ENTERSOFT

DeltaPrime

Smart Contract Audit

DATE: 8th November 2024 | PREPARED BY: Entersoft Australia

Contents

Revision History & Version Control

Start Date	End Date	Author	Comments/Details
05 Sep 2024	08 Nov 2024	Gurkirat, Shashank Wangkar and Laxmi Prasad	Interim Release for the Client

Reviewed by	Released by
Nishita Palaksha	Nishita Palaksha

Entersoft was commissioned to perform a source code review on DeltaPrime smart contracts. The review was conducted between **5 September 2024 to 8 November 2024** . The report is organized into the following sections.

- Executive Summary: A high-level overview of the security audit findings.
- Technical analysis: Our detailed analysis of the Smart Contract code

The information in this report should be used to understand overall code quality, security, correctness, and meaning that code will work as described in the smart contract.

1.0 Disclaimer

This is a limited audit report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to: (i) smart contract best coding practices and vulnerabilities in the framework and algorithms based on white paper, code, the details of which are set out in this report, (Smart Contract audit). To get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us based on what it says or does not say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Entersoft Australia and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (Entersoft) owe no duty of care towards you or any other person, nor does Entersoft make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and Entersoft hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, Entersoft hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Entersoft, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report. The analysis of the Smart contract is purely based on the smart contract code shared with us alone.

2.0 Overview

2.1 Project Overview

During the period of **05 Sep 2024 to 08 Nov 2024**, Entersoft performed smart contract security audits for **Deltaprime**.

2.2 Scope

The scope of this audit was to analyze and document the smart contract codebase for quality, security, and correctness.
Repository link :- https://github.com/DeltaPrimeLabs/deltaprime-primeloans/commits/audit/post-tge/

The following files were reviewed as part of the scope:

- deltaprime-primeloans/contracts/*

Except the provided Excluded Files.

Commit ID/Hash: 6497604a64091f9899e48baee98e96498c310868

2.3 Project Summary

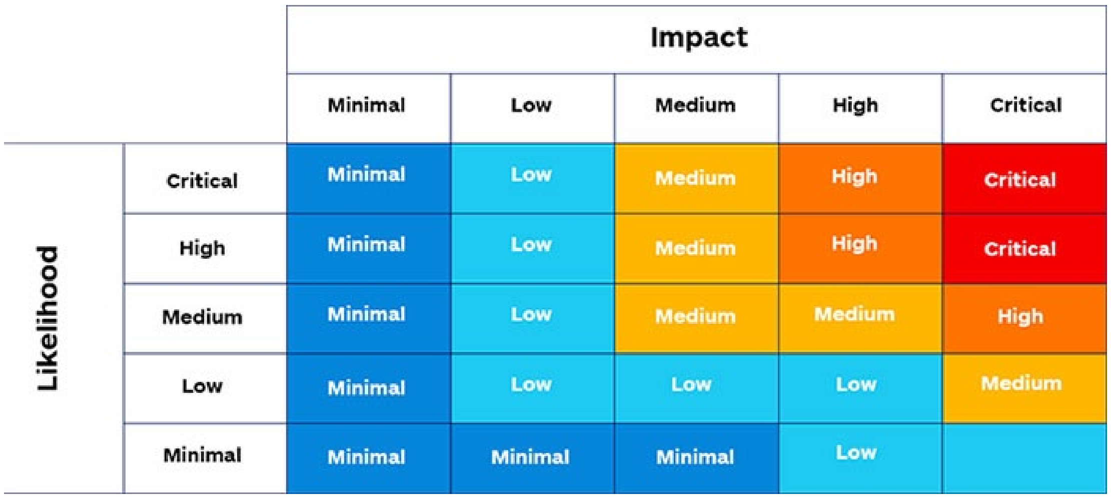
Project Name	No. of Smart Contract File(s)	Verified	Vulnerabilities
Deltaprime	1	Yes	As per report. Section 2.6

2.4 Audit Summary

Delivery Date	Method of Audit	Consultants Engaged
08 Nov 2024	Automated and Manual Methodology	3

2.5 Security Level References

Every vulnerability in this report was assigned a severity level from the following classification table:

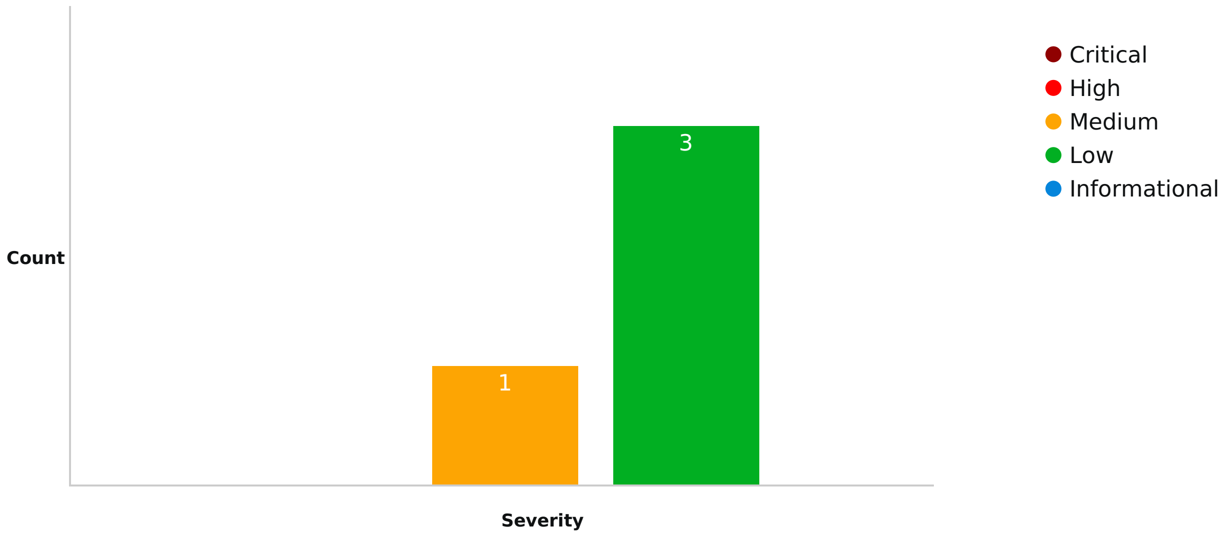


		Impact				
		Minimal	Low	Medium	High	Critical
Likelihood	Critical	Minimal	Low	Medium	High	Critical
	High	Minimal	Low	Medium	High	Critical
	Medium	Minimal	Low	Medium	Medium	High
	Low	Minimal	Low	Low	Low	Medium
	Minimal	Minimal	Minimal	Minimal	Low	

2.6 Vulnerability Summary

● Critical	● High	● Medium	● Low	● Informational
0	0	1	3	0

Vulnerabilities



3.0 Executive Summary

Entersoft Audits has conducted an in-depth security review of the Delta Prime Protocol's smart contracts. This audit aimed to identify and mitigate potential vulnerabilities, ensuring that the protocol adheres to the highest standards of security, reliability, and performance. By leveraging a meticulous and systematic approach, the audit sought to fortify the protocol against a wide range of risks. Our methodology emphasized a manual, expert-driven review process, moving beyond conventional reliance on automated tools. While automation provided initial insights, the cornerstone of our approach was a thorough, hands-on analysis to uncover subtle, complex vulnerabilities that automated systems might overlook. The audit process focused not only on identifying security gaps but also on providing actionable recommendations to address them. This ensures that the protocol is not only secure in its current form but also robust enough to withstand evolving threats. Several issues were discovered and categorized, as detailed in the Vulnerability Summary Table, with clear guidance for resolution provided to the Delta Prime team. By addressing these vulnerabilities, the Delta Prime Protocol is now better positioned to deliver a secure, reliable, and seamless experience to its users.

Importantly, our audit process intentionally avoided reliance solely on automated tools, emphasizing a more in-depth and nuanced approach to security analysis. Conducted from 5th of September to 8th of November, our team diligently assessed and validated the security posture of the solidity smart contract, ultimately finding a number of vulnerabilities as per vulnerability summary table.

Testing Methodology:

We have leveraged static analysis techniques extensively to identify potential vulnerabilities automatically with the aid of tools such as Slither. Apart from this, we carried out extensive manual testing to iron out vulnerabilities that could slip through an automated check. This included a variety of attack vectors like reentrancy attacks, overflow and underflow attacks, timestamp dependency attacks, and more.
While going through the due course of this audit, we also ensured to cover edge cases, and built a combination of scenarios to assess the contracts' resilience. Our attempt to leave no stone unturned involved coming up with both negative and positive test cases for the system, and grace handling of stressed scenarios.
Our testing methodology in Solidity adhered to industry standards and best practices, integrating partially implemented OWASP and NIST SP 800 standards for encryption and signatures. Solidity's renowned security practices were complemented by tools such as Solhint for linting, and the Solidity compiler for code optimization. Sol-profiler, Sol-coverage, and Sol-sec were employed to ensure code readability and eliminate unnecessary dependencies.

Tools Used for Audit:

In the course of our audit, we leveraged a suite of tools to bolster the security and performance of our program. While our team drew on their expertise and industry best practices, we also integrated various tools into our development environment. Noteworthy among them are Slither. This holistic approach ensures a thorough analysis, uncovering potential issues that automated tools alone might overlook. Entersoft takes pride in utilizing these tools, which significantly contribute to the quality, security, and maintainability of our codebase.

Code Review / Manual Analysis:
Our team conducted a manual analysis of the Solidity smart contracts to identify new vulnerabilities or to verify vulnerabilities found during static and manual analysis. We carefully analyzed every line of code and made sure that all instructions provided during the onboarding phase were followed. Through our manual analysis, we were able to identify potential vulnerabilities that may have been missed by automated tools and ensure that the smart contract was secure and reliable.

Auditing Approach and Methodologies Applied:
The solidity smart contract was audited in a comprehensive approach to ensure the highest level of security and reliability. Careful attention was given to the following key areas to ensure the overall quality of code:

- Code quality and structure: We conducted a detailed review of the codebase to identify any potential issues related to code structure, readability, and maintainability. This included analyzing the overall architecture of the solidity smart contract and reviewing the code to ensure it follows best practices and coding standards.

- Security vulnerabilities: Our team used manual techniques to identify any potential security vulnerabilities that could be exploited by attackers. This involved a thorough analysis of the code to identify any potential weaknesses, such as buffer overflows, injection vulnerabilities, Signatures, and deprecated functions.

3.1 Findings

Vulnerability ID	Contract Name	Severity	Status
1	VestingDistributor	● Medium	Pending
2	LinearIndex, AddressProvider	● Low	Pending
3	BtcEligibleUsersList	● Low	Pending
4	DepositSwap	● Low	Pending

3.2 Recommendations

Documentation Overview

Documentation is crucial for any project as it ensures clarity and understanding, helping developers and users grasp how the code and protocol function. It aids in new contributors, auditors facilitating maintenance and troubleshooting by providing insights into design decisions.

For blockchain protocols, thorough documentation is particularly important due to the complexity and need for security and transparency.

Below is the required documentation, rated according to the quality as provided by the protocol.

Protocol Documentation:

- Purpose: To describe the overall protocol, its goals, architecture, and how it operates.
- Quality Provided: High

Technical Documentation:

- Purpose: To provide detailed information about the technical aspects of the protocol, including smart contracts, APIs, and their interactions.
- Quality Provided: High

Inline Comments:

- Purpose: To explain specific parts of the code, such as the purpose of functions, variables, and logic.
- Quality Provided: Medium

Smart Contract Overview

Overall, the smart contracts adhere to industry best practices and security standards. While few vulnerabilities were identified and are assembled in the report.

Entersoft team encountered issues with version compatibility when setting up the development environment, as the provided Hardhat configuration and test scripts were outdated. DeltaPrime has indicated that these test scripts will be updated in the forthcoming months. To avoid delays, the Entersoft prioritized a comprehensive manual code review while keeping the protocol team informed of the setup challenges.

4.0 Technical Analysis

4.1 Missing Zero Address Checks

Severity	Status	Type of Analysis
● Medium	Identified	Static

Contract Name:

VestingDistributor

Description:

The VestingDistributor contract's constructor does not validate that the poolAddress and keeperAddress parameters are non-zero addresses. Zero addresses are often used as a placeholder or default value and can lead to unintended behavior if not properly checked.

Locations:

https://github.com/DeltaPrimeLabs/deltaprime-primeloans/blob/6497604a64091f9899e48baee98e96498c310868/contracts/VestingDistributor.sol#L52

Remediation:

Add checks in the constructor to ensure that neither poolAddress nor keeperAddress is a zero address. Implement the following checks:

```
constructor(address poolAddress, address keeperAddress) {
    require(poolAddress != address(0), "Invalid pool address");
    require(keeperAddress != address(0), "Invalid keeper address");
    pool = Pool(poolAddress);
    poolToken = IERC20Metadata(pool.tokenAddress());
    keeper = keeperAddress;
    lastUpdated = block.timestamp;
}
```

These checks will prevent the contract from being deployed with invalid addresses, ensuring proper configuration and reducing the risk of operational issues.

Impact:

1. **Potential Misconfiguration:**
    - Deploying the contract with zero addresses for critical roles could lead to misconfiguration, rendering the contract unusable or insecure.

2. **Operational Risks:**

    - If poolAddress is zero, interactions with the Pool contract will fail, affecting all functionalities dependent on it.
    - A zero keeperAddress would prevent any keeper-specific operations, as no valid keeper would be able to execute them.

3. **Security Concerns:**

    - Zero addresses can be exploited in certain scenarios, leading to potential security vulnerabilities.

Code Snippet:

```
        constructor(address poolAddress, address keeperAddress) {

pool = Pool(poolAddress);

poolToken = IERC20Metadata(pool.tokenAddress());

keeper = keeperAddress;

lastUpdated = block.timestamp;

}
```

Reference:

NA

Proof of Vulnerability:

N.A.

4.2 Missing disableinitializers() in constructor

Severity	Status	Type of Analysis
● Low	Identified	Dynamic

Contract Name:

LinearIndex, AddressProvider

Description:

All initialize calls in the implementation contract should be blocked to prevent potential security vulnerabilities. An uninitialized contract is susceptible to takeover by an attacker, which poses risks for both the proxy and its implementation contract. In the case of an implementation contract, it's crucial to call `_disableInitializers()` within the `constructor()` to prevent unauthorized initialization.

Locations:

LinearIndex, AddressProvider

Remediation:

We recommend using the `_disableInitializers()` function to prevent any initialize calls within the implementation. This can be achieved by adding the following code to the constructor:

```
constructor() {      _disableInitializers(); }
```

Impact:

Blocking initialization calls in the implementation contract secures it against takeover attacks, ensuring that only authorized parties can initialize the proxy. This enhances the security and integrity of the contract system, safeguarding it from unauthorized access or malicious manipulation.

Code Snippet:

NA

Reference:

Proof of Vulnerability:

N.A.

4.3 No check for repetition of Users in BtcEligibleUsersList::constructor() and BtcEligibleUsersList::addEligibleUsers()

Severity	Status	Type of Analysis
● Low	Identified	Dynamic

Contract Name:

BtcEligibleUsersList

Description:

I n constructor() and addEligibleUsers(), _eligibleUsersList is being given , there is no check for repetition of user addresses.This can cause a disturbance in managing eligibleUsers in eligibleUsersList.

Locations:

Line no: 9-14

Line no: 16-20

Remediation:

~~~
add the below check
```require(!isadded[_eligibleUsersList[i]], "already added");
isadded[_eligibleUsersList[i]] = true;
```
~~~

Impact:

Can create a disturbance in managing eligibleUsers in eligibleUsersList

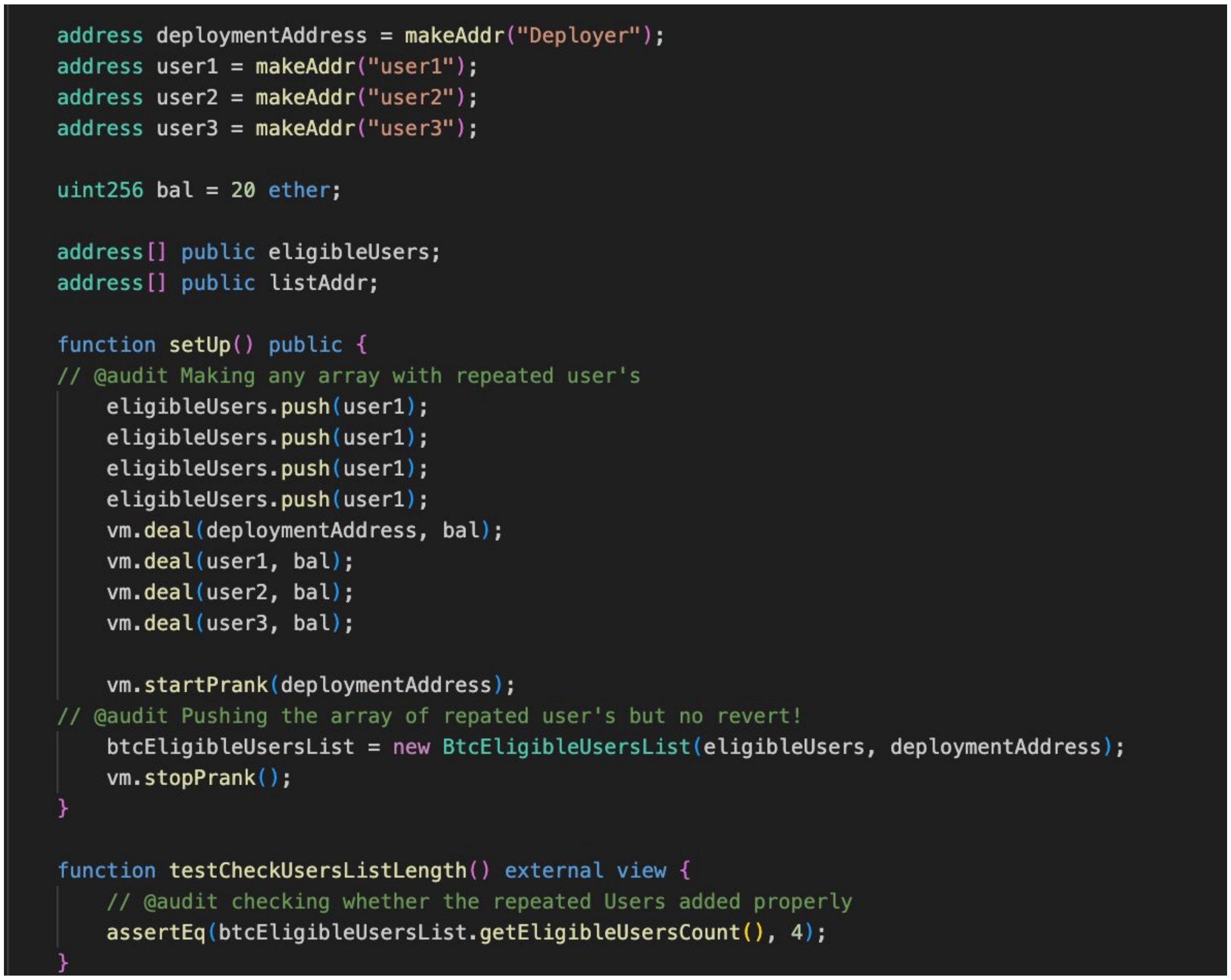
Code Snippet:

```
constructor(address[] memory _eligibleUsersList){
        // @audit no check for repitition of Users
        eligibleUsersList = _eligibleUsersList;
        _transferOwnership(0xBd2413135f3aab57195945A046cCA4e4bacD5a5b);
    }



function addEligibleUsers(address[] calldata _eligibleUsers) external onlyOwner {
        // @audit no check for repitition of Users
        for (uint256 i = 0; i < _eligibleUsers.length; i++) {
            eligibleUsersList.push(_eligibleUsers[i]);
        }
    }
```

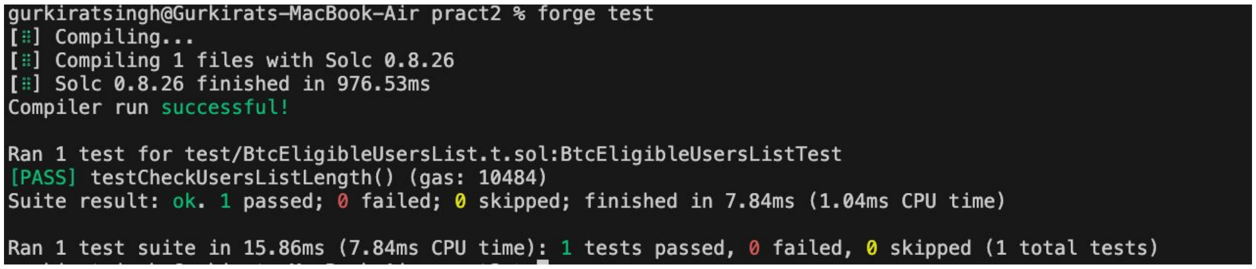
Reference:

Proof of Vulnerability:

BtcEligibleUsersList.t.sol



```solidity
    address deploymentAddress = makeAddr("Deployer");
    address user1 = makeAddr("user1");
    address user2 = makeAddr("user2");
    address user3 = makeAddr("user3");

    uint256 bal = 20 ether;

    address[] public eligibleUsers;
    address[] public listAddr;

    function setUp() public {
    // @audit Making any array with repeated user's
        eligibleUsers.push(user1);
        eligibleUsers.push(user1);
        eligibleUsers.push(user1);
        eligibleUsers.push(user1);
        vm.deal(deploymentAddress, bal);
        vm.deal(user1, bal);
        vm.deal(user2, bal);
        vm.deal(user3, bal);

        vm.startPrank(deploymentAddress);
    // @audit Pushing the array of repated user's but no revert!
        btcEligibleUsersList = new BtcEligibleUsersList(eligibleUsers, deploymentAddress);
        vm.stopPrank();
    }

    function testCheckUsersListLength() external view {
        // @audit checking whether the repeated Users added properly
        assertEq(btcEligibleUsersList.getEligibleUsersCount(), 4);
    }
```



```
gurkiratsingh@Gurkirats-MacBook-Air pract2 % forge test
[⠊] Compiling...
[⠊] Compiling 1 files with Solc 0.8.26
[⠊] Solc 0.8.26 finished in 976.53ms
Compiler run successful!

Ran 1 test for test/BtcEligibleUsersList.t.sol:BtcEligibleUsersListTest
[PASS] testCheckUsersListLength() (gas: 10484)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 7.84ms (1.04ms CPU time)

Ran 1 test suite in 15.86ms (7.84ms CPU time): 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

4.4 No checks in DepositSwap::depositSwap() on minAmountToToken

Severity	Status	Type of Analysis
● Low	Identified	Dynamic

Contract Name:

DepositSwap

Description:

The `depositSwap()` function lacks a necessary check to ensure that `minAmountToToken()` is greater than 0. This validation is already correctly implemented in the `depositSwapParaSwap()` function. Without this check, invalid `minAmountToToken()` values could be provided, increasing the risk of sandwich and frontrunning attacks.

Locations:

DepositSwap -- 80

Remediation:

Add the following check to prevent invalid values for`minAmountToToken()`: `require(minAmountToToken > 0, "Invalid minAmountToToken");`

Impact:

Failing to validate `minAmountToToken()` may allow for incorrect input, leaving the protocol vulnerable to sandwich and frontrunning attacks, and potentially causing denial of service (DoS) conditions.

Code Snippet:

```
        function depositSwap(uint256 amountFromToken, uint256 minAmountToToken, address[] calldata path, address[] callda
ta adapters) public {

address fromToken = path[0];

address toToken = path[path.length - 1];

require(_isTokenSupported(fromToken), "fromToken not supported");

require(_isTokenSupported(toToken), "toToken not supported");

Pool fromPool = _tokenToPoolTUPMapping(fromToken);

Pool toPool = _tokenToPoolTUPMapping(toToken);

address user = msg.sender;

amountFromToken = Math.min(fromPool.balanceOf(user), amountFromToken);

_withdrawFromPool(fromPool, IERC20(fromToken), amountFromToken, user);

_yakSwap(path, adapters, amountFromToken, minAmountToToken);

_depositToPool(toPool, IERC20(toToken), IERC20(toToken).balanceOf(address(this)), user);

}
```

Reference:

Proof of Vulnerability:

N.A.

5.0 Auditing Approach and Methodologies applied

Throughout the audit of the smart contract, care was taken to ensure:

- Overall quality of code
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Mathematical calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of token standards.
- Efficient use of gas.
- Code is safe from Re-entrancy and other vulnerabilities.

A combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding the scope of the smart contract audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

5.1 Structural Analysis

In this step we have analysed the design patterns and structure of all smart contracts. A thorough check was completed to ensure all Smart contracts are structured in a way that will not result in future problems.

5.2 Static Analysis

Static Analysis of smart contracts was undertaken to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

5.3 Code Review / Manual Analysis

Manual Analysis or review of done to identify new vulnerabilities or to verify the vulnerabilities found during the Static Analysis. The contracts were completely manually analysed, and their logic was checked and compared with the one described in the whitepaper. It should also be noted that the results of the automated analysis were verified manually.

5.4 Gas Consumption

In this step, we checked the behaviour of all smart contracts in production. Checks were completed to understand how much gas gets consumed, along with the possibilities of optimisation of code to reduce gas consumption.

5.5 Tools & Platforms Used For Audit

Slither

5.6 Checked Vulnerabilities

We have scanned Deltaprime smart contracts for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC-20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

6.0 Limitations on Disclosure and Use of this Report

This report contains information concerning potential details of Deltaprime and methods for exploiting them. Entersoft recommends that special precautions be taken to protect the confidentiality of both this document and the information contained herein. Security Assessment is an uncertain process, based on past experiences, currently available information, and known threats. All information security systems, which by their nature are dependent on human beings, are vulnerable to some degree. Therefore, while Entersoft considers the major security vulnerabilities of the analyzed systems to have been identified, there can be no assurance that any exercise of this nature will identify all possible vulnerabilities or propose exhaustive and operationally viable recommendations to mitigate those exposures. In addition, the analysis set forth herein is based on the technologies and known threats as of the date of this report. As technologies and risks change over time, the vulnerabilities associated with the operation of the Smart Contract described in this report, as well as the actions necessary to reduce the exposure to such vulnerabilities will also change. Entersoft makes no undertaking to supplement or update this report based on changed circumstances or facts of which Entersoft becomes aware after the date hereof, absent a specific written agreement to perform the supplemental or updated analysis. This report may recommend that Entersoft use certain software or hardware products manufactured or maintained by other vendors. Entersoft bases these recommendations upon its prior experience with the capabilities of those products. Nonetheless, Entersoft does not and cannot warrant that a particular product will work as advertised by the vendor, nor that it will operate in the manner intended. This report was prepared by Entersoft for the exclusive benefit of Deltaprime and is proprietary information. The Non-Disclosure Agreement (NDA) in effect between Entersoft and Deltaprime governs the disclosure of this report to all other parties including product vendors and suppliers.