



Cyberscope

A *TAC Security* Company

Audit Report

Good

September 2026

Network Robinhood

Address 0x5f62C57e5C537887117EeF828b7E3Ad41C009FEb

Audited by © cyberscope

Table of Contents

Table of Contents	1
Diagnostics	3
Risk Classification	4
Review	5
Audit Updates	5
Source Files	5
Scope	6
Findings Breakdown	7
CDRW - Collector Dependent Reward Withdrawal	8
Description	8
Recommendation	8
Team Update	8
USPS - Unguarded Select Pool Seeding	10
Description	10
Recommendation	10
Team Update	11
BEP - Blocked ETH Payouts	12
Description	12
Recommendation	12
Team Update	12
CO - Code Optimization	13
Description	13
Recommendation	13
Team Update	13
CR - Code Repetition	14
Description	14
Recommendation	14
Team Update	14
CCR - Contract Centralization Risk	15
Description	15
Recommendation	15
Team Update	15
IDI - Immutable Declaration Improvement	16
Description	16
Recommendation	16
Team Update	17
IVR - Inconsistent View Reporting	18
Description	18
Recommendation	18

Team Update	18
MEE - Missing Events Emission	19
Description	19
Recommendation	19
Team Update	19
RCL - Reward Checkpoint Loss	20
Description	20
Recommendation	20
Team Update	21
L05 - Unused State Variable	22
Description	22
Recommendation	22
L19 - Stable Compiler Version	23
Description	23
Recommendation	23
Functions Analysis	24
Summary	31
Disclaimer	32
About Cyberscope	33

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	CDRW	Collector Dependent Reward Withdrawal	Acknowledged
●	USPS	Unguarded Select Pool Seeding	Acknowledged
●	BEP	Blocked ETH Payouts	Acknowledged
●	CO	Code Optimization	Acknowledged
●	CR	Code Repetition	Acknowledged
●	CCR	Contract Centralization Risk	Acknowledged
●	IDI	Immutable Declaration Improvement	Acknowledged
●	IVR	Inconsistent View Reporting	Acknowledged
●	MEE	Missing Events Emission	Acknowledged
●	RCL	Reward Checkpoint Loss	Acknowledged
●	L05	Unused State Variable	Unresolved
●	L19	Stable Compiler Version	Unresolved

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation:** This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation:** This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical:** Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium:** Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor:** Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative:** Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
● Critical	Highly Likely / High Impact
● Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
● Minor / Informative	Unlikely / Low to no Impact

Review

Audit Updates

Initial Audit

14 Sep 2026

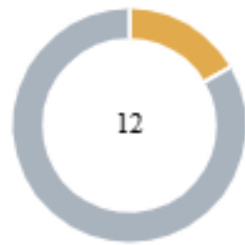
Source Files

Filename	SHA256
TokenSelectToken.sol	57c2d7938e5c229be3e022088b5645386d c8ed09d7d704f0b995f2387a84a709
libraries/V3MathLib.sol	80e6491bc7dd63d91bb06a1d21c1adfb3b 4c7c201706f7cc0e13871383a474e6
libraries/PresaleMath.sol	2459db1dc74b6bd6e069fe21c8f608b7d2 6181da6f9ea2e6ee29441a05bc5ce7
abstracts/RewardManager.sol	591a49a411445c0fb3c22f6f20e88161695 78ee638a82493027f6a8f1c7dbecd
abstracts/PresaleCore.sol	e2f52e843e28e59f02f1e323c5826352914 e247b17626073bf3eda3ffc5acc7d
abstracts/MigrationManager.sol	fcbc3981dc5ac9082eff362d99c791e0cff1 52a0aa008b82b00c083b4fe5b24c

Scope

This review covers the GOOD token in its deployed state as of 2 October 2026. The sale has closed and the migration has completed, with `migrationCompleted` and `selectPoolSeeded` both true. The token is a single deployed contract compiled from `TokenSelectToken.sol` together with the abstract contracts `PresaleCore.sol`, `MigrationManager.sol` and `RewardManager.sol`, and all four are within scope. Mechanisms belonging to the presale, deployment and migration lifecycle of that same contract are no longer reachable on this instance and are reported under the launchpad scope rather than here.

Findings Breakdown



● Critical	0
● Medium	2
● Minor / Informative	10

Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	0	0	0	0
● Medium	0	2	0	0
● Minor / Informative	2	8	0	0

CDRW - Collector Dependent Reward Withdrawal

Criticality	Medium
Location	abstracts/RewardManager.sol#L54,L82,L98,L121
Status	Acknowledged

Description

The two reward entry points both collect fees from the factory before paying anything, and the contract exposes no path to withdraw balances it has already recorded. A reverting collector therefore blocks a participant from taking rewards the token already holds for them, even when the recorded balances and checkpoints are sufficient. The collector is the factory, whose implementation is out of scope.

```
_collectAndUpdateFees();
```

Recommendation

The team is advised to ensure that a participant can withdraw entitlements the contract has already recorded without depending on a fresh external collection succeeding, while preserving the existing verification of newly collected amounts and the reentrancy protection on payouts.

Team Update

The team has acknowledged that this is not a security issue and states:

Applies. Every claim calls _collectAndUpdateFees() before paying, and no path exists to withdraw a balance the contract has already recorded, so a reverting collection blocks claiming. Preconditions: the factory or the treasury must be in a state where the collection call reverts. The reachable trigger is a treasury address that cannot receive plain ETH, because the factory pushes the service charge with a raw call and reverts on failure. A factory implementation that reverts in the collection path has the same effect. Neither is reachable by a third party. Consequence: the entire transaction reverts, so no state changes, no checkpoints advance and no balances are altered. Claiming is blocked while the condition holds and resumes once it is removed. The factory is upgradeable and setTreasury exists, so the condition is always reversible. No loss of funds is possible through this path.

For a claim to be blocked, the protocol's own treasury must reject ETH or the factory must revert during collection. The treasury is a Gnosis Safe, which accepts ETH, and every claim on GOOD has completed through this path. No third party can cause either condition. If one ever occurred, the whole transaction reverts, so no balance or checkpoint changes, and the factory owner can resolve it with setTreasury. There is no loss of funds in any case; claims are delayed until the fix, then resume in full. Under your matrix this is Unlikely with low impact, and we think that it should be rated Minor.

USPS - Unguarded Select Pool Seeding

Criticality	Medium
Location	abstracts/MigrationManager.sol#L171,354,362,381
Status	Acknowledged

Description

Two functions can seed the SELECT reserve, and only one is guarded. The retry path checks migration state, the seeded flag and the remaining balance. The other checks the caller and nothing else. Because the reserve amount is never reduced after a successful seed, each repeat call mints the full amount again, drawing on tokens that back presale claims and vesting allocations.

The supplied tick is checked for alignment and range, never against a price. Which price a tick represents flips with the token address ordering, which is decided at deployment.

```
function addSelectPoolLiquidity(int24 fairTick) external returns (uint256
nftId) {
    if (msg.sender != address(this) && msg.sender != factory) revert
OnlyFactory();
    nftId = __seedSelectPool(fairTick);
}
...
if (fairTick % tickSpacing != 0) revert InvalidFairTick();
if (fairTick < V3MathLib.minUsableTick(tickSpacing) || fairTick >
V3MathLib.maxUsableTick(tickSpacing)) revert InvalidFairTick();
```

Recommendation

The team is advised to guard every path that can seed the reserve with the same state checks, and to reject a tick that does not correspond to a sensible price for the deployment's token ordering.

Team Update

The team has acknowledged that this is not a security issue and states:

addSelectPoolLiquidity is not declared in ITokenSelectToken and no factory function calls it, so it is unreachable through the factory. Demonstrated in Appendix A. Independently, selectPoolSeeded is true, so _seedSelectPool cannot run again.

BEP - Blocked ETH Payouts

Criticality	Minor / Informative
Location	abstracts/RewardManager.sol#L217,250,292
Status	Acknowledged

Description

The three reward payout paths push ETH to the recipient with a raw call and revert when it fails. The recipient is whatever address contributed or was named at deploy, and a contract with no payable fallback, or one that spends more than the forwarded gas, rejects the transfer.

Whenever such an address holds a position, its own reward claim reverts on every attempt. The creator and referrer addresses are set at construction and cannot be changed, so a rejecting contract in either role blocks that share for the life of the token.

```
(bool success, ) = user.call{value: userETHReward}("");  
if (!success) revert RefundFailed();
```

Recommendation

The team is advised to separate the accounting of an amount owed from the act of delivering it, so a recipient unable to accept ETH does not lose access to funds already credited. This will keep one participant's address behavior from freezing their own balance.

Team Update

The team has acknowledged that this is not a security issue and states:

Requires a payout address that cannot receive plain ETH. creator() has no code, referrer() is address(0), and all 37 contributors accept a plain ETH transfer when simulated. The contributor set is frozen because contribute() reverts. No payout address on this token can reject ETH.

CO - Code Optimization

Criticality	Minor / Informative
Location	abstracts/PresaleCore.sol#L197-L198 abstracts/RewardManager.sol#L267
Status	Acknowledged

Description

There are code segments that could be optimized. The `contribute` function checks the incoming value twice, and since `MIN_CONTRIBUTION` is a positive constant any zero value already fails the second comparison, making the first statement redundant. The `_claimContributorRewards` function likewise reverts when the user has no contribution, but both of its callers only invoke it after establishing that the contribution is positive, so that check can never be the one that rejects a call.

```
if (msg.value == 0) revert InvalidContribution();  
if (msg.value < MIN_CONTRIBUTION) revert InvalidContribution();  
...  
if (contributions[user] == 0) revert NoContribution();
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

Team Update

The team has acknowledged that this is not a security issue and states:

A redundant comparison. No effect on behaviour or funds.

CR - Code Repetition

Criticality	Minor / Informative
Location	abstracts/RewardManager.sol#L42-L65,L70,L93
Status	Acknowledged

Description

The `claimRewards` and `claimRewardsFor` functions contain near-identical bodies: the same eligibility derivation, the same `_collectAndUpdateFees` invocation, and the same three-role payout sequence, differing only in whether the subject is `msg.sender` or the `user` argument and in the `onlyFactory` modifier. Duplicated logic increases bytecode size and creates a maintenance risk, since a future change applied to one path can silently diverge from the other.

```
function claimRewards() external nonReentrant {  
    ...  
    _collectAndUpdateFees();  
    ...  
    function claimRewardsFor(address user) external nonReentrant onlyFactory  
    {  
        ...  
        _collectAndUpdateFees();  
    }  
}
```

Recommendation

The team is advised to define the claim semantics once and have both entry points share that definition, while preserving the independent role branches, the authorisation on each entry point and the reentrancy protection.

Team Update

The team has acknowledged that this is not a security issue and states:

Duplicated function bodies. No effect on behaviour or funds.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	abstracts/PresaleCore.sol abstracts/MigrationManager.sol abstracts/RewardManager.sol
Status	Acknowledged

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

```
function migrate(address _treasury, int24 fairTick) external onlyFactory
returns (

function enableRefunds() external onlyFactory {

function claimRewardsFor(address user) external nonReentrant onlyFactory
{

function transferAllocations() external onlyStreamManager {
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

Team Update

The team has acknowledged that this is not a security issue and states:

Informational note on configuration. No new action is reachable that was not already reachable.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	abstracts/PresaleCore.sol#L150,L182 abstracts/MigrationManager.sol#L39,L50
Status	Acknowledged

Description

The contract stores thirty two values that are assigned exactly once during construction and never modified afterwards, yet none of them is declared immutable. In PresaleCore these are the role addresses, the raise and sale parameters, the allocations and the vesting schedule fields set in `_initializeCore`, and in MigrationManager the pool, WETH, factory, position manager, initialized tick and SELECT token fields set in `_initializeMigration` and `_initializeSelectTokenPool`. Declaring them immutable would embed the values in the runtime code, removing a storage read from every access, and would guarantee at the language level that they cannot change. The assignments are made in internal functions called from the constructor, so the change requires relocating them into the constructor body.

```
creator = params.creator;
factory = params.factory;
streamManager = params.streamManager;
targetETHRaise = params.targetETHRaise;
saleStartTime = params.saleStartTime;
saleEndTime = params.saleEndTime;
...
address public pool;
address internal weth;
IUniswapV3Factory internal uniswapV3Factory;
INonfungiblePositionManager internal positionManager;
```

Recommendation

The team is advised to make these values immutable so that they cannot change after construction and each access avoids a storage read.

Team Update

The team has acknowledged that this is not a security issue and states:

A gas and style observation on values already fixed at construction.

IVR - Inconsistent View Reporting

Criticality	Minor / Informative
Location	abstracts/PresaleCore.sol#L388 abstracts/RewardManager.sol#L54,347 abstracts/MigrationManager.sol#L188
Status	Acknowledged

Description

Two view functions disagree with the calls they describe. The readiness view tests the target and the migration flag but ignores `refundsEnabled`, while `migrate` rejects the call once that flag is set. The pending rewards view reads the fee totals as they stand, whereas a claim collects fresh fees first, so the view can report nothing owed while a claim pays out in full.

```
function isReadyForMigration() external view returns (bool) {
    return targetReached && !migrationCompleted;
}
...
if (refundsEnabled) revert MigrationAborted();
...
uint256 contributorETHSinceLastClaim = totalContributorETHFeesCollected -
lastETHClaimCheckpoint[user];
...
_collectAndUpdateFees();
```

Recommendation

The team is advised to align each view with the conditions the corresponding state changing function enforces. This will let an integrator rely on the reported state instead of discovering the difference through a reverted transaction.

Team Update

The team has acknowledged that this is not a security issue and states:

Concerns two view functions. Views move no funds and gate no path.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	abstracts/PresaleCore.sol#L310,L321
Status	Acknowledged

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
function transferAllocations() external onlyStreamManager {  
    ...  
    _transfer(address(this), streamManager, totalAllocations);  
}
```

Recommendation

The team is advised to emit dedicated events for the state changes this function performs, so that the token's own event stream is sufficient to follow its lifecycle without correlating logs from external contracts.

Team Update

The team has acknowledged that this is not a security issue and states:

Observability only.

RCL - Reward Checkpoint Loss

Criticality	Minor / Informative
Location	abstracts/RewardManager.sol#L70,279,288,289
Status	Acknowledged

Description

The contract scales each contributor's reward by their share of the raise, so the amount accrued since their last claim can floor to zero on one asset while the other still pays. The guard that skips a claim tests both assets together, so the claim proceeds and advances both `lastETHClaimCheckpoint` and `lastTokenClaimCheckpoint` to the current totals, discarding the entitlement on the asset that paid nothing. Because `claimRewardsFor` is callable by the `factory`, a contributor can be settled by another party at a time they do not choose, and each settlement discards whatever had not yet accumulated past the rounding floor. The same contribution claimed once at the end is paid in full, so what a contributor receives depends on how often they are settled rather than on what they are owed.

```
if (userETHReward == 0 && userTokenReward == 0) return false;
...
lastETHClaimCheckpoint[user] = totalContributorETHFeesCollected;
lastTokenClaimCheckpoint[user] = totalContributorTokenFeesCollected;
```

Recommendation

The team is advised to advance each reward checkpoint only by the amount actually paid on that leg, so an amount too small to pay out stays owed rather than discarded. This will stop a contributor's entitlement from being destroyed by the timing of a claim.

Team Update

The team has acknowledged that this is not a security issue and states:

A contributor's share is $\text{pool contribution} / \text{totalRaised}$, which can round down to zero on one asset while the other still pays. The skip condition tests both assets together, so the claim proceeds and advances both checkpoints, discarding the entitlement on the leg that paid nothing. Preconditions: the contributor must claim at a moment when one leg rounds to zero. Only the contributor can cause this. `claimRewardsFor` is onlyFactory, and the sole factory route is `batchClaimRewards`, which passes `msg.sender`. That function takes an array of token addresses and no user parameter, so no address can settle another. The scenario in which a contributor is settled repeatedly by another party is not reachable in this codebase. Bound on the loss: the discarded amount is the contributor's share of the window, which is below one base unit by definition, because that is why it rounded to zero. The loss is under 1 wei per claim at every contribution size, and the amount stays in the contract. Bound on the window: one leg rounds to zero only while that fee pool has grown by less than $\text{totalRaised} / \text{contribution base units}$ since the last claim. `MIN_CONTRIBUTION` of 0.0001 ETH against a `targetETHRaise` of 11 ETH means the smallest position anyone can hold is $1/110,000$ of the raise, so the widest possible window is a pool that has grown by under 110,000 base units, roughly $1.1\text{e-}13$ of the asset. The 0.75 ETH per wallet cap puts the largest holder's window under 15 base units.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	abstracts/MigrationManager.sol#L50,L162
Status	Unresolved

Description

The `selectInitTick` state variable is written once during SELECT pool initialization and is never read anywhere in the codebase. An unused variable costs a storage write at deployment and misleads readers into assuming the value participates in later logic.

```
int24 internal selectInitTick;  
...  
selectInitTick = extremeTick;
```

Recommendation

The team is advised to remove the unused variable, or to expose and consume it if it was intended to inform the retry-seeding path.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	TokenSelectToken.sol#L2 abstracts/PresaleCore.sol#L3 abstracts/MigrationManager.sol#L2 abstracts/RewardManager.sol#L2 interfaces/IWETH.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 ^0.8.26;  
...  
pragma solidity ^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.

Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
TokenSelectToken	Implementation	RewardManager		
		Public	✓	-
V3MathLib	Library			
	getSqrtRatioAtTick	External		-
	_getSqrtRatioAtTick	Internal		
	getTickAtSqrtRatio	External		-
	getTokenOrderAndAmounts	External		-
	calculateTickRange	External		-
	minUsableTick	Public		-
	maxUsableTick	Public		-
	parseAmountsUsed	External		-
	calculateAlignedTick	External		-
	_sqrt	Internal		
	_calculateInitialTick	Internal		
	_getTickAtSqrtRatio	Internal		
PresaleMath	Library			

	calculateUserShare	Internal		
	calculateTokenAllocation	Internal		
	calculateMigrationFee	Internal		
	calculateTokenPrice	Internal		
IWETH	Interface			
	deposit	External	Payable	-
	withdraw	External	✓	-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	balanceOf	External		-
IUniswapV3SwapCallback	Interface			
	uniswapV3SwapCallback	External	✓	-
IUniswapV3Pool	Interface			
	initialize	External	✓	-
	slot0	External		-
	swap	External	✓	-
IUniswapV3Factory	Interface			
	createPool	External	✓	-

	feeAmountTickSpacing	External		-
	getPool	External		-
ITokenSelectToken	Interface			
	seedInitialLiquidity	External	✓	-
	creator	External		-
	lpVault	External		-
	teamRecipient	External		-
	marketingRecipient	External		-
	communityRecipient	External		-
	getStreamRecipients	External		-
	teamAllocation	External		-
	marketingAllocation	External		-
	communityRewardsAllocation	External		-
	transferAllocations	External	✓	-
	areAllocationsReleased	External		-
	claimRewardsFor	External	✓	-
	isReadyForMigration	External		-
	getVestingSchedules	External		-
	migrate	External	✓	-
	selectPoolReserve	External		-
	selectTokenPool	External		-
	selectPoolSeeded	External		-

	ethAskPositionId	External		-
	retrySelectPoolSeeding	External	✓	-
	refundsEnabled	External		-
	enableRefunds	External	✓	-
ITokenSelectFactory	Interface			
	streamManager	External		-
	collectAndTransferFees	External	✓	-
	selectToken	External		-
	ethPoolRatioBps	External		-
	collectSelectPoolFees	External	✓	-
INonfungiblePositionManager	Interface			
	mint	External	Payable	-
	collect	External	✓	-
	positions	External		-
	factory	External		-
	WETH9	External		-
	transferFrom	External	✓	-
ILPVault	Interface			
	collect	External	✓	-
	FACTORY	External		-

	POSITION_MANAGER	External		-
RewardManager	Implementation	MigrationManager		
	claimRewards	External	✓	nonReentrant
	claimRewardsFor	External	✓	nonReentrant onlyFactory
	_collectAndUpdateFees	Internal	✓	
	_calculateInternalSplits	Internal		
	_claimCreatorRewards	Internal	✓	
	_claimReferrerRewards	Internal	✓	
	_claimContributorRewards	Internal	✓	
	getPendingRewards	External		-
PresaleCore	Implementation	ERC20, ITokenSelectToken , ReentrancyGuard		
		Public	✓	ERC20
	_initializeCore	Internal	✓	
	name	Public		-
	symbol	Public		-
	contribute	External	Payable	nonReentrant saleActive
	refund	External	✓	nonReentrant
	enableRefunds	External	✓	onlyFactory
	claimPresaleTokens	External	✓	nonReentrant
	transferAllocations	External	✓	onlyStreamManager

	cancelSale	External	✓	nonReentrant onlyCreator
	getTokenPrice	External		-
	getUserShareBasisPoints	External		-
	getUserTokenAllocation	External		-
	getContributors	External		-
	getContributorsBatch	External		-
	getContributorCount	External		-
	isReadyForMigration	External		-
	isContributionActive	Public		-
	getVestingSchedules	External		-
	getStreamRecipients	External		-
	getETHBalance	External		-
	transfer	Public	✓	-
	transferFrom	Public	✓	-
MigrationManager	Implementation	PresaleCore, IUniswapV3Swap Callback		
	_initializeMigration	Internal	✓	
	seedInitialLiquidity	External	✓	onlyFactory
	_createAndInitPoolStrict	Internal	✓	
	_initializeSelectTokenPool	Internal	✓	
	migrate	External	✓	onlyFactory
	_addWethBidLeg	Internal	✓	
	uniswapV3SwapCallback	External	✓	-

	_payPoolOwed	Private	✓	
	_mintOneSidedToken	Internal	✓	
	addSelectPoolLiquidity	External	✓	-
	retrySelectPoolSeeding	External	✓	-
	_seedSelectPool	Internal	✓	
	_ethAskRange	Internal		
	_ethPoolTokens	Internal		
	hasLiquidityPosition	External		-
	getV3Addresses	External		-
	getPoolInfo	External		-
		External	Payable	-

Summary

TokenSelectToken contract implements a presale and liquidity migration mechanism. 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 TAC 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.



A **TAC Security** Company

The Cyberscope team

cyberscope.io