Cyberscope

Audit Report

Howcat Dex

June 2024

Network	BSC
Audited by	© cyberscope

Table of Contents

Review

HowswapRouter.sol	https://bscscan.com/address/0x84dfe8da2c41e8d536748bc0195d7fe6a39d3fae
HowswapFactory.sol	https://bscscan.com/address/0xF4866c04a88Cb593374865D73D7cF5B5B3A9c641
MasterChef.sol	https://bscscan.com/address/0xaf970baf55218a308b2a2c61bbae5a2dcd5481d8
SmartChefFactory.sol	https://bscscan.com/address/0x7b20c481c3d76d7bc30a103aff155543798ea924

Contract Name	HowswapToken
Compiler Version	v0.7.4+commit.3f05b770
Optimization	200 runs
Explorer	https://bscscan.com/address/0xb88af3f097e12e6509bc47ff3654ba7cc0716ac0
Address	0xb88af3f097e12e6509bc47ff3654ba7cc0716ac0
Network	BSC
Symbol	HCAT

Decimals	18
Total Supply	280,974,702.857

Contract Name	SyrupBar
Compiler Version	v0.7.4+commit.3f05b770
Optimization	200 runs
Explorer	https://bscscan.com/address/0x89b51a235fe6439cec719e8b12e69f8cf368a249
Address	0x89b51a235fe6439cec719e8b12e69f8cf368a249
Network	BSC
Symbol	SYRUP
Decimals	18
Total Supply	16,901,397.221

Audit Updates

Initial Audit	03 Jun 2024

Source Files

Filename	SHA256
HowswapRouter.sol	a5778028684c0c1cbcfaf657f8c37ae9760dc5475a99717319d60b1fae20a65d
HowswapFactory.sol	63200d81bda7e6ab04baf83b5cec600b28ac3ad1720243b7a442ec43fbcac05c
HowswapToken.sol	0ccaced0555284e675b6301d75a15c9b68e6e791cd76daeb32520f00212cee4c
SyrupBar.sol	1804cfde53dbf1b2d59d23d0c7f1b79af3944cdb5b70487741d70b8c38276ddf
MasterChef.sol	98fb0b2c2f949c527d3c046919aa69506c6afef6a02881eb7fec90434656e5b0
SmartChefFactory.sol	92a6251b64e4aa0e168f93de1e0d9b0b1dd4c66b59aa3276671bbdafc46acf9c

Overview

The HowCat project consists of a suite of six smart contracts that collectively form the infrastructure for their Decentralized Exchange (DEX) and Staking Pools. Below is a detailed description of each contract and their interconnections:

1. Howswap Reward Token - HowswapToken.sol
    a. The Howswap Token is the native reward token of the HowCat ecosystem. It is used to incentivize liquidity providers and stakers within the platform.
    b. This token is minted and distributed to users who provide liquidity to the DEX and participate in staking pools. The token can also be used for governance purposes within the HowCat community.
2. Howswap Decentralized Exchange
    a. HowswapFactory.sol
        i. The Factory contract is responsible for creating and managing trading pairs on the DEX.
        ii. It allows the creation of new trading pairs, records all the pairs, and assigns liquidity pools for each pair. The Factory contract ensures that each pair is unique and can be referenced by the Router contract.
    b. HowswapRouter.sol
        i. The Router contract facilitates token swaps and liquidity provision on the DEX.
        ii. It interacts with the Factory contract to find the appropriate pairs for swaps and liquidity operations. The Router uses the pairs created by the Factory to execute trades, add liquidity, and remove liquidity from the pools.
3. Howswap Staking

a. SmartChefFactory.sol

   i. The SmartChefFactory contract manages the creation of new staking pools.

   ii. It allows the deployment of new staking contracts, each with its unique reward and staking parameters. This factory pattern enables scalable and flexible staking pool management.

b. MasterChef.sol

   i. The MasterChef contract is the primary staking contract that governs the staking mechanics.

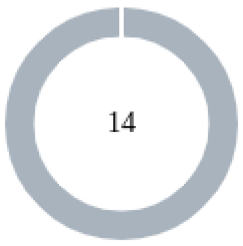
   ii. It manages user deposits, calculates rewards, and distributes the Howswap Tokens to stakers based on their stake and the staking period. The MasterChef contract is crucial for incentivizing users to lock their tokens in the staking pools.

c. SyrupBar.sol

   i. The SyrupBar contract is a supplementary ERC20 token.

   ii. It is minted to the users who stake their Howswap tokens to the masterchef as proof of stake, and they are burned when the user withdraws.

Findings Breakdown



●	Critical	0
●	Medium	0
●	Minor / Informative	14

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	0	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	14	0	0	0

Diagnostics

🔴 Critical 🟠 Medium ● Minor / Informative

Severity	Code	Description	Status
●	IDI	Immutable Declaration Improvement	Unresolved
●	MEM	Misleading Error Messages	Unresolved
●	TSI	Tokens Sufficiency Insurance	Unresolved
●	L01	Public Function could be Declared External	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L08	Tautology or Contradiction	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	MasterChef.sol#L1546,1547
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
cakePerBlock
startBlock
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	SmartChefFactory.sol#L1146,1147
Status	Unresolved

Description

The contract is using misleading error messages. These error messages do not accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(_stakedToken.totalSupply() >= 0)
require(_rewardToken.totalSupply() >= 0)
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

TSI - Tokens Sufficiency Insurance

Criticality	Minor / Informative
Location	SmartChefFactory.sol#L927,959,989 MasterChef.sol#L1677,1699,1717,1738
Status	Unresolved

Description

The tokens are not held within the contract itself. Instead, the contract is designed to provide the tokens from an external administrator. While external administration can provide flexibility, it introduces a dependency on the administrator's actions, which can lead to various issues and centralization risks.

```
rewardToken.safeTransfer(address(msg.sender), pending);
...
safeCakeTransfer(msg.sender, pending);
```

Recommendation

It is recommended to consider implementing a more decentralized and automated approach for handling the contract tokens. One possible solution is to hold the presale tokens within the contract itself. If the contract guarantees the process it can enhance its reliability, security, and participant trust, ultimately leading to a more successful and efficient process.

L01 - Public Function could be Declared External

Criticality	Minor / Informative
Location	HowswapRouter.sol#L810,820
Status	Unresolved

Description

A public function is a function that can be called from external contracts or from within the contract itself. An external function is a function that can only be called from external contracts, and cannot be called from within the contract itself.

It's generally a good idea to declare functions as external if they are only intended to be called from external contracts, as this can help make the contract's code easier to understand and maintain. Declaring a function as external can also help to improve the contract's performance and gas consumption.

```
function getAmountsOut(uint amountIn, address[] memory path)
        public
        view
        virtual
        override
        returns (uint[] memory amounts)
    {
        return PancakeLibrary.getAmountsOut(factory, amountIn, path);
    }

...
```

Recommendation

It's important to choose the appropriate visibility for each function based on how it is intended to be used. Declaring a function as external when it should be public, or vice versa can lead to unnecessary gas consumption.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	HowswapRouter.sol#L37,193,235,236,253,399 HowswapFactory.sol#L35,36,53,86,87,118,311,484,489 HowswapToken.sol#L856 MasterChef.sol#L962,1210,1215,1231,1522,1571,1587,1617,1622,1645,1663,1685,1706,1727,1747,1761 SyrupBar.sol#L857,1102,1107,1123 SmartChefFactory.sol#L803,830,870,871,872,873,874,875,876,911,941,984,994,1017,1034,1045,1064,1138,1139,1140,1141,1142,1143,1144
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);
function INIT_CODE_PAIR_HASH() external view returns (bytes32);
function DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint);
address public immutable override WETH
bytes32 public DOMAIN_SEPARATOR
address _token0
address _token1
address _feeTo
address _feeToSetter
address _to
uint256 _amount
address _from
uint256 public BONUS_MULTIPLIER = 1
IBEP20 _lpToken
bool _withUpdate
uint256 _allocPoint
uint256 _pid
address _user
address public SMART_CHEF_FACTORY
uint256 public PRECISION_FACTOR
IBEP20 _stakedToken
IBEP20 _rewardToken
uint256 _rewardPerBlock
uint256 _startBlock
uint256 _bonusEndBlock
uint256 _poolLimitPerUser
address _admin
uint256 _amount
uint256 _tokenAmount
address _tokenAddress
bool _hasUserLimit
address _user
```

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	MasterChef.sol#L1562,1576,1594 SmartChefFactory.sol#L886
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.

```
BONUS_MULTIPLIER = multiplierNumber
totalAllocPoint = totalAllocPoint.add(_allocPoint)
totalAllocPoint = totalAllocPoint.sub(prevAllocPoint).add(_allocPoint)
rewardPerBlock = _rewardPerBlock
```

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.

L08 - Tautology or Contradiction

Criticality	Minor / Informative
Location	SmartChefFactory.sol#L1146,1147
Status	Unresolved

Description

A tautology is a logical statement that is always true, regardless of the values of its variables. A contradiction is a logical statement that is always false, regardless of the values of its variables.

Using tautologies or contradictions can lead to unintended behavior and can make the code harder to understand and maintain. It is generally considered good practice to avoid tautologies and contradictions in the code.

```
require(_stakedToken.totalSupply() >= 0)
require(_rewardToken.totalSupply() >= 0)
```

Recommendation

The team is advised to carefully consider the logical conditions is using in the code and ensure that it is well-defined and make sense in the context of the smart contract.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	HowswapRouter.sol#L7 HowswapToken.sol#L357,362,400,429,455,465,484,498,508,801,840 MasterChef.sol#L171,176,341,367,396,410,490,506,515,944 SyrupBar.sol#L356,361,399,428,454,464,483,497,507,840 SmartChefFactory.sol#L535,561,590,624,634,652,662,739,755,764
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 safeApprove(address token, address to, uint value) internal {
        // bytes4(keccak256(bytes('approve(address,uint256)')));
        (bool success, bytes memory data) =
token.call(abi.encodeWithSelector(0x095ea7b3, to, value));
        require(success && (data.length == 0 || abi.decode(data,
(bool))), 'TransferHelper: APPROVE_FAILED');
    }
...
function min(uint256 x, uint256 y) internal pure returns (uint256 z) {
        z = x < y ? x : y;
    }
...
function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient
balance");

        // solhint-disable-next-line avoid-low-level-calls,
avoid-call-value
        (bool success, ) = recipient.call{value: amount}("");
        require(success, "Address: unable to send value, recipient may
have reverted");
    }
...
function functionCall(address target, bytes memory data) internal
returns (bytes memory) {
        return functionCall(target, data, "Address: low-level call
failed");
    }

...
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	MasterChef.sol#L1629,1630,1656,1658
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
uint256 cakeReward =
multiplier.mul(cakePerBlock).mul(pool.allocPoint).div(totalAllocPoint)
accCakePerShare =

accCakePerShare.add(cakeReward.mul(1e12).div(lpSupply))
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	HowswapRouter.sol#L407,408 HowswapFactory.sol#L313,314,460,486,491
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.

```
factory = _factory
WETH = _WETH
token0 = _token0
token1 = _token1
feeToSetter = _feeToSetter
feeTo = _feeTo
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	HowswapFactory.sol#L128,474 HowswapToken.sol#L407,526,1092 MasterChef.sol#L319,438,1196,1465 SyrupBar.sol#L406,525,1091,1357 SmartChefFactory.sol#L513,687,1154
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
            chainId := chainid
        }

assembly {
            pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }
assembly {
            codehash := extcodehash(account)
        }

assembly {
            let returndata_size := mload(returndata)
            revert(add(32, returndata), returndata_size)
        }
assembly {
            smartChefAddress := create2(0, add(bytecode, 32),
mload(bytecode), salt)
        }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	HowswapRouter.sol#L7,37,135,180,202,222,277,361,381,391 HowswapToken.sol#L5,28,95,192,381,541,854 MasterChef.sol#L5,195,293,454,552,576,644,960,1207,1477 SyrupBar.sol#L5,28,94,191,190,541,855,1100
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 >=0.6.0;
pragma solidity >=0.6.2;
pragma solidity >=0.5.0;
pragma solidity =0.6.6;
...
pragma solidity >=0.6.0 <0.8.0;
pragma solidity >=0.4.0;
pragma solidity >=0.6.6;
pragma solidity >0.6.6;
...
pragma solidity >=0.4.0;
pragma solidity >=0.6.6;
pragma solidity >=0.6.0;
pragma solidity >=0.6.0 <0.8.0;
pragma solidity >0.6.6;
pragma solidity >=0.6.12;
...
pragma solidity >=0.6.0 <0.8.0;
pragma solidity >=0.4.0;
pragma solidity >=0.6.6;
pragma solidity >0.6.6;
pragma solidity >=0.6.12;
```

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	HowswapRouter.sol#L7,37,135,180,202,222,277,361,381,391 HowswapToken.sol#L5,28,95,192,381,541,854 MasterChef.sol#L5,195,293,454,552,576,644,960,1207,1477 SyrupBar.sol#L5,28,94,191,190,541,855,1100 SmartChefFactory.sol#L5
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 >=0.6.0;
pragma solidity >=0.6.2;
pragma solidity >=0.5.0;
pragma solidity =0.6.6;
...
pragma solidity >=0.6.0 <0.8.0;
pragma solidity >=0.4.0;
pragma solidity >=0.6.6;
pragma solidity >0.6.6;
...
pragma solidity >=0.4.0;
pragma solidity >=0.6.6;
pragma solidity >=0.6.0;
pragma solidity >=0.6.0 <0.8.0;
pragma solidity >0.6.6;
pragma solidity >=0.6.12;
...
pragma solidity >=0.6.0 <0.8.0;
pragma solidity >=0.4.0;
pragma solidity >=0.6.6;
pragma solidity >0.6.6;
pragma solidity >=0.6.12;
...
pragma solidity >=0.6.0 <0.8.0;
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	HowswapRouter.sol#L496 MasterChef.sol#L1234,1236 SyrupBar.sol#L1126,1128
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
IPancakePair(pair).transferFrom(msg.sender, pair, liquidity)
...
cake.transfer(_to, cakeBal)
cake.transfer(_to, _amount)
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
HowswapRouter	Implementation	IPancakeRouter02		
		Public	✓	-
		External	Payable	-
	_addLiquidity	Internal	✓	
	addLiquidity	External	✓	ensure
	addLiquidityETH	External	Payable	ensure
	removeLiquidity	Public	✓	ensure
	removeLiquidityETH	Public	✓	ensure
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	removeLiquidityETHSupportingFeeOnTransferTokens	Public	✓	ensure
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	_swap	Internal	✓	
	swapExactTokensForTokens	External	✓	ensure
	swapTokensForExactTokens	External	✓	ensure
	swapExactETHForTokens	External	Payable	ensure
	swapTokensForExactETH	External	✓	ensure
	swapExactTokensForETH	External	✓	ensure

	swapETHForExactTokens	External	Payable	ensure
	_swapSupportingFeeOnTransferTokens	Internal	✓	
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	ensure
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	ensure
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	ensure
	quote	Public		-
	getAmountOut	Public		-
	getAmountIn	Public		-
	getAmountsOut	Public		-
	getAmountsIn	Public		-
HowswapFactory	Implementation	IPancakeFactory		
		Public	✓	-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
HowswapToken	Implementation	BEP20		
	mintFor	Public	✓	onlyOwner
	mint	Public	✓	onlyOwner
	delegates	External		-

	delegate	External	✓	-
	delegateBySig	External	✓	-
	getCurrentVotes	External		-
	getPriorVotes	External		-
	_delegate	Internal	✓	
	_moveDelegates	Internal	✓	
	_writeCheckpoint	Internal	✓	
	safe32	Internal		
	getChainId	Internal		
SyrupBar	Implementation	BEP20		
	mint	Public	✓	onlyOwner
	burn	Public	✓	onlyOwner
		Public	✓	-
	safeCakeTransfer	Public	✓	onlyOwner
	delegates	External		-
	delegate	External	✓	-
	delegateBySig	External	✓	-
	getCurrentVotes	External		-
	getPriorVotes	External		-
	_delegate	Internal	✓	
	_moveDelegates	Internal	✓	
	_writeCheckpoint	Internal	✓	

	safe32	Internal		
	getChainId	Internal		
MasterChef	Implementation	Ownable		
		Public	✓	-
	updateMultiplier	Public	✓	onlyOwner
	poolLength	External		-
	add	Public	✓	onlyOwner
	set	Public	✓	onlyOwner
	updateStakingPool	Internal	✓	
	getMultiplier	Public		-
	pendingCake	External		-
	massUpdatePools	Public	✓	-
	updatePool	Public	✓	-
	deposit	Public	✓	-
	withdraw	Public	✓	-
	enterStaking	Public	✓	-
	leaveStaking	Public	✓	-
	emergencyWithdraw	Public	✓	-
	safeCakeTransfer	Internal	✓	
	transferHowswapTokenOwnerShip	Public	✓	onlyOwner
	transferSyrupOwnerShip	Public	✓	onlyOwner

SmartChefFactory	Implementation	Ownable		
		Public	✓	-
	deployPool	External	✓	onlyOwner

Inheritance Graph

See the detailed images in the github repository.

Flow Graph

See the detailed images in the github repository.

Summary

Howcat is an interesting project that has a friendly and growing community. The Howcat ecosystem consists of their token, their decentralized exchange as well as their staking contracts. This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an “endorsement” or “disapproval” of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

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