Impact on the community	10/10
7	Impact on the business	10/10
8	Preparing for the future	10/10
Total Points		**10/10**

Contract details

Token contract details for 29th December 2021

Contract name	Hodl4Gold
Contract address	0xE8c4bEce93084D649fB630886b5332942b643BB9
Token supply	1,000,000,000,000,000
Token ticker	H4G
Decimals	9
Token holders	1
Transaction count	2
Lottery wallet	0x9cec888042de4bf5941608b30a3da50a39fcf9a5
Dividend tracker	0x1c847588c8db79e1519838edac3c3741dcf348a3
Marketing wallet	0x4d9811c62a3c9299aae9ca573727b3f7c8730989
Team wallet	0x9a24337bbf6a553182692b5b9d20bc6818b1d853
Contract deployer address	0x8E5B6881A65ab67178BF7Bc4B5AbE6AD3fCce0dD
Contract's current owner address	0x500a3bfadb7308e932bb6d07f49c674d088c981b

Contract code function details

No	Category	Item	Result
1	Coding conventions	BRC20 Token standards	pass
		compile errors	pass
		Compiler version security	pass
		visibility specifiers	pass
		Gas consumption	pass
		SafeMath features	pass
		Fallback usage	pass
		tx.origin usage	pass
		deprecated items	pass
		Redundant code	pass
		Overriding variables	pass
2	Function call audit	Authorization of function call	pass
		Low level function (call/delegate call) security	pass
		Returned value security	pass
		Selfdestruct function security	pass
3	Business security	Access control of owners	pass
		Business logics	pass
		Business implementations	pass
4	Integer overflow/underflow		pass
5	Reentrancy		pass
6	Exceptional reachable state		pass
7	Transaction ordering dependence		pass
8	Block properties dependence		pass
9	Pseudo random number generator (PRNG)		pass
10	DoS (Denial of Service)		pass
11	Token vesting implementation		pass
12	Fake deposit		pass
13	Event security		pass

Contract description table

Below table represents the summary of the contracts and methods in the token contract. We scanned the whole contract and listed down all the Interfaces, functions and implementations with its visibility and mutability.

Contract	**Type**	**Bases**		
└	Function Name	Visibility	Mutability	Modifiers
Context	**Implementation**			
└	_msgSender	Internal 🔒		
└	_msgData	Internal 🔒		
Ownable	**Implementation**	**Context**		
└		Public ❗	🛑	NO❗
└	owner	Public ❗		NO❗
└	renounceOwnership	Public ❗	🛑	onlyOwner
└	transferOwnership	Public ❗	🛑	onlyOwner
IERC20	**Interface**			
└	totalSupply	External ❗		NO❗
└	balanceOf	External ❗		NO❗
└	transfer	External ❗	🛑	NO❗

└	allowance	External ❗		NO❗
└	approve	External ❗	🔘	NO❗
└	transferFrom	External ❗	🔘	NO❗
ERC20	**Implementation**	**Context, IERC20**		
└		Public ❗	🔘	NO❗
└	name	Public ❗		NO❗
└	symbol	Public ❗		NO❗
└	decimals	Public ❗		NO❗
└	totalSupply	Public ❗		NO❗
└	balanceOf	Public ❗		NO❗
└	transfer	Public ❗	🔘	NO❗
└	allowance	Public ❗		NO❗
└	approve	Public ❗	🔘	NO❗
└	transferFrom	Public ❗	🔘	NO❗
└	increaseAllowance	Public ❗	🔘	NO❗
└	decreaseAllowance	Public ❗	🔘	NO❗
└	_transfer	Internal 🔒	🔘	

└	_mint	Internal 🔒	🛑	
└	_burn	Internal 🔒	🛑	
└	_approve	Internal 🔒	🛑	
└	_setupDecimals	Internal 🔒	🛑	
└	_beforeTokenTransfer	Internal 🔒	🛑	
IDividendPayingToken	**Interface**			
└	dividendOf	External ❗		NO❗
└	distributeDividends	External ❗	💵	NO❗
└	withdrawDividend	External ❗	🛑	NO❗
IDividendPayingTokenOptional	**Interface**			
└	withdrawableDividendOf	External ❗		NO❗
└	withdrawnDividendOf	External ❗		NO❗
└	accumulativeDividendOf	External ❗		NO❗
DividendPayingToken	**Implementation**	**ERC20, IDividendPayingToken, IDividendPayingTokenOptional**		
└		Public ❗	🛑	ERC20

└		External ❗	💸	NO❗
└	distributeDividends	Public ❗	💸	NO❗
└	distributeDividends	Public ❗	🛑	NO❗
└	withdrawDividend	Public ❗	🛑	NO❗
└	setDividendTokenAddress	Public ❗	🛑	NO❗
└	_withdrawDividendOfUser	Internal 🔒	🛑	
└	dividendOf	Public ❗		NO❗
└	withdrawableDividendOf	Public ❗		NO❗
└	withdrawnDividendOf	Public ❗		NO❗
└	accumulativeDividendOf	Public ❗		NO❗
└	_transfer	Internal 🔒	🛑	
└	_mint	Internal 🔒	🛑	
└	_burn	Internal 🔒	🛑	
└	_setBalance	Internal 🔒	🛑	
IUniswapV2Factory	**Interface**			
└	feeTo	External ❗		NO❗
└	feeToSetter	External ❗		NO❗

└	getPair	External ❗		NO❗
└	allPairs	External ❗		NO❗
└	allPairsLength	External ❗		NO❗
└	createPair	External ❗	🔘	NO❗
└	setFeeTo	External ❗	🔘	NO❗
└	setFeeToSetter	External ❗	🔘	NO❗
IUniswapV2Pair	**Interface**			
└	name	External ❗		NO❗
└	symbol	External ❗		NO❗
└	decimals	External ❗		NO❗
└	totalSupply	External ❗		NO❗
└	balanceOf	External ❗		NO❗
└	allowance	External ❗		NO❗
└	approve	External ❗	🔘	NO❗
└	transfer	External ❗	🔘	NO❗
└	transferFrom	External ❗	🔘	NO❗
└	DOMAIN_SEPARATOR	External ❗		NO❗

└	PERMIT_TYPEHASH	External ❗		NO❗
└	nonces	External ❗		NO❗
└	permit	External ❗	🔘	NO❗
└	MINIMUM_LIQUIDITY	External ❗		NO❗
└	factory	External ❗		NO❗
└	token0	External ❗		NO❗
└	token1	External ❗		NO❗
└	getReserves	External ❗		NO❗
└	price0CumulativeLast	External ❗		NO❗
└	price1CumulativeLast	External ❗		NO❗
└	kLast	External ❗		NO❗
└	mint	External ❗	🔘	NO❗
└	burn	External ❗	🔘	NO❗
└	swap	External ❗	🔘	NO❗
└	skim	External ❗	🔘	NO❗
└	sync	External ❗	🔘	NO❗
└	initialize	External ❗	🔘	NO❗

IUniswapV2Router01	Interface			
└	factory	External ❗		NO❗
└	WETH	External ❗		NO❗
└	addLiquidity	External ❗	🛑	NO❗
└	addLiquidityETH	External ❗	💵	NO❗
└	removeLiquidity	External ❗	🛑	NO❗
└	removeLiquidityETH	External ❗	🛑	NO❗
└	removeLiquidityWithPermit	External ❗	🛑	NO❗
└	removeLiquidityETHWithPermit	External ❗	🛑	NO❗
└	swapExactTokensForTokens	External ❗	🛑	NO❗
└	swapTokensForExactTokens	External ❗	🛑	NO❗
└	swapExactETHForTokens	External ❗	💵	NO❗
└	swapTokensForExactETH	External ❗	🛑	NO❗
└	swapExactTokensForETH	External ❗	🛑	NO❗
└	swapETHForExactTokens	External ❗	💵	NO❗
└	quote	External ❗		NO❗
└	getAmountOut	External ❗		NO❗
└	getAmountIn	External ❗		NO❗

└	getAmountsOut	External ❗		NO❗
└	getAmountsIn	External ❗		NO❗
IUniswapV2Router02	**Interface**	**IUniswapV2Router01**		
└	removeLiquidityETHSupportingFeeOnTransferTokens	External ❗	🔘	NO❗
└	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External ❗	🔘	NO❗
└	swapExactTokensForTokensSupportingFeeOnTransferTokens	External ❗	🔘	NO❗
└	swapExactETHForTokensSupportingFeeOnTransferTokens	External ❗	💸	NO❗
└	swapExactTokensForETHSupportingFeeOnTransferTokens	External ❗	🔘	NO❗
IterableMapping	**Library**			
└	get	Public ❗		NO❗
└	getIndexOfKey	Public ❗		NO❗
└	getKeyAtIndex	Public ❗		NO❗
└	size	Public ❗		NO❗
└	set	Public ❗	🔘	NO❗

└	remove	Public ❗	🛑	NO❗
SafeMath	**Library**			
└	tryAdd	Internal 🔒		
└	trySub	Internal 🔒		
└	tryMul	Internal 🔒		
└	tryDiv	Internal 🔒		
└	tryMod	Internal 🔒		
└	add	Internal 🔒		
└	sub	Internal 🔒		
└	mul	Internal 🔒		
└	div	Internal 🔒		
└	mod	Internal 🔒		
└	sub	Internal 🔒		
└	div	Internal 🔒		
└	mod	Internal 🔒		
SafeMathInt	**Library**			
└	mul	Internal 🔒		

└	div	Internal 🔒		
└	sub	Internal 🔒		
└	add	Internal 🔒		
└	toUint256Safe	Internal 🔒		
SafeMathUint	**Library**			
└	toInt256Safe	Internal 🔒		
H4GToken	**Implementation**	**ERC20, Ownable**		
└		Public ❗	🛑	ERC20
└		External ❗	💸	NO❗
└	whitelistDxSale	Public ❗	🛑	onlyOwner
└	excludeFromDividends	External ❗	🛑	onlyOwner
└	includeInDividend	External ❗	🛑	onlyOwner
└	setTradingIsEnabled	Public ❗	🛑	onlyOwner
└	updateDividendTracker	Public ❗	🛑	onlyOwner
└	updateBuyFees	Public ❗	🛑	onlyOwner
└	updateSellFees	Public ❗	🛑	onlyOwner
└	updateUniswapV2Router	Public ❗	🛑	onlyOwner

└	updateMarketingAddress	Public		onlyOwner
└	updateTeamAddress	Public		onlyOwner
└	updateLotteryAddress	Public		onlyOwner
└	excludeFromFees	Public		onlyOwner
└	excludeMultipleAccountsFromFees	Public		onlyOwner
└	setAutomatedMarketMakerPair	Public		onlyOwner
└	_setAutomatedMarketMakerPair	Private		
└	updateGasForProcessing	Public		onlyOwner
└	updateClaimWait	External		onlyOwner
└	getClaimWait	External		NO
└	getTotalDividendsDistributed	External		NO
└	isExcludedFromFees	Public		NO
└	withdrawableDividendOf	Public		NO
└	dividendTokenBalanceOf	Public		NO
└	getAccountDividendsInfo	External		NO
└	getAccountDividendsInfoAtIndex	External		NO
└	processDividendTracker	External		NO
└	claim	External		NO

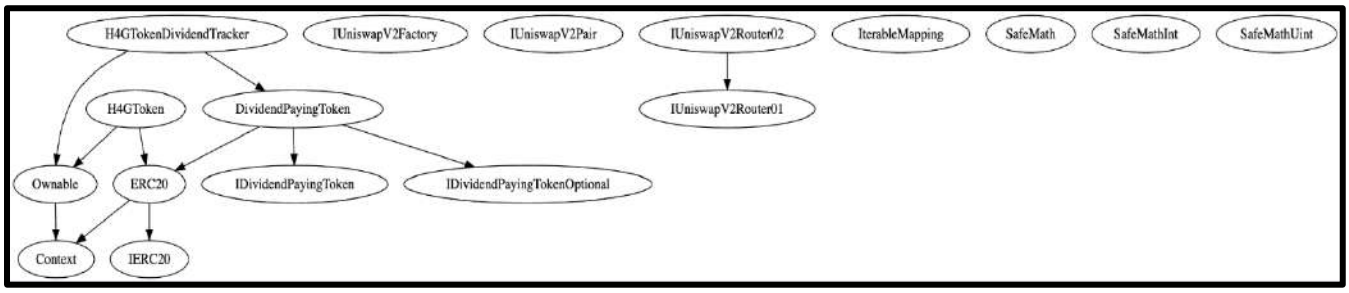
└	getLastProcessedIndex	External ❗		NO❗
└	getNumberOfDividendTokenHolders	External ❗		NO❗
└	isBlackListed	Public ❗		NO❗
└	blacklistUpdate	Public ❗	🔘	onlyOwner
└	_transfer	Internal 🔒	🔘	
└	swapAndSendBnb	Private 🔐	🔘	
└	swapTokensForBNB	Private 🔐	🔘	
└	swapTokensForDividendToken	Private 🔐	🔘	
└	swapAndSendDividends	Private 🔐	🔘	
└	swapAndSendDividendsInBNB	Private 🔐	🔘	
└	transferToWallet	Private 🔐	🔘	
└	swapAndLiquify	Private 🔐	🔘	
└	addLiquidity	Private 🔐	🔘	
H4GTokenDividendTracker	**Implementation**	**DividendPayingToken, Ownable**		
└		Public ❗	🔘	DividendPayingToken
└	_transfer	Internal 🔒		
└	withdrawDividend	Public ❗		NO❗

└	excludeFromDividends	External ❗	🔘	onlyOwner
└	includeInDividends	External ❗	🔘	onlyOwner
└	updateClaimWait	External ❗	🔘	onlyOwner
└	getLastProcessedIndex	External ❗		NO❗
└	getNumberOfTokenHolders	External ❗		NO❗
└	getAccount	Public ❗		NO❗
└	getAccountAtIndex	Public ❗		NO❗
└	canAutoClaim	Private 🔐		
└	setBalance	External ❗	🔘	onlyOwner
└	process	Public ❗	🔘	NO❗
└	processAccount	Public ❗	🔘	onlyOwner

Legend

Symbol	Meaning
🔴	**Function can modify state**
💵	**Function is payable**

Inheritance Hierarchy



Security issue checking status

- **High severity issues**
  No high severity issues found.

- **Medium severity issues**
  No medium severity issues found.

- **Low severity issues**
  No low severity issues found.

- **Informational**

  - The owner can enable/disable trading

```
ftrace | funcSig
function setTradingIsEnabled(bool _enabled⬆) public onlyOwner {
    tradingIsEnabled = _enabled⬆;
    emit SwapAndLiquifyEnabledUpdated(_enabled⬆);
}
```

- The owner can update buy fees

```
ftrace | funcSig
function updateBuyFees(
    uint256 reward,
    uint256 team,
    uint256 lottery,
    uint256 liquidity,
    uint256 burn,
    uint256 marketing
) public onlyOwner {
    buyDividendRewardsFee = reward;
    buyLotteryFee = lottery;
    buyTeamFee = team;
    buyLiquidityFee = liquidity;
    buyBurnFee = burn;
    buyMarketingFee = marketing;
    buyTotalFees = reward
        .add(team)
        .add(liquidity)
        .add(burn)
        .add(marketing)
        .add(lottery);
}
```

- The owner can update sell fees.

Owner privileges

- The owner can whitelist pre sale address.

```
ftrace | funcSig
function whitelistDxSale(address _presaleAddress⬆) public onlyOwner {
    presaleAddress = _presaleAddress⬆;
    dividendTracker.excludeFromDividends(_presaleAddress⬆);
    excludeFromFees(_presaleAddress⬆, true);
}
```

- The owner can exclude and include wallets from dividends.

```
ftrace | funcSig
function excludeFromDividends(address wallet⬆) external onlyOwner {
    dividendTracker.excludeFromDividends(wallet⬆);
}

ftrace | funcSig
function includeInDividend(address wallet⬆) external onlyOwner {
    dividendTracker.includeInDividends(wallet⬆, balanceOf(wallet⬆));
}
```

- The owner can enable/disable trading.

```
ftrace | funcSig
function setTradingIsEnabled(bool _enabled⬆) public onlyOwner {
    tradingIsEnabled = _enabled⬆;
    emit SwapAndLiquifyEnabledUpdated(_enabled⬆);
}
```

❖ The owner can update the dividend tracker.

```
ftrace | funcSig
function updateDividendTracker(address newAddress⬆) public onlyOwner {
    require(
        newAddress⬆ != address(dividendTracker),
        "H4G: The dividend tracker already has that address"
    );

    H4GTokenDividendTracker newDividendTracker = H4GTokenDividendTracker(
        payable(newAddress⬆)
    );

    require(
        newDividendTracker.owner() == address(this),
        "H4G: The new dividend tracker must be owned by the H4G token contract"
    );

    newDividendTracker.excludeFromDividends(address(newDividendTracker));
    newDividendTracker.excludeFromDividends(address(this));
    newDividendTracker.excludeFromDividends(address(uniswapV2Router));

    emit UpdateDividendTracker(newAddress⬆, address(dividendTracker));

    dividendTracker = newDividendTracker;
}
```

❖ The owner can update buy fees.

```
ftrace | funcSig
function updateBuyFees(
    uint256 reward⬆,
    uint256 team⬆,
    uint256 lottery⬆,
    uint256 liquidity⬆,
    uint256 burn⬆,
    uint256 marketing⬆
) public onlyOwner {
    buyDividendRewardsFee = reward⬆;
    buyLotteryFee = lottery⬆;
    buyTeamFee = team⬆;
    buyLiquidityFee = liquidity⬆;
    buyBurnFee = burn⬆;
    buyMarketingFee = marketing⬆;
    buyTotalFees = reward⬆
        .add(team⬆)
        .add(liquidity⬆)
        .add(burn⬆)
        .add(marketing⬆)
        .add(lottery⬆);
}
```

❖ The owner can update sell fees.

```
ftrace | funcSig
function updateSellFees(
    uint256 reward⬆,
    uint256 team⬆,
    uint256 lottery⬆,
    uint256 liquidity⬆,
    uint256 burn⬆,
    uint256 marketing⬆
) public onlyOwner {
    sellDividendRewardsFee = reward⬆;
    sellLotteryFee = lottery⬆;
    selTeamFee = team⬆;
    sellLiquidityFee = liquidity⬆;
    sellBurnFee = burn⬆;
    sellMarketingFee = marketing⬆;
    sellTotalFees = reward⬆
        .add(team⬆)
        .add(liquidity⬆)
        .add(burn⬆)
        .add(marketing⬆)
        .add(lottery⬆);
}
```

❖ The owner can update the router address.

```
ftrace | funcSig
function updateUniswapV2Router(address newAddress⬆) public onlyOwner {
    require(
        newAddress⬆ != address(uniswapV2Router),
        "H4G: The router already has that address"
    );
    emit UpdateUniswapV2Router(newAddress⬆, address(uniswapV2Router));
    uniswapV2Router = IUniswapV2Router02(newAddress⬆);
}
```

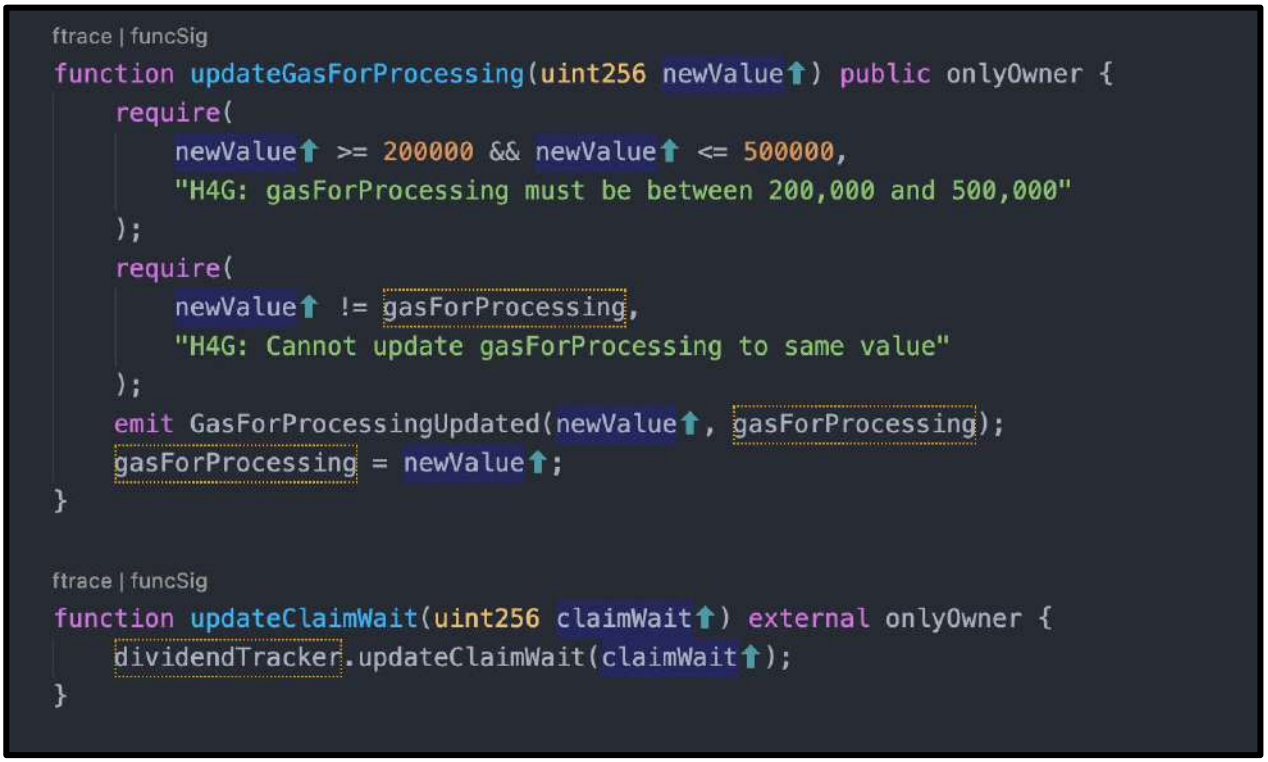
❖ The owner can update marketing, team and lottery address.

```
ftrace | funcSig
function updateMarketingAddress(address newAddress⬆) public onlyOwner {
    excludeFromFees(marketingWallet, false);
    marketingWallet = newAddress⬆;
    excludeFromFees(newAddress⬆, true);
}

ftrace | funcSig
function updateTeamAddress(address newAddress⬆) public onlyOwner {
    excludeFromFees(teamWallet, false);
    teamWallet = newAddress⬆;
    excludeFromFees(newAddress⬆, true);
}

ftrace | funcSig
function updateLotteryAddress(address newAddress⬆) public onlyOwner {
    excludeFromFees(lotteryWallet, false);
    lotteryWallet = newAddress⬆;
    excludeFromFees(newAddress⬆, true);
}
```

❖ The owner can exclude/include wallets from fees.

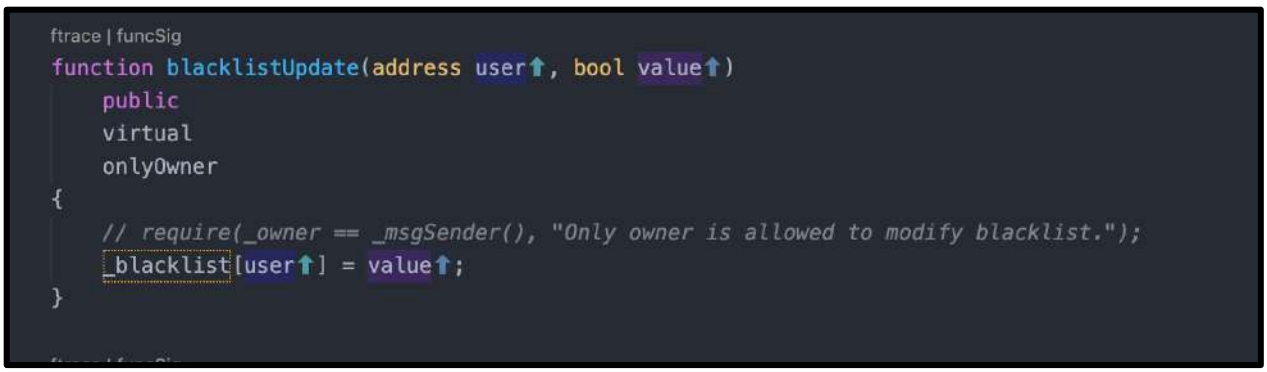
```
ftrace | funcSig
function excludeFromFees(address account⬆, bool excluded⬆) public onlyOwner {
    require(
        _isExcludedFromFees[account⬆] != excluded⬆,
        "H4G: Account is already the value of 'excluded'"
    );
    _isExcludedFromFees[account⬆] = excluded⬆;

    emit ExcludeFromFees(account⬆, excluded⬆);
}
```

❖ The owner can update max gas limit and minimum claim wait in the dividend tracker.



```
ftrace | funcSig
function updateGasForProcessing(uint256 newValue↑) public onlyOwner {
    require(
        newValue↑ >= 200000 && newValue↑ <= 500000,
        "H4G: gasForProcessing must be between 200,000 and 500,000"
    );
    require(
        newValue↑ != gasForProcessing,
        "H4G: Cannot update gasForProcessing to same value"
    );
    emit GasForProcessingUpdated(newValue↑, gasForProcessing);
    gasForProcessing = newValue↑;
}

ftrace | funcSig
function updateClaimWait(uint256 claimWait↑) external onlyOwner {
    dividendTracker.updateClaimWait(claimWait↑);
}
```

❖ The owner can blacklist wallets.



```
ftrace | funcSig
function blacklistUpdate(address user↑, bool value↑)
    public
    virtual
    onlyOwner
{
    // require(_owner == _msgSender(), "Only owner is allowed to modify blacklist.");
    _blacklist[user↑] = value↑;
}
```

❖ The owner can transfer and renounce ownership.

```
ftrace | funcSig
function renounceOwnership() public virtual onlyOwner {
    emit OwnershipTransferred(_owner, address(0));
    _owner = address(0);
}

/**
 * @dev Transfers ownership of the contract to a new account (`newOwner`).
 * Can only be called by the current owner.
 */
ftrace | funcSig
function transferOwnership(address newOwner) public virtual onlyOwner {
    require(
        newOwner != address(0),
        "Ownable: new owner is the zero address"
    );
    emit OwnershipTransferred(_owner, newOwner);
    _owner = newOwner;
}
```

Audit conclusion

While conducting the audit of the Hodl4Gold smart contract, it was observed that there is nothing alarming with the code, and it only contains Informational issues.