QuillAudits

Audit Report
June, 2024

For

DATAMINT

Table of Content

Executive Summary

Project Name	Data Mint - GRANTFIN
Overview	datamint.ai is a platform specializing in providing advanced data management and analytics solutions tailored for the DeFi (Decentralized Finance) sector. It offers tools for integrating, visualizing, and analyzing large datasets to help DeFi projects and businesses make informed decisions, optimize operations, and leverage their data for strategic advantages in the blockchain ecosystem.
Timeline	22nd May 2024 - 29th May 2024
Updated Code Recieved	10th June 2024
Second Review	10th June 2024 - 11th June 2024
Method	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
Audit Scope	The scope of this audit was to analyse the DM Vault Contract for quality, security, and correctness.
Blockchain	EVM
Source Code	.sol file was provided
Contracts In-Scope	DMVault_flattened.sol
Branch	NA
Fixed In	.sol file was provided

Number of Security Issues per Severity

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	3	1
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	1	0	2

Checked Vulnerabilities

We have scanned the Solidity program 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
- ERC20 API violation
- 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

Techniques and Methods

Throughout the audit of DM Vault, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of various token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the Solana programs.

Structural Analysis

In this step, we have analysed the design patterns and structure of Solana programs. A thorough check was done to ensure the Solana program is structured in a way that will not result in future problems.

Static Analysis

Static analysis of Solana programs was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of Solana programs.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, and their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behaviour of Solana programs in production. Checks were done to know how much gas gets consumed and the possibilities of optimising code to reduce gas consumption.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four severity levels, each of which has been explained below.

High Severity Issues

A high severity issue or vulnerability means your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impacts and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These issues were identified in the initial audit and successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Medium Severity Issues

1. Handling Non-Standard ERC20 Tokens and Transfer Failures

Path

src/DMVault.sol

Description

The contract does not correctly handle non-standard ERC20 tokens, which can lead to failed transfers or unexpected behavior. Additionally, transfer operations do not ensure the success of the transactions, leading to potential issues if transfers fail. Specific areas requiring attention include:

1. **withdrawERC20**: Needs to use OpenZeppelin's SafeERC20 to handle non-standard ERC20 tokens.
2. **withdrawAllERC20**: Should ensure the success of transfers and handle failures gracefully.
3. **withdrawEtherStableVolatile**: Lacks checks to ensure transfer success, potentially leading to unexpected behavior if transfers fail.

Impact

Failing to handle non-standard ERC20 tokens correctly can cause transaction failures and potential loss of funds. Additionally, not ensuring transfer success can lead to unexpected contract behavior and potential vulnerabilities.

Recommendation

Use OpenZeppelin's SafeERC20 library to handle non-standard ERC20 tokens and ensure successful token transfers. Implement checks to confirm transfer success and handle failures gracefully. Incorporate comprehensive token handling mechanisms to support a wide range of ERC20 token standards and ensure robust error handling for all transfer operations.

Status

Resolved

Low Severity Issues

1. Potential Loss of Administrative Access

Path

src/DMVault.sol

Description

The contract contains functions to remove admins and callers, but lacks checks to ensure that at least one admin or caller always remains. Specifically, the `removeAdmins` and `removeCallers` functions can be called in a way that removes all admins or callers, leaving the contract without any authorized personnel to perform critical functions.

Impact

Removing all admins or callers can leave the contract in a state where no authorized personnel can manage or interact with it. This can lead to loss of control over the contract, inability to perform necessary administrative tasks, and potential security risks if no authorized users remain.

Functions Affected

1. `removeAdmins‘
2. `addCallers’
3. `removeCallers‘

Recommendation

Implement checks in the `removeAdmins` and `removeCallers` functions to ensure that at least one admin and one caller always remains. Prevent the removal operation if it would result in zero admins or callers.

Status

Acknowledged

2. Use Low-Level Call to Prevent Gas Griefing Attacks When Returned Data Not Required

Description

Using `call()` when the returned data is not required unnecessarily exposes the contract to gas griefing attacks from huge returned data payloads. For example:

```
(bool sentToUser, ) = recipient.call{ value: finalUserAmount }("");
require(sentToUser, "Failed to send Ether");
```

is effectively the same as writing:

```
(bool sentToUser, bytes memory data) = recipient.call{ value: finalUserAmount }("");
require(sentToUser, "Failed to send Ether");
```

In both cases, the returned data is copied into memory, potentially leading to gas griefing attacks even though the returned data is not utilized.
Instances of potential gas griefing attacks can be found in the following functions:

1. withdrawEther
2. withdrawAllEther
3. withdrawEtherStableVolatile

Remediation

Use a low-level call without handling returned data when it is not required.

For example:

```
bool sent;
assembly {
  sent := call(gas(), recipient, finalUserAmount, 0, 0, 0, 0)
}
if (!sent) revert Unauthorized();
```

Consider using libraries like ExcessivelySafeCall to handle such scenarios safely.

Status

Acknowledged

3. Lack of Comprehensive Input Validation for Parameters and Addresses

Path

src/DMVault.sol

Description

The contract lacks comprehensive input validation for parameters and addresses in various functions. This can lead to invalid or zero addresses being used, resulting in unexpected behavior or potential vulnerabilities. Additionally, parameters like `_stableCap` should be greater than zero and `feeTier` should not be zero. Specific areas lacking input validation include:

1. Constructor
2. setStableCap
3. setPool
4. addAdmins
5. removeAdmins
6. addCallers
7. removeCallers

Impact

The absence of these validations can lead to invalid or zero addresses being used, resulting in unexpected behavior or potential vulnerabilities.

Recommendation

Implement validation checks to ensure addresses are non-zero and parameters like `_stableCap` and `feeTier` are within reasonable ranges and greater than zero. Incorporate comprehensive validation mechanisms for all sensitive inputs throughout the contract.

Status

Acknowledged

Informational Issues

1. Missing Event Emissions in Functions

Path

src/DMVault.sol

Description

The contract lacks event emissions in several important functions, which are essential for providing transparency and traceability of important actions and state changes. Specifically, the following functions do not emit events:

- `setStableCap`
- `setPool`
- `convertToWETH`
- `withdrawERC20`
- `withdrawAllERC20`
- `withdrawEther`
- `withdrawAllEther`
- `withdrawEtherStableVolatile`
- `closePositionAndWithdrawAll`
- `addAdmins`
- `removeAdmins`
- `addCallers`
- `removeCallers`
- `openPosition`
- `closePosition`
- `closePositionAndExitToStable`
- `performRebalance`

Impact

The absence of events makes it difficult to track and audit critical state changes, reducing transparency and making it harder to debug issues or investigate incidents. Event emissions are essential for providing a clear record of significant actions and state changes within the contract.

Recommendation

Add event emissions for all important functions to ensure that all significant actions and state changes are logged. Implement a comprehensive event logging system throughout the contract to enhance transparency and facilitate better tracking and auditing of all actions and state changes.

Status

Acknowledged

2. Unnecessary Initialization of Variables with Default Values

Description

Initializing variables with default values is unnecessary and can be optimized. For example, in the `DMVault` contract, the variable `positionLiquidity` is initialized with a default value of `0`. Instances of unnecessary initialization in the DMVault contract:

lowerActiveTick = 0;
upperActiveTick = 0;
positionLiquidity = 0;
positionLiquidity = 0;
baseInStable = 0
I = 0

Recommendation

Avoid initializing variables with default values

Status

Resolved

3. Unused Function `_doExactOutputSwap ‘

Path

src/DMVault.sol

Description

The contract contains an unused function `_doExactOutputSwap ’. This function is designed to perform an exact output swap using the Uniswap V3 router, but it is never called within the contract.

Recommendation

Remove the unused `_doExactOutputSwap ‘ function if it has no use from the contract to reduce complexity and eliminate unnecessary code.

Status

Resolved

Closing Summary

In this report, we have considered the security of the Data Mint's DMVault Contract. We performed our audit according to the procedure described above.

Some issues of High, Medium, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Data Mint's DMVault smart contract. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of Data Mint's DMVault smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the Data Mint's DMVault to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

1000+
Audits Completed

$30B
Secured

1M+
Lines of Code Audited

Follow Our Journey

Audit Report
June, 2024

For

DATAMINT

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com