SOURCEHAT

Ladder Protocol

Smart Contract Audit Report

EXECUTIVE SUMMARY

This report presents the outcomes of our collaborative engagement with the Ladder Protocol team, focusing on the comprehensive evaluation of the LadderProtocol and Stake contracts.

Our team conducted an initial security assessment from **April 11th** to **April 18th, 2024**. On **May 10th**, our team amended this report to reflect changes made to the contracts to resolve the findings we had identified during our initial review from commit `d5fea76` to commit `5b35d3f` .

Ladder Protocol is a new ERC-20 token with an innovative pricing structure. The Ladder Protocol token is pegged to a "peg" token. The contract mathematically calculates the token's current price based on various modes the contract can enter. In order to function properly, the contract is reliant on all of the tokens initially minted being added to liquidity. If they are not, due to the way the contract manipulates the Pair's reserves, it is possible for liquidity to decrease to a point where users will receive a minimal amount of the peg token when selling Ladder tokens.

AUDIT SCOPE

Name	Source Code	Visualized
LadderProtocol	`5b35d3f`	Inheritance Chart. Function Graph.
Stake	`5b35d3f`	Inheritance Chart. Function Graph.

AUDIT FINDINGS

A Medium finding was identified and was partially resolved. In addition, centralized aspects are present.

Finding #1 LadderProtocol — High Resolved

Description: The contract intends to maintain a minimum peg of 1:1 between Ladder tokens and the peg token in the DEXV2Pair. In order to set the initial peg, the contract contains a HARD_UPGRADE() function. This sets the contracts expected price that is used in all subsequent price calculations. However, this function can be called again by the owner with a specified price. This will cause the contract to add or remove Ladder tokens from the Pair's reserves to reach the specified price.

Risk/Impact: This function can be used to perform a rug pull. By specifying a price lower than the current one, users' tokens become more valuable. This allows users who sell directly after the price is changed to sell their tokens for much more of the underlying peg token. As a result, the liquidity in the Pair can be decreased to a point where users who did not sell directly after the price change to receive minimal peg tokens for their ladder tokens.

Recommendation: The team should be unable to change the amount of tokens in the the Pair's reserves after deployment.

Resolution: The team has changed the HARD_UPGRADE() function such that it cannot be called after the initial contract setup.

Finding #2 LadderProtocol — High Acknowledged

Description: The contract intends to maintain a minimum peg of 1:1 between Ladder tokens and the peg token in the DEXV2Pair. In order to maintain this peg the contract calculates how many Ladder tokens should be in the DEXV2Pair based on its peg token reserves. If

the token would fall below the peg, the contract removes tokens from the pair's ladder reserves.

```
uint  expectedDiff = pairLadderBal.sub(newAmount);
uint  expectedPairAmount = pairLadderBal.sub(expectedDiff);
_balances[address(DEXV2Pair)].balance = expectedPairAmount.sub(_amount);
contractAmount = _amount.add(expectedDiff);
DEXV2Pair.sync();
_balances[address(DEXV2Pair)].balance = expectedPairAmount;
```

Risk/Impact: When removing token's from the DEXV2Pair's ladder reserves, the overall amount of liquidity is decreased. If this were to continue enough times, the overall liquidity can reach a point where users are not able to buy or sell their tokens. This also allows for more impactful sandwich attacks. A malicious user can monitor the memory pool for a transaction that would cause liquidity to be removed. The malicious user's transactions would be more impactful on the resulting smaller liquidity pool.

Recommendation: The team should not enforce a minimum peg by removing liquidity from the pair.

Resolution: The team has expressed their intent to add the total Ladder token supply to liquidity. Additionally, there are no external mint or burn functions. This means that theoretically the Ladder token price should never fall below the initial peg and thus the above scenario should not happen.

Finding #3 LadderProtocol & Stake — Medium — Partially Resolved

Description: The LadderProtocol contract collects fees that are later used to perform a swapback for the peg token. The swapback() function is public and can be called by any user. Users can also determine which transactions will execute the swapback. During the course of the LadderProtocol swapback, the Stake contract performs a second swap if its print token is different from the LadderProtocol's peg token.

Risk/Impact: As the both the LadderProtocol swapback() function and the Stake contract's swap() function do not have any minimum amount out, a malicious user can manipulate the reserves for either trade. This can cause either contract to perform a trade at a loss. This will also lead to less rewards distributed for the Stake contract stakers.

Recommendation: The team should consider implementing a Time-Weighted Average Price (TWAP) Oracle. The swapback() function should additionally not be made public so malicious users cannot cause the contract to perform trades when the reserves are manipulated.

Resolution: The team has made the swapback() function private preventing users from manually triggering a swapback. However, it is still possible to track when a swapback would occur and sandwich the transaction. The team must ensure that the amount of tokens being swapped during a swapback is minimal enough such that the price impact is too small to allow a sandwich attack to be profitable.

Finding #4 LadderProtocol — Informational — Resolved

Description: The _srebase() and SOFT_UPGRADE() functions check that the current reserve peg is greater than or less than the last reserve peg.

```
if(LADDER_MODE && RESERVE_PEG_LP > (LAST_RESERVE_PEG_LP.mul(Reserve_flactuate_rate)).div(pegDenominator)
|| LADDER_MODE && RESERVE_PEG_LP < (LAST_RESERVE_PEG_LP.mul(Reserve_flactuate_rate)).div(pegDenominator))
```

Recommendation: Rather than checking the less than condition and the greater than condition, the team can simplify this to one check where the last reserve peg is not equal to the reserve peg.

```
if(LADDER_MODE && RESERVE_PEG_LP != (LAST_RESERVE_PEG_LP.mul(Reserve_flactuate_rate)).div(pegDenominator))
```

Resolution: The team has adjusted this logic such that both the if statements are required.

STAKE SYSTEM OVERVIEW

Staking & Unstaking

The Staking contract is associated with a "Caller" token intended to be Ladder tokens. Any user may stake the minimum stake amount Ladder tokens when a swap is not being performed. The minimum stake amount default to 100 tokens though the owner may set it to any value at any time.

Any user may withdraw their staked Ladder tokens when a swap is not being performed.

Rewards

This contract accumulates rewards in the form of a specified "print" token when the Ladder token initiates this contract to perform a swap. The specified "swap" token will be swapped for the print token if the swap token is not the print token. The print token must be paired with the swap token in a liquidity pool or a 2-step swap path through WETH if it does not. The swap will not be performed if the swap or first swap in the path does not yield 100 wei based on the current liquidity pool reserves. These reserves can potentially be manipulated causing the contract to swap for a lesser amount or causing a DOS. A Treasury fee is taken and transferred to the Treasury address if the "treasury portion" set by the owner is non-zero. The owner may set the treasury portion to any value and update the Treasury address at any time.

The print tokens remaining after the Treasury fee are distributed as rewards if "stake on" is enabled. The print tokens are transferred back to the Ladder token contract if stake on is not enabled. The owner may toggle the stake on value at any time. Users will be credited with rewards based on the amount of Ladder tokens they have staked proportionally to the total amount of Ladder tokens staked. Any dust remaining after rewards are distributed will be transferred back to the Ladder token contract.

Users may claim their credited rewards for a specified token when the contract is not performing a swap. A claim tax is taken from the rewards and transferred to the Treasury address if the "claim tax" is non-zero. The owner may set the claim tax to any value at any time.

The owner may withdraw and tokens not allocated for rewards at any time. The owner may withdraw any ETH from the contract at any time.

LADDERPROTOCOL SYSTEM OVERVIEW

General

The LadderProtocol intends to maintain an "expected price" per "peg token" in a DEXV2Pair. The price is originally determined by the "base price" provided when the owner calls HARD_UPGRADE(). The corresponding amount of Ladder tokens are added or removed from the DEXV2Pair reserves such that the expected price is reached. However, the team intends to use the price returned from priceNow() function. The priceNow() function returns the current Ladder price based on the DEXV2Pair's reserves. Users will be unable to buy or sell Ladder tokens due to DOS until the HARD_UPGRADE() function is called, though regular transfers may still occur.

The DEXV2Pair is set through an owner-restricted call to the setAMMPair() function. This can also be used to add other pair addresses to the list of non-DEXV2Pair AMMs. Users are unable to add liquidity or sell through a pair on the list of AMMs.

The expected price for the DEXV2Pair changes based on the contract's current mode. The LadderProtocol token has 5 different modes:

- Stable mode
- Appreciation/Depreciation mode
- APY mode
- Ladder mode
- Blast mode

The HARD_UPGRADE() function will start the contract in Stable mode.

Stable Mode

While there is a STABLE_MODE variable, it is not used meaningfully in the contract. Stable mode can instead be defined as the contract being in neither Ladder mode or Blast mode.

When a user performs a buy where the recipient is not a "liquidity provider" and the contract is not in Blast mode, a buy fee is taken if the recipient is not excluded from fees. A "buy rebase" also occurs.

During a Stable mode buy rebase, the contract first calculates the amount of peg tokens was required to purchase the Ladder tokens. This is used with the expected price to calculate the expected DEXV2Pair's current Ladder token reserves. The contract then transfers or mints the contract Ladder tokens if its current Ladder reserves are too low. It transfers tokens from the DEXV2Pair to the contract if the reserves are too high.

When a user performs a sell where the sender is not a "liquidity provider" and the contract is not in Blast mode, a sell fee is taken if the sender is not excluded from fees and the contract is not in Depreciation mode. An additional fee may be taken based on the last time the seller performed a buy if the "high frequency tax" is enabled. The seller will be taxed the "hft" fee is their last buy was within the "hft timing threshold". The seller will be taxed the "quick sell" fee if their last buy was within the "quick sell timing threshold". The owner may set the hft timing threshold and quick sell timing threshold to any values at any time. If the recipient is not excluded from the max transaction amount and "LTM" is enabled, the maximum amount of tokens that may be sold in one transaction is the "GSL" percent of the DEXV2Pair's current peg token reserves. A "sell rebase" also occurs.

During a Stable mode sell rebase, the contract first calculates the amount of peg tokens the user should receive for the Ladder tokens they are selling. This is used with the expected price to calculate the expected DEXV2Pair's current Ladder token reserves. The contract then transfers or mints the contract Ladder tokens if its current Ladder reserves are too low. It transfers tokens from the DEXV2Pair to the contract if the reserves are too high. Additionally, if the DEXV2Pair's current Ladder token reserves before they were adjusted were greater than the "ladder threshold" the contract will enter Ladder mode and the "last reserve peg" is set to the current DEXV2Pair's peg token reserves.

Appreciation/Depreciation Mode

This mode is defined as Buy Appreciation and Sell Depreciation. The contract will only enter this mode if enabled by the owner. It is intended to limit the change of the expected price during buys and sells. The main difference between Appreciation/Depreciation mode and Stable mode is during the buy rebase and sell rebase functionality. These functions are triggered in the same way as in Stable mode.

During a Buy Appreciation buy rebase, the contract similarly calculates the expected DEXV2Pair's expected Ladder reserves based on the amount of Ladder tokens being bought. However, in Buy Appreciation mode the contract potentially updates the expected price. If Volume mode is enabled, the contract increases the expected price based on the "higher threshold" and the amount of tokens being purchased. The higher threshold is set during an _upgradeApp() call. It is the expected price at the time of the call plus the "app" rate. If volume mode is not enabled, the new price will be equal to the current expected price plus the "buy appreciation rate". In both cases if the newly calculated price is less than the current real price, the expected price and corresponding ladder reserves will be set to the newly calculated values.

During a Sell Depreciation sell rebase, the contract similarly calculates the amount of peg tokens the user should receive for the Ladder tokens they are selling. However, in this mode if the current expected price is more than 100 wei greater than the lower threshold the contract potentially updates the expected price. The lower threshold is calculated in the same manner as the higher threshold except the app rate is subtracted from the expected price rather than added. Additionally, the Volume mode and non-Volume mode new prices are calculated the same way as a Buy Appreciation buy rebase where the lower threshold and "sell depreciation rate" values are subtracted rather than added. In both cases if the newly calculated price is less than the current real price, the expected price and corresponding ladder reserves will be set to the newly calculated values.

APY Mode

The contract will only enter APY mode if enabled by the owner. In APY mode the expected price increases based on the number of elapsed "APY periods". The owner may set the APY period to any value at any time. The expected price increases by the "APY rate" per elapsed period. The owner may set the APY rate to any value at any time. The higher threshold and lower threshold will be updated for Appreciation/Depreciation mode accordingly. When enabled, the contract will perform the APY functionality during a sell rebase if the contract is not in Ladder mode.

Ladder Mode

The contract will enter Ladder mode if manually set by the owner or during a sell rebase if the DEXV2Pair's peg token reserves have reached the "ladder threshold". The "last reserve peg" will also be set.

During each sell rebase the contract will compare the last reserve peg to the DEXV2Pair's current peg token reserves. If the current reserves are greater than the last reserve peg, the expected price will increase by the "peg fluctuate rate". If the current reserves are less than last reserve peg, the expected price will decrease by the peg fluctuate rate. In both cases the last reserve peg is updated to the current reserves. Additionally, the higher threshold and lower threshold are set accordingly for the Appreciation/Depreciation mode.

Blast Mode

The contract will enter Blast mode is manually set by the owner or during a mint if the token's total supply reaches the "blast threshold". When entering Blast mode, the blast threshold will increase by the "blast threshold increasing rate" until reaching the max supply. The owner may set the blast threshold

to any value at any time. The owner may increase the blast threshold increasing rate by any amount at any time. Entering Blast mode will also set the Treasury and Printing fees to 50%.

During a Blast mode buy, the "blast fee" of 1% will be taken from the purchased amount.

During a Blast mode sell, the blast fee will also be taken from the sold amount. Similarly to the standard sell rebase, the contract will calculate the current price after the sell based on the DEXV2Pair's current reserves. If the newly calculated price has fallen below the expected price, the contract will exit Blast mode. This will set the Treasury and Printing fee back to 20%, set the expected price to the newly calculated price, updated the higher threshold and lower threshold for Appreciation/Depreciation mode, and set the last reserve peg for Ladder mode.

Swapback

During all transfers, if there is a fee to be collected it is stored in the contract. If "printing" is enabled, the "printing amount" of the fee being collected will be reserved for printing. The contract will attempt to perform an automated swapback during any sell. Any users may call the swapback functionality at any time.

The contract will only perform the swapback if the contract's fees reserved for print is greater than the "minimum swapback" amount. When LTM is enabled the contract can only swap up to the "GSL" percentage of the DEXV2Pair's current peg reserves. Otherwise, the max swap amount is 2% of the contract's total supply.

The Ladder tokens will be swapped for peg tokens. These tokens will then be swapped in the Stake contract to be distributed as rewards for stakers.

Soft Upgrade

The contract includes a public SOFT_UPGRADE() function that can be called by any user at any time. The function will also execute during any transfer that is not a buy or a sell.

The SOFT_UPGRADE() function will only perform any functionality if the contract is in Blast mode and APY mode or in Blast mode and Ladder mode. This function implements the functionality of other modes. The APY functionality will be executed if the contract is in APY mode. The sell rebase Ladder mode functionality will be executed if the contract is in Ladder mode.

In both cases the contract will "recover". This is the same functionality as executed in the HARD_UPGRADE() function but with the current expected price rather than an owner specified price. The corresponding amount of Ladder tokens are added or removed from the DEXV2Pair reserves such that the expected price is reached.

AirDrop & SafeTransfer

The contract also includes airdrop functionality. This allows the owner to transfer a specified amount of Ladder tokens from the contract to any number of addresses. The contract will not transfer Ladder tokens that have been reserved for printing.

The SafeTransfer function allows the owner to transfer any amount of tokens or ETH to a a specified address.

VULNERABILITY ANALYSIS

Vulnerability Category	Notes	Result
Arbitrary Jump/Storage Write	N/A	PASS
Centralization of Control	• The Stake owner may set the claim tax to any value. • The Stake owner may disable rewards distributions at any time. • The LendingProtocol owner may HARD_UPGRADE at any time. • The LendingProtocol owner may update the contract's current mode. • The LendingProtocol owner may prevent adding liquidity and sells on specified AMMs. • The team must ensure that all tokens are added to liquidity in order to ensure the contract functions as intended.	WARNING
Compiler Issues	N/A	PASS
Delegate Call to Untrusted Contract	N/A	PASS
Dependence on Predictable Variables	N/A	PASS
Ether/Token Theft	N/A	PASS
Flash Loans	N/A	PASS
Front Running	N/A	PASS
Improper Events	N/A	PASS
Improper Authorization Scheme	N/A	PASS
Integer Over/Underflow	N/A	PASS
Logical Issues	If the LadderToken were to fall below the minimum peg, liquidity could be removed to the point users cannot buy or sell.	WARNING
Oracle Issues	N/A	PASS
Outdated Compiler Version	N/A	PASS
Race Conditions	N/A	PASS
Reentrancy	N/A	PASS

Vulnerability Category	Notes	Result
Signature Issues	N/A	PASS
Sybil Attack	N/A	PASS
Unbounded Loops	N/A	PASS
Unused Code	N/A	PASS
Overall Contract Safety		WARNING

ABOUT SOURCEHAT

SourceHat has quickly grown to have one of the most experienced and well-equipped smart contract auditing teams in the industry. Our team has conducted 1800+ solidity smart contract audits covering all major project types and protocols, securing a total of over $50 billion U.S. dollars in on-chain value!

Our firm is well-reputed in the community and is trusted as a top smart contract auditing company for the review of solidity code, no matter how complex. Our team of experienced solidity smart contract auditors performs audits for tokens, NFTs, crowdsales, marketplaces, gambling games, financial protocols, and more!

Contact us today to get a free quote for a smart contract audit of your project!

WHAT IS A SOURCEHAT AUDIT?

Typically, a smart contract audit is a comprehensive review process designed to discover logical errors, security vulnerabilities, and optimization opportunities within code. A *SourceHat Audit* takes this a step further by verifying economic logic to ensure the stability of smart contracts and highlighting privileged functionality to create a report that is easy to understand for developers and community members alike.

HOW DO I INTERPRET THE FINDINGS?

Each of our Findings will be labeled with a Severity level. We always recommend the team resolve High, Medium, and Low severity findings prior to deploying the code to the mainnet. Here is a breakdown on what each Severity level means for the project:

- **High** severity indicates that the issue puts a large number of users' funds at risk and has a high probability of exploitation, or the smart contract contains serious logical issues which can prevent the code from operating as intended.
- **Medium** severity issues are those which place at least some users' funds at risk and has a medium to high probability of exploitation.
- **Low** severity issues have a relatively minor risk association; these issues have a low probability of occurring or may have a minimal impact.
- **Informational** issues pose no immediate risk, but inform the project team of opportunities for gas optimizations and following smart contract security best practices.