Cyberscope

Audit Report
GoldHunt

January 2023

Type	BEP20
Network	BSC
Address	0x6D3dd96f97b10fc86A7F8607704BBe9936083b89
Audited by	© cyberscope

Table of Contents

Review

Contract Name	GoldHunt
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	200 runs
Explorer	https://bscscan.com/address/0x6d3dd96f97b10fc86a7f8607704bbe9936083b89
Address	0x6d3dd96f97b10fc86a7f8607704bbe9936083b89
Network	BSC
Symbol	GHT
Decimals	18
Total Supply	100,000,000

Audit Updates

Initial Audit	30 Jan 2023

Source Files

Filename	SHA256
GoldHunt.sol	bee10229054ba4bc08132f8c3d40600b1ea49e6dde2000888c392920140940ba

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	OCTD	Transfers Contract's Tokens	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Unresolved
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

ST - Stops Transactions

Criticality	Critical
Location	GoldHunt.sol#L338,344
Status	Unresolved

Description

The contract utilizes the variable `rebaseMultiple`. If this variable is altered by the `rebase` function, it will cause the contract to halt transactions for all users. The following example depits this issue.

Assuming a user owns 10 tokens and the `rebase` function is triggered, setting the `rebaseMultiple` to 10. If the user tries to transfer these 10 tokens to another address, the transaction will fail because the rebaseMultiple will multiply the transfer amount to 100 tokens, which the user does not have.

```
function transfer(address to, uint256 amount) public virtual override
returns (bool) {
    address owner = _msgSender();
    _transfer(owner, to, amount * rebaseMultiple);
    return true;
}

function transferFrom( address from, address to, uint256 amount
) public virtual override returns (bool) {
  address spender = _msgSender();
  _spendAllowance(from, spender, amount * rebaseMultiple);
  _transfer(from, to, amount * rebaseMultiple);
  return true;
}
```

Recommendation

The contract could remove the `rebaseMultiple`. 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. That risk can be prevented by temporarily locking the contract or renouncing ownership.

MT - Mints Tokens

Criticality	Critical
Location	GoldHunt.sol#L552
Status	Unresolved

Description

The contract owner has the authority to mint tokens. The owner may take advantage of it by calling the `rebase` function. As a result, the contract tokens will be highly inflated.

```
function rebase(uint multiple) public onlyOwner{
    require(rewardsPool != address(0),"please set rewardsPool");
    uint rewardsPoolAmount = balanceOf(rewardsPool);

    _mint(rewardsPool,rewardsPoolAmount * (multiple - 1));

    uint newbaseMultiple = rebaseMultiple * multiple;
    emit Rebase(rebaseMultiple,newbaseMultiple);
    rebaseMultiple = newbaseMultiple;

    InterfaceLP(pair).sync();
}
```

Recommendation

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. That risk can be prevented by temporarily locking the contract or renouncing ownership.

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	MVN	Misleading Variables Naming	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	Unresolved

MVN - Misleading Variables Naming

Criticality	Minor / Informative
Location	GoldHunt.sol#L551
Status	Unresolved

Description

Variables and functions can have misleading names if their names do not accurately reflect the value they contain or the purpose they serve. The contract uses some variable names that are too generic or do not clearly convey the information stored in the variable. Misleading names can lead to confusion, making the code more making the code more difficult to read and understand.

The rebase function is not fulfilling its intended purpose. Instead, it is adding new tokens to the supply through a minting process.

```
event Rebase(uint beforeRebaseMultiple,uint newRebaseMultiple);
function rebase(uint multiple) public onlyOwner{ ... }
```

Recommendation

It's always a good practice for the contract to contain names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	GoldHunt.sol#L300
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `setSwapThreshold` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```
function setSwapThreshold(uint swapThreshold) public onlyOwner{
        _swapThreshold = swapThreshold * rebaseMultiple;
}
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the total supply. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	GoldHunt.sol#L283
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 usdt = 0x55d398326f99059fF775485246999027B3197955
```

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	GoldHunt.sol#L242,273,278,285,291,303
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.

```
function WETH() external pure returns (address);
address _rewardsPool
address _pair
address _router
address _marketingWallet
bool _swapEnabled
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	GoldHunt.sol#L297
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
_swapThreshold = swapThreshold * rebaseMultiple
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	GoldHunt.sol#L489
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 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero
address");

        _beforeTokenTransfer(account, address(0), amount);

        uint256 accountBalance = _balances[account];
...
            _totalSupply -= amount;
        }

        emit Transfer(account, address(0), amount / rebaseMultiple);

        _afterTokenTransfer(account, address(0), amount);
    }
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	GoldHunt.sol#L274,279,286,292
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
rewardsPool = _rewardsPool
pair = _pair
router = _router
marketingtWallet = _marketingWallet
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	GoldHunt.sol#L7,34,119,204,232
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```
pragma solidity ^0.8.0;
pragma solidity ^0.8.9;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	GoldHunt.sol#L7,34,119,204,232
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 ^0.8.0;
pragma solidity ^0.8.9;
```

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.

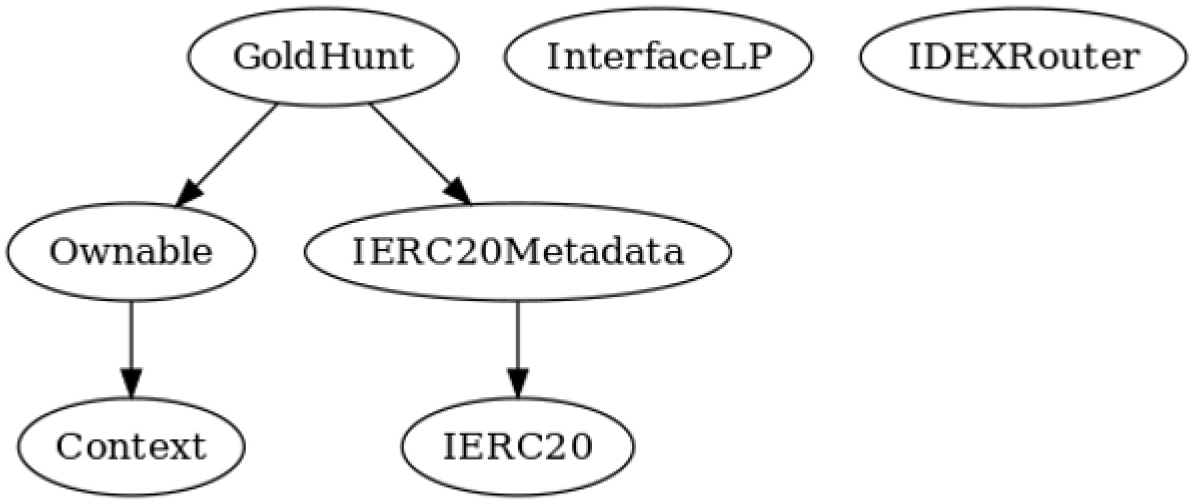
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-

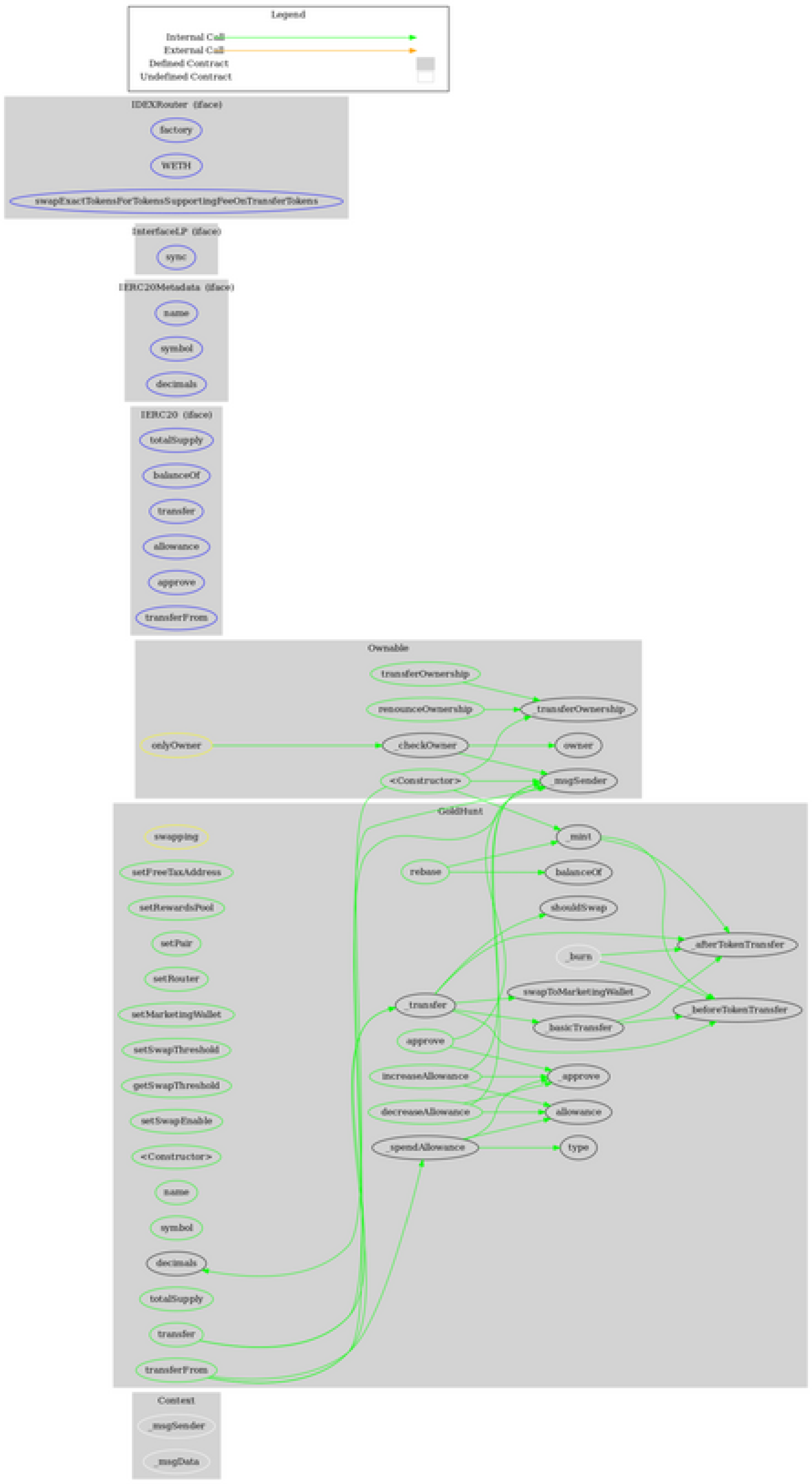
	decimals	External		-
InterfaceLP	Interface			
	sync	External	✓	-
IDEXRouter	Interface			
	factory	External		-
	WETH	External		-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
GoldHunt	Implementation	IERC20Metadata, Ownable		
	setFreeTaxAddress	Public	✓	onlyOwner
	setRewardsPool	Public	✓	onlyOwner
	setPair	Public	✓	onlyOwner
	setRouter	Public	✓	onlyOwner
	setMarketingWallet	Public	✓	onlyOwner
	setSwapThreshold	Public	✓	onlyOwner
	getSwapThreshold	Public		-
	setSwapEnable	Public	✓	onlyOwner
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	transferFrom	Public	✓	-
	allowance	Public		-

	approve	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_basicTransfer	Internal	✓	
	shouldSwap	Internal		
	swapToMarketingWallet	Internal	✓	swapping
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
	rebase	Public	✓	onlyOwner

Inheritance Graph


$$\begin{tikzcd}
 & GoldHunt \arrow[dl] \arrow[d] & InterfaceLP & IDEXRouter \\
Ownable \arrow[d] & IERC20Metadata \arrow[d] & & \\
Context & IERC20 & &
\end{tikzcd}$$

Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IDEXRouter (iface)
factory
WETH
swapExactTokensForTokensSupportingFeeOnTransferTokens
InterfaceLP (iface)
sync
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Ownable
transferOwnership
renounceOwnership
_transferOwnership
onlyOwner
_checkOwner
owner
<Constructor>
_msgSender
GoldHunt
swapping
setFreeTaxAddress
setRewardsPool
setPair
setRouter
setMarketingWallet
setSwapThreshold
getSwapThreshold
setSwapEnable
<Constructor>
name
symbol
decimals
totalSupply
transfer
transferFrom
_mint
rebase
balanceOf
shouldSwap
_burn
_afterTokenTransfer
swapToMarketingWallet
_transfer
_basicTransfer
_beforeTokenTransfer
approve
increaseAllowance
_approve
decreaseAllowance
allowance
_spendAllowance
type
Context
_msgSender
_msgData

Summary

There are some functions that can be abused by the owner like stop transactions and mint tokens. The contract can be converted into a honeypot and prevent users from selling if the owner abuses the admin functions. if the contract owner abuses the mint functionality, then the contract will be highly inflated. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

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