Cyberscope

A TAC Security Company

Audit Report

Coinpool

January 2026

Repositories : https://github.com/0xCoinPool/sub-account-contracts ,

https://github.com/0xCoinPool/auto-coinpool-gateway-contracts

Commits : d7ee05a11f8a21a212b22bea9a057ee8802a97a2,

a913a1124ef8c1c7582cd1224e2e40f201bc67df

Audited by © cyberscope

Table of Contents

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	06 Jan 2026

Source Files

Filename	SHA256
DelegatedAccountFactory.sol	26b9390297bc7e6d766f002cbf0e627382619843c29b8dca74f19b8f0bc57f24
DelegatedAccount.sol	1f668879d7cc505b8ba2bae66d2486d14423cb11a87f3872ab072f17b328d570
AutoCoinPoolGateway.sol	3666aa5ea10f7bd044ce09a3835558b816c6ddd5f92370937a1e851eee08c5fe
interfaces/IWrapNative.sol	4c2116a32462962129803eb9a99ce5841d0dfc8b0ffb85712792a8fc52edcd8e
interfaces/IV3PoolState.sol	226c19e60a21a559aef962b708a5b6848da5c88a6ab0df846b36bcffd48fd996
interfaces/IV3Factory.sol	8cdc17d3c26843ac31113c9f9865104fbb4eb5e749050042940123e35e153a2e
interfaces/INonfungiblePositionManager.sol	73539d4018046ae75e9cde3231f7c9ec094f3d6416149f17a45e2900c4d5cacf
interfaces/IDelegatedAccountFactory.sol	3f92cdfde15d263ccfb392a974e4f429d912273711581ebee20fbb82f997d425

Overview

CoinPool implements a Uniswap V3 Position Management System designed for automated fee collection, USDC conversion, and position repositioning. The architecture enables whitelisted executors to perform automated operations on behalf of users (with sensitive operations requiring multi-signature authorization) via two distinct custody models:

1. **Custodial (DelegatedAccount):** The protocol holds user NFTs.
2. **Non-Custodial (AutoCoinPoolGateway):** Users retain ownership of their NFTs.

DelegatedAccountFactory

Role: Deployment Hub & Global Configuration Registry (Custodial Model)

`DelegatedAccountFactory` serves as the deployment hub. It deploys `DelegatedAccount` clones via `CREATE2` for deterministic addressing and maintains protocol-wide settings referenced by all instances.

Key Responsibilities

- **Clone Deployment:** Creates `DelegatedAccount` instances using the **minimal proxy pattern (EIP-1167)**, with salt derived from owner and operator addresses.
- **Executor Registry:** Maintains `EnumerableSets` of approved `collectFeeExecutors` and `removeLiquidityExecutors` that can trigger automated operations across all accounts.
- **Signer Management:** Tracks authorized signers and the threshold required for multi-signature validation on reposition operations.
- **Router Whitelist:** Maintains approved swap router addresses to prevent arbitrary external calls.
- **Global Pause:** Controls the pause state for all `DelegatedAccount` instances, enabling protocol-wide emergency stops.

Technical Highlights

- **Gas Efficiency:** Centralized configuration reduces gas costs for individual accounts while enabling protocol-wide updates.

- **Unified Indexing:** Events are emitted through the factory (via the `onlyDelegatedAccount` modifier) for unified indexing.
- **Protocol Funding:** Native gas fee collection on account creation funds protocol operations.

DelegatedAccount

Role: Per-User Custodial Wallet

`DelegatedAccount` is a custodial wallet that holds Uniswap V3 position NFTs and underlying assets. Users deposit funds, and the contract manages positions on their behalf. An **operator** (typically the protocol) shares control with the owner and receives a configurable percentage of fees.

Key Responsibilities

- **Asset Custody:** Holds `ERC20` tokens, native ETH, and Uniswap V3 NFTs on behalf of the owner.
- **Position Management:** Provides `mint`, `increaseLiquidity`, `decreaseLiquidity`, `collect`, and `burn` operations for Uniswap V3 positions.
- **Manual Operations:** The Owner or Operator can perform swaps, fee collection, USDC conversion, and repositioning without external authorization.
- **Automated Operations:** Whitelisted executors can trigger fee collection and USDC swaps. **Repositioning** requires threshold signatures from factory-registered signers.
- **Fee Distribution:** Automated operations split USDC rewards between the operator (based on `operatorRewardBP`) and the owner.

Technical Highlights

- **EIP-712 Signatures:** Uses typed signatures for reposition authorization with per-`tokenId` nonces.
- **Native Token Support:** Supports ETH swaps via WETH wrapping/unwrapping.
- **Inheritance:** Inherits pause state, executor lists, signer set, and router whitelist directly from the factory.

- **Granular Permissions:** Allowance-based system lets owners grant per-`tokenId` permissions to specific executors.

AutoCoinPoolGateway

Role: Non-Custodial Gateway

`AutoCoinPoolGateway` offers a non-custodial alternative where users retain ownership of their Uniswap V3 NFTs. The gateway acts as an authorized operator that executors can invoke to perform automated operations, ensuring fees and refunds are sent directly to the NFT owner.

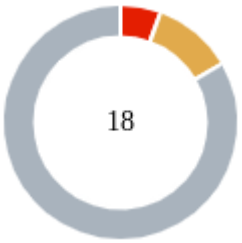
Key Responsibilities

- **Permission Management:** Users grant two-layer authorization:
  1. `ERC721` approval to the gateway on `NonfungiblePositionManager`.
  2. Internal allowance mappings for specific executors.
- **Fee Collection:** Executors can collect fees, swap to USDC, and transfer funds directly to the NFT owner.
- **Liquidity Reinvestment:** Supports automated fee collection with immediate reinvestment into the existing position.
- **Repositioning:** Closes out-of-range positions and opens new ones at updated tick ranges. It handles the automatic migration of executor allowances to the new `tokenId`.

Technical Highlights

- **No Fee Split:** 100% of converted USDC goes to the NFT owner.
- **Local Configuration:** Maintains its own configuration (signers, threshold, routers) rather than referencing an external factory.
- **Replay Protection:** Uses per-executor nonces, with the executor address included in the signed data.
- **ERC20 Only:** Operations are restricted to ERC20 tokens (no native ETH swap support).
- **Non-Custodial Minting:** New position NFTs are minted directly to the user's wallet.

Findings Breakdown



Critical	1
Medium	2
Minor / Informative	15

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	1	0	0
Medium	0	2	0	0
Minor / Informative	0	15	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	OCTD	Transfers Contract's Tokens	Acknowledged
●	UTIP	Unsigned Token ID Parameter	Acknowledged
●	UC	Unverified Calldata	Acknowledged
●	CCR	Contract Centralization Risk	Acknowledged
●	EAI	ERC20 Allowance Incompatibility	Acknowledged
●	MMN	Misleading Method Naming	Acknowledged
●	MPOV	Missing Position Ownership Validation	Acknowledged
●	NURG	Non Upgradeable Reentrancy Guard	Acknowledged
●	OLO	Out-of-Range Liquidity Operations	Acknowledged
●	PF	Pausable Functionality	Acknowledged
●	SPNR	Stale Position NFT Retention	Acknowledged
●	SLAV	Strict Liquidity Amount Validation	Acknowledged
●	UFR	Uncollected Fee Residual	Acknowledged
●	UMP	Unsanitized Mint Parameters	Acknowledged

●	UOP	Unvalidated Operator Parameter	Acknowledged
●	UTPD	Unverified Third Party Dependencies	Acknowledged
●	L14	Uninitialized Variables in Local Scope	Acknowledged
●	L16	Validate Variable Setters	Acknowledged

OCTD - Transfers Contract's Tokens

Criticality	Critical
Location	DelegatedAccount.sol#L253,268,325,457,498,628,675,771 AutoCoinPoolGateway.sol#L332,375,473
Status	Acknowledged

Description

The `mint`, `swap`, `increaseLiquidity`, `manualCollectFeeAndSwapUsdc`, `manualReposition`, `autoCollectFeeAndSwapUsdc`, `autoCollectFeeAndIncreaseLiquidity`, `autoReposition`, `collectFeeAndSwapUsdc`, `collectFeeAndIncreaseLiquidity`, `reposition` functions accept a `SwapOp` struct that includes arbitrary calldata for execution on a whitelisted router. Although the router address is validated, the calldata is not, allowing a malicious operator to specify an external recipient,such as their own address,in the swap parameters. Since the `_swap` function only checks for any increase in the contract's token balance without enforcing a minimum return, an operator could divert most tokens externally while returning a minimal amount, bypassing detection. It is worth noting, that constraints such as `swapInfo_[0].tokenOutAmount` & `swapInfo_[1].tokenOutAmount` may be ineffective as they are also externally provided.

```
Shell
struct SwapOp {
    address to;
    uint256 value;
    bytes data;

}
```

Recommendation

The team should implement strict validation of the calldata within the `SwapOp` struct to ensure that token transfers cannot be redirected to unauthorized addresses. In addition, implementing script slippage protection for the output amount can help prevent manipulation of trade outcomes. These measures will safeguard the swap mechanism from malicious exploitation, preserve the integrity of token flows, and protect user assets.

Team Update

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

To mitigate the issue, the contract includes several restrictions, such as allowing only system-approved Swap Routers to be used; restricting transaction calls to the owner or authorized operators/bots; and verifying the contract's token balances after a successful swap to ensure that the correct token type and required amount are received.

UTIP - Unsigned Token ID Parameter

Criticality	Medium
Location	DelegatedAccount.sol#L771 AutoCoinPoolGateway.sol#L473
Status	Acknowledged

Description

The `autoReposition` and `reposition` functions require multi-signature approval for repositioning operations, but the `tokenId_` parameter is not included in the signed data. The `RepositionSignData` struct only contains `swapInfo`, `swapOp`, `nonce`, and `deadline`. This means signers approve the swap parameters and execution details, but not which position the reposition applies to. A malicious executor could obtain valid signatures for a reposition operation and apply them to a different position than the signers intended, as long as that position passes the other validation checks.

```shell
RepositionSignData memory repositionSignData =
RepositionSignData({ swapInfo: swapInfo_, swapOp: swapOp_,
nonce: nonce_, deadline: deadline_ });
```

Recommendation

The team is advised to include the `tokenId_` parameter in the `RepositionSignData` struct to ensure that signers explicitly authorize the specific position being repositioned. Additionally, it is recommended to include `repositionData_`, which contains the tick range and slippage parameters, in the signed data. This prevents executors from altering these critical parameters after the signatures have been collected, thereby preserving the integrity of the signed intent.

Team Update

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

The purpose of that signature is only for validating the swaps. Therefore, if the swap params are the same, it should be valid.

UC - Unverified Calldata

Criticality	Medium
Location	DelegatedAccount.sol#L1113,1121 AutoCoinPoolGateway.sol#L799
Status	Acknowledged

Description

The `_customCall` and `_customCallWithValue` functions allow execution of arbitrary calldata on whitelisted router addresses without validating the calldata's contents. Although the target address is restricted to a list of approved routers, the operator or owner retains full discretion over the calldata passed. The operator can invoke any function with any parameters on the router contract. A malicious or compromised operator could exploit this by crafting calldata that routes swaps through illiquid pools, enables sandwich attacks, or triggers unintended functions on the router, potentially deviating from the intended behavior.

```shell
function _customCallWithValue(address target_, uint256
msgValue_, bytes calldata callData_) internal {
if (!delegatedAccountFactory.isSupportedRouter(target_)) {
revert NotSupportedRouter();
}
target_.functionCallWithValue(callData_, msgValue_);
}

function _customCall(address target_, bytes calldata
callData_) internal {
if (!delegatedAccountFactory.isSupportedRouter(target_)) {
revert NotSupportedRouter();
}
target_.functionCall(callData_);}
```

Recommendation

The team is advised to implement calldata validation by restricting which function selectors can be called on the whitelisted routers. Additionally, critical swap parameters such as minimum output amounts and recipient addresses could be extracted and validated from the calldata to ensure they match expected values, preventing the operator from specifying malicious routing paths or alternative recipients.

Team Update

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

In this contract, the calldata is treated as trusted input from the external Dex aggregator. We have also implemented pause and router-update functions to handle emergency situations

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	DelegatedAccount.sol#L168,180,195,211,233,245,253,268,325,384,425,448,457,498,628,675,771 AutoCoinPoolGateway.sol#L221,225,229,233,252,266,280,294,308,322,332,375 DelegatedAccountFactory.sol#L228,232,236,244,251,265,279,293,302
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.

```shell
function _onlyOwnerOrOperator() internal view {
        if (msg.sender != owner && msg.sender != operator) {
            revert UnauthorizedAccount();
        }

    }
```

Shell

```
_validateSignatures(_hashRepositionSignData(swapSignDa
ta), signatures_);
```

Shell

```
function updateSigners(address[] calldata signers_, bool
isTrue_) public onlyOwner {
        if (isTrue_) {
            for (uint256 i = 0; i < signers_.length; i++) {
                _signers.add(signers_[i]);
            }
        } else {
            for (uint256 i = 0; i < signers_.length; i++) {
                _signers.remove(signers_[i]);
            }
        }

        emit SignersUpdated(signers_, isTrue_);
    }

    function setThreshold(uint256 threshold_) public
onlyOwner {
        if (threshold_ == 0) {
            revert ZeroThreshold();
        }

        threshold = threshold_;

        emit ThresholdSet(threshold);}
```

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.

EAI - ERC20 Allowance Incompatibility

Criticality	Minor / Informative
Location	DelegatedAccount.sol#L1129 AutoCoinPoolGateway.sol810
Status	Acknowledged

Description

The `_refundUnusedToken` function utilizes `safeDecreaseAllowance` to revoke unused token approvals after operations such as minting or increasing liquidity. However, `safeDecreaseAllowance` is not part of the ERC20 standard and relies on the token implementing the `decreaseAllowance` function. Tokens that implement only the base ERC20 interface will cause transactions to fail.

```
Shell
IERC20(token_).safeDecreaseAllowance(spender_,
tokenAmountRefunded);
```

Recommendation

The team is advised to replace the safeDecreaseAllowance call with a universally compatible approach. This will ensure compatibility with tokens that only implement the base ERC20 interface without the allowance modification extensions.

MMN - Misleading Method Naming

Criticality	Minor / Informative
Location	DelegatedAccount.sol#L1129
Status	Acknowledged

Description

The naming of the `_refundUnusedToken` function may be misleading, as it suggests that unused tokens are actively refunded to a recipient. However, when `spender_` is set to `address(0)`, no actual token transfer occurs. Instead, the function performs a slippage check and calculates the difference between the actual tokens received from the swap and the minimum expected amount (`swapInfo_[i].tokenOutAmount`). This difference is recorded and emitted as a "refunded" amount in the event, even though the tokens remain in the contract and are not returned to any user.

```shell
for (uint8 i = 0; i < 2; i++) {
    (, uint256 tokenOutAmount) = _performSwap(swapInfo_[i],
swapOp_[i]);

    amount01SwapRefunded[i] =
        _refundUnusedToken(swapInfo_[i].tokenOut,
address(0), tokenOutAmount, swapInfo_[i].tokenOutAmount);

}
```

Recommendation

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

MPOV - Missing Position Ownership Validation

Criticality	Minor / Informative
Location	DelegatedAccount.sol#L284,425,448,457,498
Status	Acknowledged

Description

The `removeLiquidityAndCollectFee` , `collectFee` , `burn` , `manualCollectFeeAndSwapUsdc` , `manualReposition` functions do not explicitly verify that the contract owns the NFT position before attempting to remove liquidity and collect fees. In contrast to functions like `increaseLiquidity` , which include an ownership check using `IERC721.ownerOf` , these methods directly invokes `decreaseLiquidity` and `collect` on the `NonfungiblePositionManager` without prior validation. Absence of an explicit ownership check leads to less informative error messages and introduces inconsistency in validation practices across the contract.

```
Shell
function removeLiquidityAndCollectFee(uint256 tokenId_)
external nonReentrant {
...

}
```

Recommendation

The team is advised to add an explicit ownership check. This ensures that a clear and descriptive error message is returned when attempting to operate on a position not owned by the contract, and helps maintain consistent validation patterns throughout the codebase.

NURG - Non Upgradeable Reentrancy Guard

Criticality	Minor / Informative
Location	DelegatedAccount.sol#L18
Status	Acknowledged

Description

The `DelegatedAccount.sol` contract is designed to be upgradeable but inherits from OpenZeppelin's `ReentrancyGuard`, which is not intended for use in upgradeable contracts. This mismatch can lead to storage layout conflicts during upgrades. If an upgrade is performed, the contract's storage structure may become misaligned, potentially disabling the reentrancy protection and exposing the contract to reentrancy attacks.

```shell
import { ReentrancyGuard } from
"@openzeppelin/contracts/utils/ReentrancyGuard.sol";
```

Recommendation

The contract should replace the `ReentrancyGuard` import with `ReentrancyGuardUpgradeable` from OpenZeppelin's upgradeable contracts library. This ensures compatibility with the upgradeable contract pattern and preserves the correct storage layout, maintaining effective reentrancy protection during and after upgrades.

Team Update

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

We used the OZ contracts (v5.5.0) in the Delegated Account source code. In this version, the OZ team has changed the ReentrancyGuard to stateless and no longer need to use `ReentrancyGuardUpgradeable` . Also, the Delegated Account is non-upgradeable, the upgradeable libraries from OZ are only used for initializing purposes on contracts created by the factory.

OLO - Out-of-Range Liquidity Operations

Criticality	Minor / Informative
Location	DelegatedAccount.sol#L268,328,675 AutoCoinPoolGateway.sol#L375
Status	Acknowledged

Description

The `mint`, `increaseLiquidity`, `autoCollectFeeAndIncreaseLiquidity`, `collectFeeAndIncreaseLiquidity` functions do not consider whether the position's tick range includes the current pool price. In Uniswap V3, if a position is out of range, only one of the two tokens is required when adding liquidity—the other token will not be used.

Both functions currently perform unconditional swaps from ETH to token0 and token1 before adding liquidity, without verifying whether both tokens are necessary. In the `mint` function, if the user specifies a tick range that excludes the current tick, one of the swapped tokens will remain unused. In the `increaseLiquidity` function, the tick range is predetermined by the existing position. If the pool price has moved outside this range since the position was created, one of the swaps becomes redundant.

In both scenarios, the unused tokens remain locked in the contract, leading to unnecessary gas consumption and reduced capital efficiency for the user.

```shell
uint256[2] memory amount01SwapRefunded;
for (uint8 i = 0; i < 2; i++) {
(, uint256 tokenOutAmount) = _performSwap(swapInfo_[i],
swapOp_[i]);
amount01SwapRefunded[i] =
_refundUnusedToken(swapInfo_[i].tokenOut, address(0),
tokenOutAmount, swapInfo_[i].tokenOutAmount);}
```

Recommendation

The team is advised to query the current pool tick and compare it with the position's tick range before executing swaps. For the `increaseLiquidity` function, if the position is out of range, the function should either revert with a clear error message or proceed with only the swap corresponding to the token that will be deposited. For the `mint` function, the system could allow the caller to specify a single swap when intentionally setting up a range order. This behavior can be implemented by retrieving `tickLower` and `tickUpper` from the position parameters or `MintData`, querying the current tick from the pool's `slot0`, and conditionally skipping any unnecessary swap when the position is determined to be out of range.

Team Update

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

We agree that the position can be out-of-range but we allow the users to do so. If the position is out-of-range and the user insists on swapping the unused token, the swap still be successful but the Uniswap protocol will refund the unused token to the contract, then the contract will refund that token to the user, not locking the token in the contract.

PF - Pausable Functionality

Criticality	Minor / Informative
Location	DelegatedAccount.sol#L945 DelegatedAccountFactory#L228
Status	Acknowledged

Description

The factory admin is able to pause the contract. The result will be that the owner will not be able to access their accounts.

```
Shell
function _whenNotPaused() internal view {
        if (delegatedAccountFactory.paused()) {
            revert EnforcedPause();
        }

    }
```

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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.

- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

SPNR - Stale Position NFT Retention

Criticality	Minor / Informative
Location	DelegatedAccount.sol#L498,771 AutoCoinPoolGateway.sol#473
Status	Acknowledged

Description

The `manualReposition`, `autoReposition` and `reposition` functions remove all liquidity from an existing position, collect any accrued fees, and creates a new position within a different tick range. However, after this process, the original position NFT (`tokenId_`) is not burned. As a result, the old NFT remains in existence with zero liquidity and no uncollected fees.

This can clutter the contract's state, complicate position management, and potentially confuse off-chain systems or user interfaces that rely on enumerating active positions.

```shell
INonfungiblePositionManager.DecreaseLiquidityParams memory
removeLiquidityParams = INonfungiblePositionManager
.DecreaseLiquidityParams({
tokenId: tokenId_,
liquidity: liquidity,
amount0Min: 0,
amount1Min: 0,
deadline: block.timestamp
});
(uint256 amount0LiquidityRemoved, uint256
amount1LiquidityRemoved) =

nonfungiblePositionManager.decreaseLiquidity(removeLiquidit
yParams);
```

Recommendation

The team can consider burning the stale position NFT after successfully minting the new position. This can be done by calling nonfungiblePositionManager.burn(tokenId_) at the end of the function.

SLAV - Strict Liquidity Amount Validation

Criticality	Minor / Informative
Location	DelegatedAccount.sol#L498,771 AutoCoinPoolGateway.sol#473
Status	Acknowledged

Description

The `manualReposition`, `autoReposition` and `reposition` functions enforce a strict equality check between the amounts returned from `decreaseLiquidity` and the expected values specified in `swapInfo_[0].tokenInAmount` and `swapInfo_[1].tokenInAmount`. This requirement poses a challenge because it assumes the operator can precisely predict the liquidity amounts at the time of transaction submission. However, due to potential changes in the pool state caused by other transactions, slight variations in the returned amounts may occur between submission and execution. Even minimal discrepancies will cause the transaction to revert, making the repositioning process fragile and susceptible to failure, especially during periods of high pool activity.

```shell
if (
amount0LiquidityRemoved !=
swapInfo_[0].tokenInAmount
|| amount1LiquidityRemoved !=
swapInfo_[1].tokenInAmount
) {
revert InvalidAmountLiquidityRemoved();

}
```

```shell
if (
amount0LiquidityRemoved !=
swapInfo_[0].tokenInAmount
|| amount1LiquidityRemoved !=
swapInfo_[1].tokenInAmount
) {
revert InvalidAmountLiquidityRemoved();

}
```

Recommendation

The team is advised to replace strict equality checks with minimum threshold validations using the greater-than-or-equal-to operator, aligning with the approach used in other parts of the contract. Additionally, the swap calldata should be designed to accommodate the actual amounts removed, rather than relying on exact value predictions. This ensures greater flexibility and consistency in execution.

UFR - Uncollected Fee Residual

Criticality	Minor / Informative
Location	DelegatedAccount.sol#L457,498,628,675,771
Status	Acknowledged

Description

The `manualCollectFeeAndSwapUsdc` , `manualReposition` , `autoCollectFeeAndIncreaseLiquidity` , `autoCollectFeeAndSwapUsdc` and `autoReposition` functions are designed to collect fees from a Uniswap V3 position and perform token swaps. However, they do not utilize the actual amounts collected during execution. Instead, they rely on predefined values provided through the `swapInfo_[i].tokenInAmount` parameters to determine the swap amounts.

Although the functions check that the collected fees meet or exceed the specified input amounts (e.g., `amount0FeeCollected >= swapInfo_[0].tokenInAmount`), any surplus beyond these values remains in the contract and is not swapped. This design requires the operator to accurately estimate the fee amounts in advance. If the estimate is too high, the excess tokens are left unused in the contract.

```shell
uint256 usdcSwapped;
for (uint8 i = 0; i < 2; i++) {
(, uint256 tokenOutAmount) =
_performSwap(swapInfo_[i], swapOp_[i]);
usdcSwapped += tokenOutAmount;

}
```

Recommendation

The team is advised to use the actual collected fee amounts as inputs for the swap operations rather than relying on externally specified parameters. This ensures that all collected fees are accurately converted to USDC and prevents any tokens from remaining unprocessed within the contract.

UMP - Unsanitized Mint Parameters

Criticality	Minor / Informative
Location	DelegatedAccount.sol#L268,498,771 AutoCoinPoolGateway.sol#L473
Status	Acknowledged

Description

The `mint` function and the `manualReposition`, `autoReposition`, `reposition` methods accept several user-supplied parameters without performing necessary validation checks, which may lead to unintended behavior or failed transactions. Specifically:

- The tick range parameters are not verified to ensure that `tickLower` is less than `tickUpper`, or that both values are valid multiples of the pool's tick spacing.
- The fee tier is not validated against Uniswap V3's supported values: 100, 500, 3000, or 10000.
- The token addresses provided via `swapInfo_[0].tokenOut` and `swapInfo_[1].tokenOut` are not checked to confirm they are sorted in ascending order, as required by Uniswap V3 conventions (`token0 < token1`).
- The `amount0Min` and `amount1Min` values are not verified to be less than or equal to their respective desired amounts.
- The specific pool configuration exists and is initialized with sufficient liquidity.

Passing these unchecked parameters directly to the Uniswap V3 `NonfungiblePositionManager` can result in the creation of positions in incorrect pools, transaction failures, or other unintended outcomes.

```shell
INonfungiblePositionManager.MintParams memory mintParams =
INonfungiblePositionManager.MintParams({
    token0: swapInfo_[0].tokenOut,
    token1: swapInfo_[1].tokenOut,
    fee: mintData_.fee,
    tickLower: mintData_.tickLower,
    tickUpper: mintData_.tickUpper,
    amount0Desired: swapInfo_[0].tokenOutAmount,
    amount1Desired: swapInfo_[1].tokenOutAmount,
    amount0Min: mintData_.amount0Min,
    amount1Min: mintData_.amount1Min,
    recipient: address(this),
    deadline: block.timestamp

});
```

Recommendation

The team is advised to implement comprehensive parameter validation before executing the mint operation. This includes:

- Ensuring that `tickLower` is less than `tickUpper`.
- Verifying that both ticks are valid multiples of the pool's tick spacing for the specified fee tier.
- Confirming that the fee tier matches one of Uniswap V3's supported values.
- Checking that the output tokens are correctly ordered, with `swapInfo_[0].tokenOut` less than `swapInfo_[1].tokenOut`.
- Validating that minimum amounts do not exceed the desired amounts.
- Ensuring the pool configuration is initialized.

Implementing these checks at the contract level provides clear error messages and prevents wasted gas on transactions that would otherwise fail within the Uniswap contracts.

UOP - Unvalidated Operator Parameter

Criticality	Minor / Informative
Location	DelegatedAccountFactory.sol#L323
Status	Acknowledged

Description

In the `DelegatedAccountFactory.createDelegatedAccount` function, the `operator_` parameter is not validated against the factory's approved executor lists (`_collectFeeExecutors` , `_removeLiquidityExecutors`). Although executors must be whitelisted to perform automated operations, the operator who is granted privileged access through the `_onlyOwnerOrOperator()` modifier and receives a share of USDC rewards via `operatorRewardBP` can be any arbitrary address.

```
Shell
address delegatedAccount =

Clones.cloneDeterministic(delegatedAccountImplementation,
keccak256(abi.encode(msg.sender, operator_)));
```

Recommendation

Consider validating the `operator_` parameter against a factory-maintained whitelist of approved operators. This ensures that only trusted entities can be assigned as operators, reducing the risk of misuse.

UTPD - Unverified Third Party Dependencies

Criticality	Minor / Informative
Location	DelegatedAccount.sol#L1113,1121 AutoCoinPoolGateway.sol#L799
Status	Acknowledged

Description

The contract uses an external contract in order to determine the transaction's flow. The external contract is untrusted. As a result, it may produce security issues and harm the transactions.

```shell
function _customCallWithValue(address target_, uint256 msgValue_, bytes calldata callData_) internal {
if (!delegatedAccountFactory.isSupportedRouter(target_)) {
revert NotSupportedRouter();
}
target_.functionCallWithValue(callData_, msgValue_);
}

function _customCall(address target_, bytes calldata callData_) internal {
if (!delegatedAccountFactory.isSupportedRouter(target_)) {
revert NotSupportedRouter();
}
target_.functionCall(callData_);

}
```

Recommendation

The contract should use a trusted external source. A trusted source could be either a commonly recognized or an audited contract. The pointing addresses should not be able to change after the initialization.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	DelegatedAccount.sol#L283,349,582,732,871,1162,1163,1190 AutoCoinPoolGateway.sol#L431,562,563,585,586,846,847,877
Status	Acknowledged

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
Shell
uint256[2] memory amount01SwapRefunded
bytes32[4] memory swapInfoHashes
bytes32[4] memory swapOpHashes
uint256 validSignatures
uint256 amount0FeeCollected
uint256 amount1FeeCollected
uint256 amount0SwapRefunded

uint256 amount1SwapRefunded
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	DelegatedAccountFactory.sol#L181,184,190,191,192,193,246 DelegatedAccount.sol#L150,151
Status	Acknowledged

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.

```shell
delegatedAccountImplementation =
delegatedAccountImplementation_
nativeGasRecipient = nativeGasRecipient_
nonfungiblePositionManager =
nonfungiblePositionManager_
v3Factory = v3Factory_
wrapNative = wrapNative_
usdc = usdc_
owner = owner_

operator = operator_
```

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.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
DelegatedAccountFactory	Implementation	Ownable, Pausable		
		Public	✓	Ownable
	getCollectFeeExecutors	External		-
	isCollectFeeExecutor	External		-
	getRemoveLiquidityExecutors	External		-
	isRemoveLiquidityExecutor	External		-
	getSigners	External		-
	isSigner	External		-
	getSupportedRouters	External		-
	isSupportedRouter	External		-
	pause	External	✓	onlyOwner
	unpause	External	✓	onlyOwner
	rescue	External	✓	onlyOwner
	setNativeGas	External	✓	onlyOwner
	updateCollectFeeExecutors	External	✓	onlyOwner
	updateRemoveLiquidityExecutors	External	✓	onlyOwner
	updateSigners	External	✓	onlyOwner
	setThreshold	External	✓	onlyOwner
	updateSupportedRouters	External	✓	onlyOwner

	createDelegatedAccount	External	Payable	whenNotPaused
	emitCollectFeeExecutorSet	External	✓	onlyDelegatedAccount
	emitRemoveLiquidityExecutorSet	External	✓	onlyDelegatedAccount
	emitCollectFeeAllowancesUpdated	External	✓	onlyDelegatedAccount
	emitRemoveLiquidityAllowancesUpdated	External	✓	onlyDelegatedAccount
	emitDeposited	External	✓	onlyDelegatedAccount
	emitTokenWithdrawn	External	✓	onlyDelegatedAccount
	emitNFTWithdrawn	External	✓	onlyDelegatedAccount
	emitSwapped	External	✓	onlyDelegatedAccount
	emitMinted	External	✓	onlyDelegatedAccount
	emitLiquidityIncreased	External	✓	onlyDelegatedAccount
	emitLiquidityRemovedAndFeeCollected	External	✓	onlyDelegatedAccount
	emitFeeCollected	External	✓	onlyDelegatedAccount
	emitBurned	External	✓	onlyDelegatedAccount
	emitManualFeeCollectedAndUsdcSwapped	External	✓	onlyDelegatedAccount
	emitManualRepositioned	External	✓	onlyDelegatedAccount
	emitAutoFeeCollectedAndUsdcSwapped	External	✓	onlyDelegatedAccount
	emitAutoFeeCollectedAndLiquidityIncreased	External	✓	onlyDelegatedAccount
	emitAutoRepositioned	External	✓	onlyDelegatedAccount

DelegatedAccount	Implementation	Initializable, ReentrancyGuard, EIP712Upgradeable		
		Public	✓	-
		External	Payable	-
	initialize	External	✓	initializer
	getCollectFeeAllowances	External		-
	getRemoveLiquidityAllowances	External		-
	setCollectFeeExecutor	External	✓	-
	setRemoveLiquidityExecutor	External	✓	-
	setCollectFeeAllowance	External	✓	-
	setRemoveLiquidityAllowance	External	✓	-
	deposit	External	Payable	-
	withdrawToken	External	✓	-
	withdrawNFT	External	✓	-
	swap	External	✓	nonReentrant
	mint	External	✓	nonReentrant
	increaseLiquidity	External	✓	nonReentrant
	removeLiquidityAndCollectFee	External	✓	nonReentrant
	collectFee	External	✓	nonReentrant
	burn	External	✓	nonReentrant
	manualCollectFeeAndSwapUsdc	External	✓	nonReentrant
	manualReposition	External	✓	nonReentrant
	autoCollectFeeAndSwapUsdc	External	✓	nonReentrant

	autoCollectFeeAndIncreaseLiquidity	External	✓	nonReentrant
	autoReposition	External	✓	nonReentrant
	_whenNotPaused	Internal		
	_onlyOwner	Internal		
	_onlyOwnerOrOperator	Internal		
	_onlyCollectFeeExecutor	Internal		
	_onlyRemoveLiquidityExecutor	Internal		
	_performSwap	Internal	✓	
	_performMint	Internal	✓	
	_performIncreaseLiquidity	Internal	✓	
	_swapExactTokenIn	Internal	✓	
	_swap	Internal	✓	
	_customCallWithValue	Internal	✓	
	_customCall	Internal	✓	
	_refundUnusedToken	Internal	✓	
	_hashSwapInfo	Internal		
	_hashSwapOp	Internal		
	_hashRepositionSignData	Internal		
	_validateSignatures	Internal		
AutoCoinPoolGateway	Implementation	EIP712, Ownable, Pausable, ReentrancyGuard		
		Public	✓	EIP712 Ownable
	getSupportedRouters	Public		-

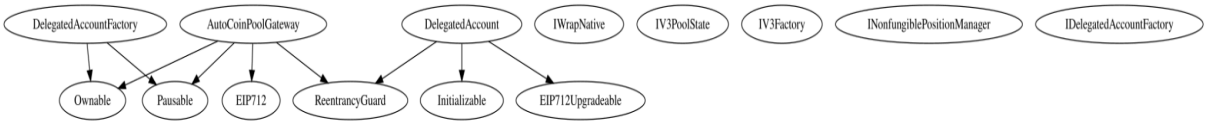
	getCollectFeeExecutors	Public		-
	getRemoveLiquidityExecutors	Public		-
	getSigners	Public		-
	pause	Public	✓	onlyOwner
	unpause	Public	✓	onlyOwner
	rescue	Public	✓	onlyOwner
	updateSupportedRouters	Public	✓	onlyOwner
	updateCollectFeeExecutors	Public	✓	onlyOwner
	updateRemoveLiquidityExecutors	Public	✓	onlyOwner
	setCollectFeeAllowance	Public	✓	-
	setRemoveLiquidityAllowance	Public	✓	-
	updateSigners	Public	✓	onlyOwner
	setThreshold	Public	✓	onlyOwner
	collectFeeAndSwapUsdc	Public	✓	whenNotPaused onlyCollectFeeExecutor nonReentrant
	collectFeeAndIncreaseLiquidity	Public	✓	whenNotPaused onlyCollectFeeExecutor nonReentrant
	reposition	Public	✓	whenNotPaused onlyRemoveLiquidityExecutor nonReentrant
	_performSwap	Internal	✓	
	_performMint	Internal	✓	
	_performIncreaseLiquidity	Internal	✓	
	_swapExactTokenIn	Internal	✓	

	_swap	Internal	✓	
	_customCall	Internal	✓	
	_refundUnusedToken	Internal	✓	
	_hashSwapInfo	Internal		
	_hashSwapOp	Internal		
	_hashRepositionSignData	Internal		
	_validateSignatures	Internal		
IWrapNative	Interface			
	deposit	External	Payable	-
	withdraw	External	✓	-
	totalSupply	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
IV3PoolState	Interface			
	slot0	External		-
IV3Factory	Interface			
	getPool	External		-
INonfungiblePositionManager	Interface			
	positions	External		-

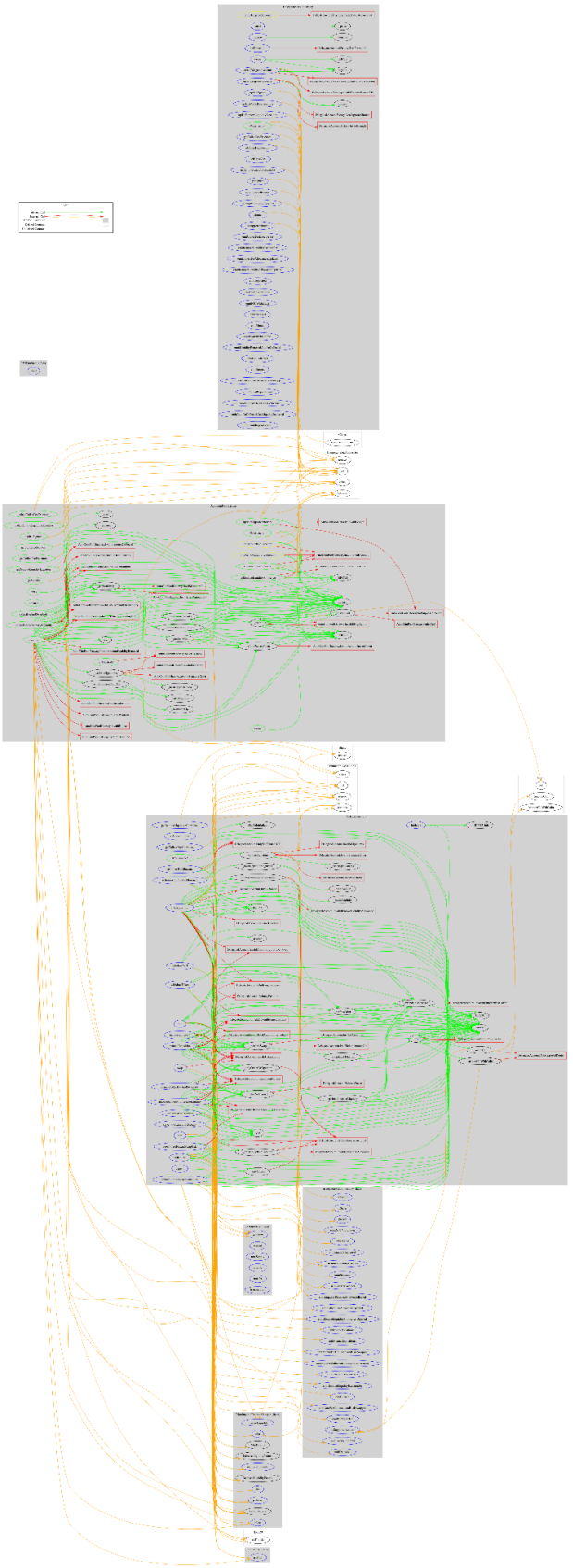
	mint	External	Payable	-
	increaseLiquidity	External	Payable	-
	decreaseLiquidity	External	Payable	-
	collect	External	Payable	-
	burn	External	Payable	-
IDelegatedAccountFactory	Interface			
	paused	External		-
	isCollectFeeExecutor	External		-
	isRemoveLiquidityExecutor	External		-
	isSigner	External		-
	threshold	External		-
	isSupportedRouter	External		-
	emitCollectFeeExecutorSet	External	✓	-
	emitRemoveLiquidityExecutorSet	External	✓	-
	emitCollectFeeAllowancesUpdated	External	✓	-
	emitRemoveLiquidityAllowancesUpdated	External	✓	-
	emitDeposited	External	✓	-
	emitTokenWithdrawn	External	✓	-
	emitNFTWithdrawn	External	✓	-
	emitSwapped	External	✓	-
	emitMinted	External	✓	-
	emitLiquidityIncreased	External	✓	-
	emitLiquidityRemovedAndFeeCollected	External	✓	-

	emitFeeCollected	External	✓	-
	emitBurned	External	✓	-
	emitManualFeeCollectedAndUsdcSwapped	External	✓	-
	emitManualRepositioned	External	✓	-
	emitAutoFeeCollectedAndUsdcSwapped	External	✓	-
	emitAutoFeeCollectedAndLiquidityIncreased	External	✓	-
	emitAutoRepositioned	External	✓	-

Inheritance Graph



Flow Graph



Summary

CoinPool contract implements a utility 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