HYDN

Smart Contract Security Assessment

July 12, 2023

Project Name:

Revert Finance: Auto-Exit, Auto-Move Range

Prepared by:

HYDN

Executive Summary

Client Name: Revert Finance: Auto-Exit, Auto-Move Range

Ecosystem: Ethereum

Language: Solidity

Timeline: Delivered 12th July 2023

Repository: https://github.com/revert-finance/v3utils

Method: Static Automated Analysis + Manual Review

Vulnerability Summary

6	5	1	0
Total Findings	**Resolved**	**Acknowledged**	**Declined**

Contents

Introduction

About HYDN

HYDN is an industry leader in blockchain security and smart contract audits. Founded by Warren Mercer, a world renowned cybersecurity and blockchain expert, who has previously held senior roles at NYSE, Cisco, and Alert Logic. Having been involved in cryptocurrency for over 10 years, Warren is dedicated to making the blockchain ecosystem as secure as it can be for everyone. Warren serves as the CEO for HYDN and heads up the delivery team to ensure that work is carried out to the highest standard.

The HYDN delivery team has over 10 years combined experience in blockchain, having performed smart contract audits and built security systems for a large range of protocols and companies. HYDN have performed smart contract security services for the likes of SushiSwap, Sablier, SpookySwap, Swapsicle, Nau Finance, CrossWallet, Dancing Seahorse, Octane, Position Exchange, Looter, and more.

HYDN worked closely with Revert Finance to consider their unique business needs, objectives, and concerns. Our mission is to ensure the blockchain supports secure business operations for all and he has built the team at HYDN from the ground up to meet this very personal objective.

To keep up to date with our latest news and announcements, check out our website https://hydnsec.com/ or follow @hydnsecurity on Twitter.

About Revert Finance

Revert develops analytics and management tools for liquidity providers in AMM protocols.

Revert believes AMMs are going to become a fundamental part of financial markets in the coming years, this will create new investment opportunities for retail investors but will also require open, transparent, and accessible tools for everyone.

Methodology

When tasked with conducting a security assessment, HYDN works through multiple phases of security auditing to ensure smart contracts are audited thoroughly. To begin the process automated tests are carried out, before HYDN then moves onto carrying out a detailed manual review of the smart contracts.

HYDN uses a variety of open-source tools and analyzers as and when they are required and alongside this, HYDN primarily focuses on the following classes of security and reliability issues:

Basic Coding Mistakes

One of the most common causes of critical vulnerabilities is basic coding mistakes. Countless projects have experienced hacks and exploits due to simple, surface level mistakes that could have been flagged and addressed by a simple code review. The HYDN automated audit process which includes model checkers, fuzzers, and theorem provers analyses the smart contract for many of these basic coding mistakes. Once the automated audit has taken place, HYDN then performs a manual review of the code to gain familiarity with the contracts.

Business Logic Errors

HYDN reviews the platform or projects design documents before analysing the code to ensure that the team has a deep understanding of the business logic and goals. Following this, HYDN reviews the smart contracts to ensure that the contract logic is in line with the expected functionality.

HYDN also analyses the code for inconsistencies, flaws, or vulnerabilities which could impact business logic such as Tokenomics errors, arbitrage opportunities, and share pricing.

Complex Integration Risks

Several high-profile exploits have been the result of not any bug within the contract itself, but rather an unintended consequence of its interaction with the broader DeFi ecosystem.
We perform a meticulous review of all of the contract's possible external interactions, and summarise the associated risks; for example: flash loan attacks, oracle price manipulation, MEV/sandwich attacks, etc.

Code Maturity

HYDN reviews the smart contracts for potential improvements in the codebase. These improvements ensure that the code follows industry best practices and guidelines, or code quality standards. Alongside this, HYDN makes suggestions for code optimization items such as gas optimization, upgradeability weaknesses, centralization risks, and more.

Impact Rankings

Once HYDN has completed the security assessment, each issue is assigned an impact rating. This impact rating is based upon both the severity and likelihood of the issue occurring. Each security assessment is different and the business needs and goals, such as project timelines, and Revert Finance threat modelling. are taken into account when the impact rankings are created.

HYDN assigns the following impact rankings: Critical, High, Medium, Low (listed by severity).

Disclaimer

This HYDN security assessment does not provide any warranties on finding all possible issues within the scope and the evaluation results do not guarantee the absence of any issues. **HYDN cannot make any guarantees on any further code which is added or altered after the security assessment has taken place**. As a single security assessment can never be considered fully comprehensive, HYDN always recommends multiple independent assessments paired with a bug bounty program. This security assessment report should not be considered as financial or investment advice.

Summary of Findings

Executive Summary

As part of this audit, HYDN was tasked with performing a Smart Contract Audit on the Revert Finance v3utils smart contracts. HYDN found one High vulnerability issue which can result in loss of assets. Alongside this, HYDN found several further vulnerabilities ranging from Medium to Low impact.

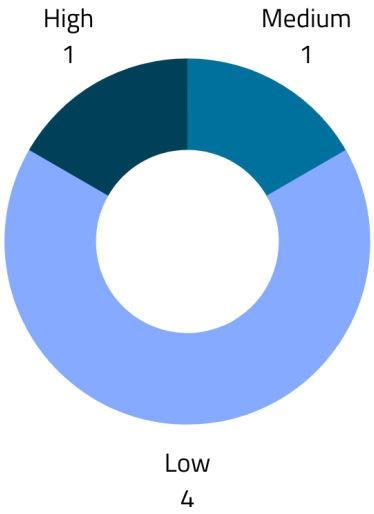
Project Summary

Platform	Ethereum
Language	Solidity
Repository	https://github.com/revert-finance/v3utils
Scope	https://github.com/revert-finance/v3utils/tree/8e50ae9b1c9145857ac7fc763118d20ccb84ee19/src/automators
Initial Commits Hash	8e50ae9b1c9145857ac7fc763118d20ccb84ee19
Latest Commit Hash	7faa9b9abbe809c9b4fc56787d76ec859e69a887

Audit Summary

Delivery Date	July 06 2023

Breakdown of Findings



ID	Title	Severity	Status
1	Swap Parameters and Position State Inconsistency Can Lead To Loss of Assets	High	Resolved
2	Exit Position Exposes Rate Manipulation	Medium	Resolved
3	Static TWAPS Configuration May Cause Issues	Low	Acknowledged But No Action Required
4	Optimize Overkill Check To Reduce Gas	Low	Resolved
5	Invalid Memory State May Lead To Unexpected Situations and Extra Gas	Low	Resolved
6	Native Transfer May Lead To Loss of Funds	Low	Resolved

Detailed Findings

1 Swap Parameters and Position State Inconsistency Can Lead To Loss of Assets [High]

1.1 Description

AutoRange executes functions sequentially, executing multiple states including: closing a position, performing a swap, and creating a new position. The swap parameters are calculated off-chain due to the simplicity of front-running attacks that can exploit market movements. However, this introduces a vulnerability where the expected withdrawal amount may change, even though the swap amount is statically defined by parameters. This creates an opportunity for an attacker to economically benefit in the scenario below:

1. Revert operator broadcasts transactions to AutoRange in order to adjust the position.
2. Upon detecting such a transaction, the attacker reduces the liquidity size of the position or even reduces it to zero.
3. The AutoRange ***execute()*** function is then triggered, resulting in withdrawal amounts that are lower than expected. However, the pre-defined swap is still executed as initially planned in step 1. This execution consumes contract funds, including the Revert Finance fees waiting for withdrawal.

```
120         if (state.currentTick < state.tickLower - config.lowerTickLimit || state.currentTick >= state.tickUpper + config.upperTickLimit) {
121
122             int24 tickSpacing = _getTickSpacing(state.fee);
123             int24 baseTick = state.currentTick - (((state.currentTick % tickSpacing) + tickSpacing) % tickSpacing);
124
125             // check if new range same as old range
126             if (baseTick + config.lowerTickDelta == state.tickLower && baseTick + config.upperTickDelta == state.tickUpper) {
127                 revert SameRange();
128             }
129
130             (state.amountInDelta, state.amountOutDelta) = _swap(
131                 params.swap0To1 ? IERC20(state.token0) : IERC20(state.token1),
132                 params.swap0To1 ? IERC20(state.token1) : IERC20(state.token0),
133                 params.amountIn,
134                 state.amountOutMin,
135                 params.swapData
136             );
137
138             state.amount0 = params.swap0To1 ? state.amount0 - state.amountInDelta : state.amount0 + state.amountOutDelta;
139             state.amount1 = params.swap0To1 ? state.amount1 + state.amountOutDelta : state.amount1 - state.amountInDelta;
140
```

Extract of AutoRange.sol

1.2 Impact

The attacker will get a new position initiated with Revert Finance fees, he can then withdraw his position. This would result in net profit for the attacker, draining Revert fees.

1.3 Recommendations

To mitigate potential risks, it is important to ensure that the value of ***params.amountIn*** is lower or equal to the sum of the token withdraw and the token fee collected.

Furthermore, it is advisable to enforce that the ***state.liquidity*** value exactly matches the off-chain Revert calculation value. This precautionary measure prevents attackers from reducing their position between the on-chain Revert computation and the actual execution, ensuring a more secure process.

1.4 Remediation

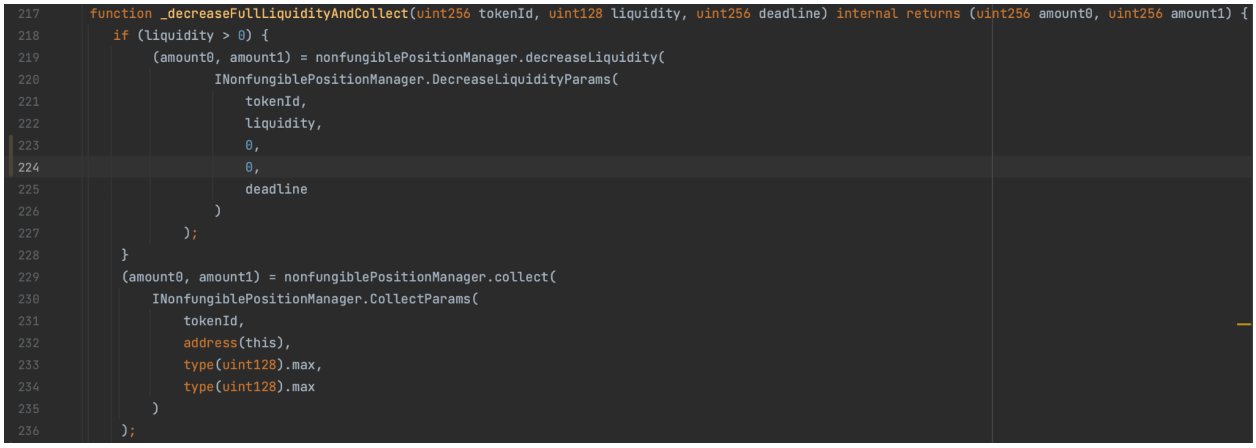
Assertions to control ***params.amountIn*** and ***state.liquidity*** have been added by the Revert Finance team.

2 Exit Position Exposes Rate Manipulation [Medium]

2.1 Description

During the period between transaction broadcast and execution, the chain state may change. It is important to exercise caution when dealing with swaps and position opening or closing with boundaries.

It is worth noting that the function ***_decreaseFullLiquidityAndCollect*** does not impose any minimum token1 and token2 withdrawal requirements from the position.



```
217     function _decreaseFullLiquidityAndCollect(uint256 tokenId, uint128 liquidity, uint256 deadline) internal returns (uint256 amount0, uint256 amount1) {
218         if (liquidity > 0) {
219             (amount0, amount1) = nonfungiblePositionManager.decreaseLiquidity(
220                     INonfungiblePositionManager.DecreaseLiquidityParams(
221                         tokenId,
222                         liquidity,
223                         0,
224                         0,
225                         deadline
226                     )
227                 );
228         }
229         (amount0, amount1) = nonfungiblePositionManager.collect(
230             INonfungiblePositionManager.CollectParams(
231                 tokenId,
232                 address(this),
233                 type(uint128).max,
234                 type(uint128).max
235             )
236         );
```

Extract of Automater.sol

2.2 Impact

Unlike swaps, which are limited by TWAPS config, closing a position does not have a defined boundary. This can be exploited by front-runners or executed in an unexpected state due to market movements between transaction broadcast and execution.

For instance, in the case of a 100% swap limit order, if the market comes back and the position withdraws, it may result in 90% of token1 and 10% of token2, which deviates from the expected outcome.

2.3 Recommendations

Define the ***minAmount1*** and ***minAmount2*** from off-chain calculated value parameters with a certain slippage tolerance to ensure execution is done as expected or at least as close to it, so that it avoids extreme manipulation.

2.4 Remediation

MinAmount1 and minAmount2 are calculated off-chain and pass as parameters for safe on-chain execution.

3 Static TWAPS Configuration May Cause Issues [Low]

3.1 Description

TWAPS is used in both ***AutoRange*** and ***AutoExit*** to prevent rate manipulation during swap execution. It defines the acceptable number of tick divergences (slippage) and the duration of the lookback.

It is important to note that different assets, such as ETH and DAI, exhibit distinct market dynamics. Additionally, different pools, such as STETH/ETH and ETH/DAI, have different correlations. Consequently, the expected price variation differs significantly based on these factors.

For assets with high correlation, like STEH/ETH or DAI/USDC, the anticipated price deviation is extremely low. On the other hand, pairs like ETH/DAI can experience rapid price fluctuations.

Liquidity also plays a significant role, particularly for small-cap and highly volatile assets, as it can lead to substantial price variations.

```
94      function setTWAPConfig(uint16 _maxTWAPTickDifference, uint32 _TWAPSeconds) external onlyOwner {
95          if (_TWAPSeconds < TWAPSeconds) {
96              revert InvalidConfig();
97          }
98          if (_maxTWAPTickDifference > maxTWAPTickDifference) {
99              revert InvalidConfig();
100         }
101         emit TWAPConfigChanged(_TWAPSeconds, _maxTWAPTickDifference);
102         TWAPSeconds = _TWAPSeconds;
103         maxTWAPTickDifference = _maxTWAPTickDifference;
104     }
```

Extract of Automator.sol

3.2 Impact

The consequence of the Revert TWAPS is that it necessitates accepting all types of tokens and pairs, binding them to the most volatile pair. This may expose high liquidity and a highly correlated pool to market manipulation and front-run attackers.

3.3 Recommendations

As this only affects the swapping logic, the Revert operator is able to define a minimum return of the swap to ensure the current price at execution time is still acceptable compared to the value calculated before off-chain.

Revert also has the ability to perform TWAPS control off-chain.

Off-chain or on-chain it would be more precise to have a TWAPS configuration for each pool.

3.4 Remediation

The Revert Finance team has acknowledged HYDN's comments and decided to keep the current configuration granularity.

4 Optimize Overkill Checks To Reduce Gas [Low]

4.1 Description

Including extra checks that are not strictly necessary can be considered overkill. Verifying that the update of the ***swapRouter*** value is not equal to the ***positionManager*** is not a crucial step.

The motivation behind this check, as explained in the comment, was to prevent the theft of approved NFTs. However, since the admin has the ability to execute arbitrary calls to any contract and has approval for any position NFT, they already hold a powerful position to potentially exploit the underlying assets of the position.

```
67      /**
68       * @notice Owner controlled function to change swap router (onlyOwner)
69       * @param _swapRouter new swap router
70       */
71      function setSwapRouter(address _swapRouter) external onlyOwner {
72
73          // don't let this ever be possible - it would enable owner to steal all approved NFTs
74          if (swapRouter == address(nonfungiblePositionManager)) {
75              revert InvalidConfig();
76          }
77
78          emit SwapRouterChanged(_swapRouter);
79          swapRouter = _swapRouter;
80      }
```

Extract of Automator.sol

4.2 Impact

Including extra checks which are not strictly necessary will increase deployment and execution gas costs.

4.3 Recommendations

Remove the unnecessary check.

4.4 Remediation

Recommendation applied by the Revert Finance team.

5 Invalid Memory State May Lead To Unexpected Situations and Extra Gas [Low]

5.1 Description

During contract execution, the variable config is set as storage, meaning that any changes made during the current execution will be directly reflected. While this behavior may be intentional and even permitted for direct value updates, in the actual use case, none of these actions are necessary. There is no expectation for the value to be updated, and the value is not directly reassigned.

This observation applies to both AutoExit and, to a lesser extent, AutoRange, with a lower impact.

```
96      function execute(ExecuteParams memory params) external {
97
98          if (msg.sender != operator) {
99              revert Unauthorized();
100         }
101
102         ExecuteState memory state;
103         PositionConfig storage config = positionConfigs[params.tokenId];
```

Extract of AutoRange.sol

```
168         // send it to current owner
169         nonfungiblePositionManager.safeTransferFrom(address(this), state.owner, state.newTokenId);
170
171         // protocol reward is calculated based on added amount (to incentivize optimal swap done by operator)
172         state.protocolReward0 = state.amountAdded0 * protocolRewardX64 / Q64;
173         state.protocolReward1 = state.amountAdded1 * protocolRewardX64 / Q64;
174
175         // send leftover to owner
176         if (state.amount0 - state.protocolReward0 - state.amountAdded0 > 0) {
177             _transferToken(state.owner, IERC20(state.token0), state.amount0 - state.protocolReward0 - state.amountAdded0, true);
178         }
179         if (state.amount1 - state.protocolReward1 - state.amountAdded1 > 0) {
180             _transferToken(state.owner, IERC20(state.token1), state.amount1 - state.protocolReward1 - state.amountAdded1, true);
181         }
182
183         // copy token config for new token
184         positionConfigs[state.newTokenId] = config;
185         emit PositionConfigured(
186             state.newTokenId,
187             config.lowerTickLimit,
```

Extract of AutoRange.sol

5.2 Impact

This would lead to extra gas costs and may lead to unexpected situations. For example, re-entrance is possible here, with a malicious token or simply with the NFT hook on transfer Line 169. The user calls ***configToken()***, and then the config line is set to Line 184 in place of the original one. Whilst this has no direct impact, bad actors may leverage it to find more harmful use cases.

5.3 Recommendations

Change the PositionConfig state from storage to memory to avoid any manipulation during execution and to reduce gas costs.

5.4 Remediation

PositionConfig state changed to memory by the Revert Finance team.

6 Native Transfer May Lead To Loss of Funds [Low]

6.1 Description

The ***receive()*** function only accepts native tokens from the wrapper native contract. This safeguard helps protect regular users from unintentionally transferring tokens directly to the contract. However, in more advanced scenarios, native tokens may become locked within the contract.

For instance, if a ***selfdestruct()*** function is triggered on the contract, it will force the transfer of native tokens. While the admin retains the ability to withdraw any ERC20 token using the ***withdrawBalance*** function, there is currently no provision for withdrawing native tokens.

6.2 Impact

In unknown or unexpected situations, native tokens may be locked and lost in the contract.

6.3 Recommendations

Add an admin function to withdraw native tokens in a similar way to ***withdrawBalance***.

6.4 Remediation

Withdraw native function added by the Revert Finance team.