hashlock.

2024

Security Audit

Shezmu (CDP)

October 2024 info@hashlock.com.au Hashlock Pty Ltd

Table of Contents

CAUTION

THIS DOCUMENT IS A SECURITY AUDIT REPORT AND MAY CONTAIN CONFIDENTIAL INFORMATION. THIS INCLUDES IDENTIFIED VULNERABILITIES AND MALICIOUS CODE WHICH COULD BE USED TO COMPROMISE THE PROJECT. THIS DOCUMENT SHOULD ONLY BE FOR INTERNAL USE UNTIL ISSUES ARE RESOLVED. ONCE VULNERABILITIES ARE REMEDIATED, THIS REPORT CAN BE MADE PUBLIC. THE CONTENT OF THIS REPORT IS OWNED BY HASHLOCK PTY LTD FOR USE OF THE CLIENT.

Executive Summary

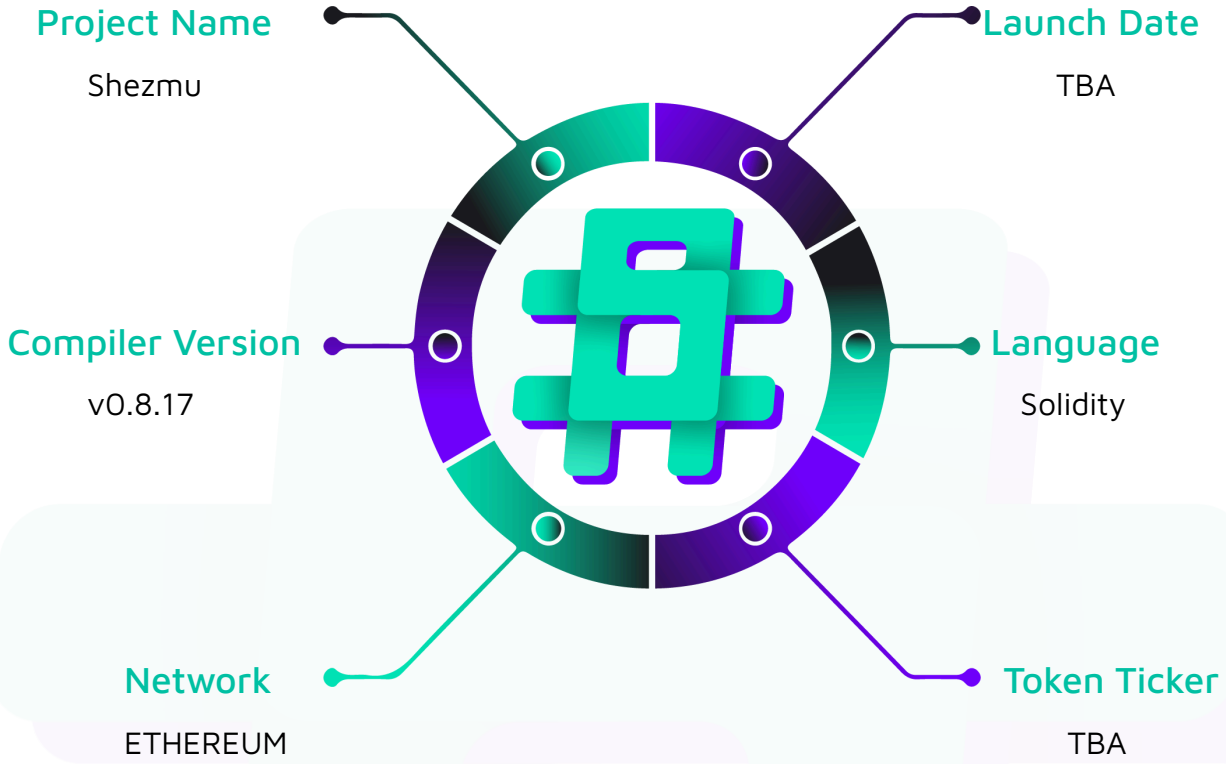
The Shezmu team partnered with Hashlock to conduct a security audit of their smart contracts. Hashlock manually and proactively reviewed the code in order to ensure the project's team and community that the deployed contracts are secure.

Project Context

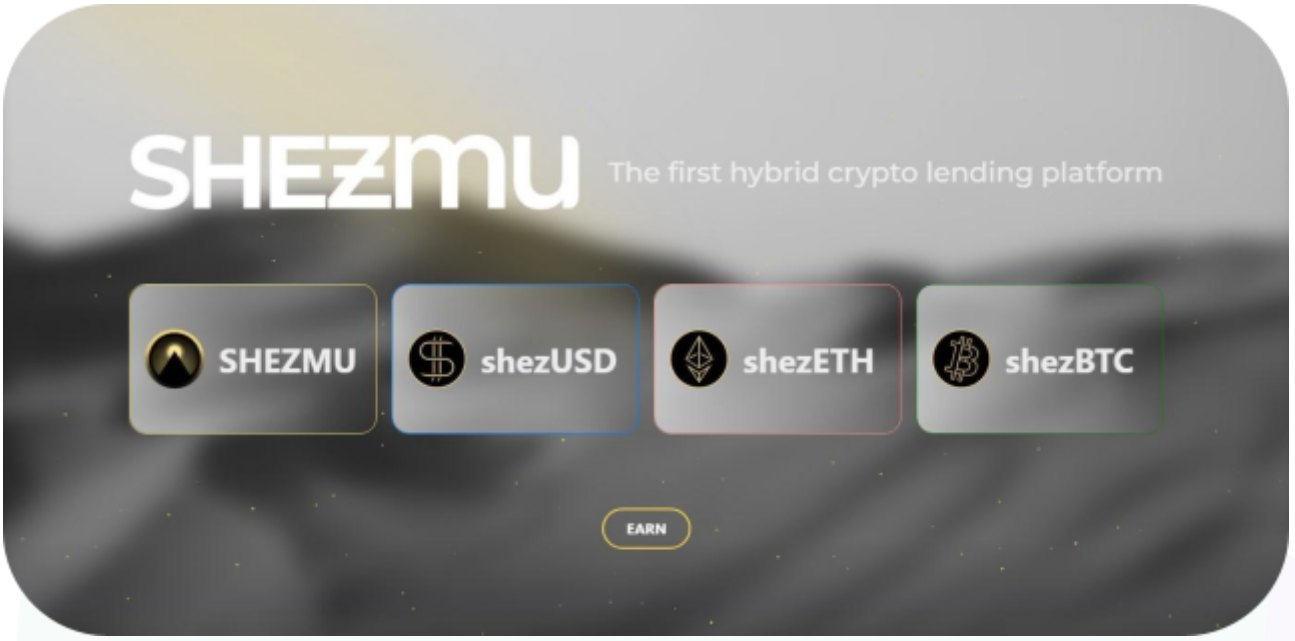
Shezmu introduces a groundbreaking hybrid Collateralized Debt Position (CDP) platform that innovatively combines the capabilities of both NFTs and Yield-Bearing Tokens. The platform allows users to borrow against both NFTs and Yield-Bearing Tokens, providing unparalleled flexibility and liquidity in the digital asset space. In addition to the core CDP functionality, the project offers a suite of utilities designed to enhance user experience and asset value.

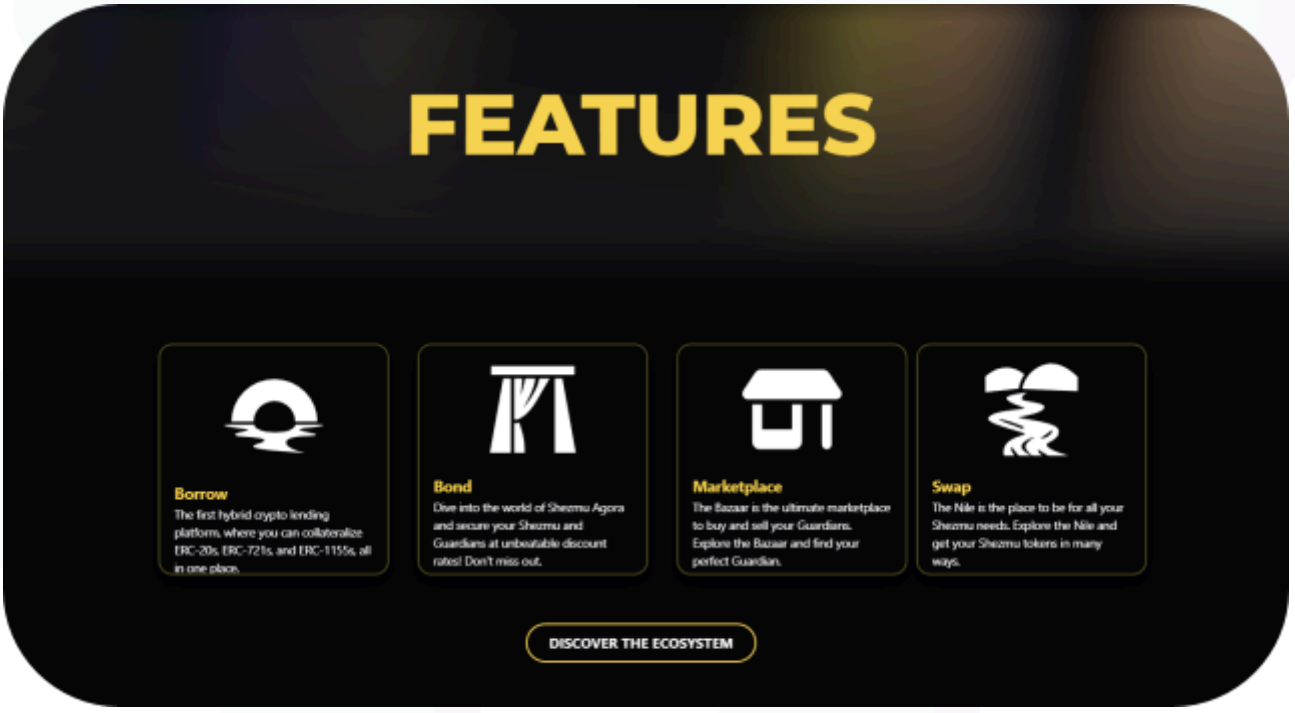
Project Name: Shezmu
Compiler Version: 0.8.17
Website: www.shezmu.io
Logo:

Visualised Context:



Project Visuals:





Audit scope

We at Hashlock audited the solidity code within the Shezmu project, the scope of work included a comprehensive review of the smart contracts listed below. We tested the smart contracts to check for their security and efficiency. These tests were undertaken primarily through manual line-by-line analysis and were supported by software-assisted testing.

Description	**Shezmu Smart Contracts**
Platform	**Ethereum / Solidity**
Audit Date	**October, 2024**
Folder 1	Farming
Folder 1 Commit Hash	b22956a844e1784fb0242159e57e4639fb9d301d
Folder 2	Marketplace
Folder 2 Commit Hash	b22956a844e1784fb0242159e57e4639fb9d301d
Folder 3	Oasis
Folder 3 Commit Hash	b22956a844e1784fb0242159e57e4639fb9d301d
Folder 4	Oracle
Folder 4 Commit Hash	b22956a844e1784fb0242159e57e4639fb9d301d
Folder 5	Sale
Folder 5 Commit Hash	b22956a844e1784fb0242159e57e4639fb9d301d
Folder 6	Staking
Folder 6 Commit Hash	b22956a844e1784fb0242159e57e4639fb9d301d
Folder 7	Token
Folder 7 Commit Hash	b22956a844e1784fb0242159e57e4639fb9d301d
Folder 8	Utils
Folder 8 Commit Hash	b22956a844e1784fb0242159e57e4639fb9d301d

Security Rating

After Hashlock's Audit, we found the smart contracts to be **"Secure"**. The contracts all follow simple logic, with correct and detailed ordering. They use a series of interfaces, and the protocol uses a list of Open Zeppelin contracts. We initially identified some significant vulnerabilities that have since been addressed.

The 'Hashlocked' rating is reserved for projects that ensure ongoing security via bug bounty programs or on chain monitoring technology.

All issues uncovered during automated and manual analysis were meticulously reviewed and applicable vulnerabilities are presented in the Audit Findings section. The general security overview is presented in the Standardised Checks section and the project's contract functionality is presented in the Intended Smart Contract Functions section.

All vulnerabilities initially identified have now been resolved and acknowledged.

Hashlock found:

6 High severity vulnerabilities

3 Medium severity vulnerabilities

7 Low severity vulnerabilities

4 QAs

2 Gas Optimisations

Caution: *Hashlock's audits do not guarantee a project's success or ethics, and are not liable or responsible for security. Always conduct independent research about any project before interacting.*

Intended Smart Contract Functions

Claimed Behaviour	Actual Behaviour
Farming A robust farming and liquidity provision system allowing users to stake tokens, earn rewards, and manage liquidity in trading pairs.	**Contract achieves this functionality.**
Marketplace A versatile NFT marketplace supporting both ERC721 and ERC1155 tokens, with auction and direct sale functionalities, discounts, taxes, and escrow mechanisms.	**Contract achieves this functionality.**
Oasis A suite of DeFi products including lending vaults for various asset types, stablecoin implementations (ShezmuUSD, ShezmuETH, ShezmuBTC), liquidation mechanisms, auction systems, stability pools, and yield farming strategies.	**Contract achieves this functionality.**
Oracle A sophisticated price oracle system aggregating data from multiple sources like Chainlink and Uniswap V2 to provide accurate and up-to-date asset pricing for the ecosystem.	**Contract achieves this functionality.**
Sale A token distribution system featuring bonding mechanisms, public sales, and liquidity provision on Uniswap, with customizable discounts and vesting periods.	**Contract achieves this functionality.**

Staking A flexible staking system with multiple lock periods, rewards distribution, and version migration, offering users various options to maximise their returns.	**Contract achieves this functionality.**
Token A multi-faceted token system featuring the main Shezmu token with tax and liquidity mechanisms, a Guardian staking and rewards system, an Obelisk multiplier system, and an Option token for discounted purchases.	**Contract achieves this functionality.**
Utils A collection of utility functions and libraries providing essential support for access control, mathematical operations, and rate calculations across the entire ecosystem.	**Contract achieves this functionality.**

Code Quality

This audit scope involves the Shezmu project's smart contracts, as outlined in the Audit Scope section. All contracts, libraries, and interfaces mostly follow standard best practices to help avoid unnecessary complexity that increases the likelihood of exploitation; however, some refactoring was required.

The code is very well commented on and closely follows best practice nat-spec styling. All comments are correctly aligned with code functionality.

Audit Resources

We were given the Shezmu project's smart contract code in the form of GitHub access.

As mentioned above, code parts are well-commented. The logic is straightforward, and therefore it is easy to quickly comprehend the programming flow as well as the complex code logic. The comments are helpful in providing an understanding of the protocol's overall architecture.

Dependencies

As per our observation, the libraries used in this smart contracts infrastructure are based on well-known industry-standard open source projects.

Severity Definitions

Significance	Description
High	High-severity vulnerabilities can result in loss of funds, asset loss, access denial, and other critical issues that will result in the direct loss of funds and control by the owners and community.
Medium	Medium-level difficulties should be solved before deployment, but won't result in loss of funds.
Low	Low-level vulnerabilities are areas that lack best practices that may cause small complications in the future.
QA	Quality Assurance (QA) findings are purely informational and don't impact functionality. These notes help clients improve the clarity, maintainability, or overall structure of the code, ensuring a cleaner and more efficient project. They should be addressed for optimization but are not critical to the system's performance or security.
Gas	Gas Optimisations, issues, and inefficiencies

Audit Findings

High

[H-01] oasis/migration/CollateralUnlockerUSD#unlockCollateral - Incorrect token convert

Description

The `unlockCollateral` function is intended to convert users `USDC` to `shezUSD` but the current implementation works opposite by passing the parameters with an incorrect order to the `curveStableSwapNG.exchange()`.

Vulnerability Details

In the `curveStableSwapNG` contract, the `exchange` function passes the input token address as the first parameter and the output token address as the second parameter.

However, the collateral of the index 0 is `shezUSD` in the `curveStableSwapNG` contract

```
function unlockCollateral(address vault) external {
    ...
    curveStableSwapNG.exchange(int128(0), int128(1), SWAP_AMOUNT, 0);
}
```

It means the current implementation converts users `shezUSD` to `USDC`.

Impact

The function execution will always fail.

Recommendation

Replace the first parameter value and the second parameter value passed.

Status

Resolved

[H-02] oasis/stable/ShezmuStablecoin#migrate - Lack of burning old stablecoins

Description

The `migrate` function is intended to mint new stablecoins equal to the amount of old stablecoins held by users. However, the function is missing to burn the old stablecoins of the caller.

Vulnerability Details

The function mint new stablecoins equal to the amount of old stablecoins held by users but it doesn't handle the old stablecoins at all.

```
function migrate() external {
    require(
        !isBlacklisted[_msgSender()],
        'You are not whitelisted to migrate'
    );
    uint256 balance = oldStablecoin.balanceOf(msg.sender);
    require(balance > 0, 'You have no old stablecoin to migrate');
    _mint(msg.sender, balance);
}
```

Impact

This allows anyone to mint an unlimited amount of new stablecoins by calling the function unlimited times.

Recommendation

Burn the old stablecoins held by the caller in the `migrate` function.

Status

Resolved

[H-03] oasis/stablecoin/ShezmuStablecoin#migrate - Lack of unpausing before burning

Description

The old stablecoin will be kept paused forever. However, the `migrate` function is missing to unpause the old stablecoin before calling the `burn` function. The `burn` function internally calls the `_update` function and the `_update` function has a `whenNotPaused` modifier which allows it to be called only when the contract is not paused.

Recommendation

Unpause the old stablecoin before calling the `burn` function, and pause again after calling.

Status

Resolved

[H-04] oasis/migration/BalancerLpMigration#migrate - Call to the incorrect contract

Description

In the `BalancerLpMigration` contract, the `migrate` function calls the old stablecoins `migrate` function, which doesn't exist. This causes the `migrate` function to always fail.

Vulnerability Details

The old stablecoins `migrate` function which doesn't exist is called.

```
function migrate() external {
    ...
    oldStablecoin.migrate();
}
```

Impact

The `migrate` function always failed.

Recommendation

Update the old stablecoins migrate function with the new stablecoins migrate function.

Status

Resolved

[H-05] oasis/migration/BalancerLpMigration#migrate - Lack of access permission check for the migrate() function

Description

In the `BalancerLpMigration` contract, the `migrate` function is missing the access permission check and it allows blacklisted users holding LPs to migrate new stablecoins and then get new LPs.

Recommendation

Add a blacklisting functionality in the BalancerLpMigration contract like the functionality of the new stablecoin.

Status

Resolved

[H-06] farming/Farming - Potential Denial of Service (DOS) attack in receiveETHForDividends and massUpdatePools functions

Description

The `receiveETHForDividends` and `massUpdatePools` functions are designed to handle all the pools stored in the `poolInfo` array.

However, this design can lead to a potential Denial of Service (DOS) attack if the `poolInfo` array is too large.

Vulnerability Details

The functions iterate over all pools in the array, performing multiple calls and state changes for each pool. If the array size exceeds the block gas limit, the transaction will always fail.

Recommendation

Add a maximum value for the number of pools to be added.

Status

Resolved

Medium

[M-01] oasis/btc/ERC20ValueProviderBTC#getPriceBTC, oasis/btc /ERC1155ValueProviderBTC#getFloorBTC,ERC20ValueProviderETH#getPrice ETH,oasis/eth/ERC1155ValueProviderETH#getFloorETH,asis/usd/ERC20Valu eProvider#getPriceUSD,oasis/usd/ERC721ValueProvider#getFloorETH, oasis/usd/ERC1155ValueProvider#getFloorUSD - Lack of time based checks

Description

The functions retrieve price data from Chainlink oracles but do not implement any checks to ensure the data's freshness. This oversight could lead to the use of outdated price information, potentially resulting in incorrect valuations, mispriced transactions, or vulnerability to market manipulation.

Impact

All calculations are performed based on the old data.

Recommendation

Implement a time-based check to ensure the Oracle data is recent.

This can be done by:

- Defining a maximum acceptable age for the data (e.g., 1 hour).
- Comparing the current block timestamp with the timestamp returned by the oracle.
- Reverting the transaction if the data is too old.

Status

Resolved

[M-02] oasis/usd/ERC20Vault, oasis/usd/ERC721Vault, oasis/usd/ERC1155Vault, marketplace/ERC721Marketplace, marketplace/ERC1155Marketplace, oasis/liquidations/ERC1155/ERC1155ShezmuAuction - Anyone can initialise the implementation contracts

Description

All the above contracts are upgradeable smart contracts.

The implementation contracts allow anyone to call the `initialize()` function due to the absence of a `_disableInitializers()` call in the `constructor`.

Impact

Anyone can take control of the implementation contracts.

Recommendation

It's recommended to add `oz-upgrades-unsafe-allow constructor`.

```
/// @custom:oz-upgrades-unsafe-allow constructor

constructor() {

    _disableInitializers();

}
```

Status

Resolved

[M-03] oasis/strategy/BaseStrategy - Incomplete implementation of deposit function

Description

The `deposit` function in the `BaseStrategy` contract is incomplete. It transfers tokens from the user to the contract but fails to update critical state variables, including the user's balance and the total staked amount. This omission can lead to incorrect accounting of user stakes and rewards, potentially causing significant issues in the contract's functionality.

Recommendation

Modify the `deposit` function to properly update all relevant state variables:

Status

Resolved

Low

[L-01] oasis/usd/ERC721Vault#setSettings, oasis/usd/ERC1155Vault#setValueProvider, oasis/usd/ERC721ValueProvider#disableFloorOverride, token/Shezmu#setSwapTaxSettings, token/Obelisk#setAddressProvider, setInfo, setThreshold, Guardian#setAddressProvider, setMintLimit, setTreasury, setFee, setZap, setLpFeeReceiver, setPricePerGuardian, setWhitelist, setWhitelist, setRewardRate, addFeeTokens, removeFeeTokens, sale/PublicSale#setTotalAmount, setTimestamp, setLimitEth, oasis/migration/CollateralUnlockerUSD#setSwapAmount, setWhitelisted, staking/SingleStaking#setStakingV2, enableRelock, enableExtendPeriod, enableMoreStaking, staking/SingleStakingV2#setStakingV1, enableRelock, enableExtendPeriod, enableMoreStaking - Lack of emitting events

Description

The above setter functions are missing events when states are set.

Recommendation

Add meaningful events to the functions.

Status

Resolved

[L-02] oasis/migration/CollateralUnlockerUSD#unlockCollateral - Lack of the vault check

Description

The `unlockCollateral` function is missing a check if the vault input value is the correct one used by the protocol. In case that the contract has a balance of any tokens, lack of this check allows anyone to call the `unlockCollateral` function with a fake vault which has the same interface, and transfer the tokens.

Recommendation

Add a whitelisting functionality for vaults to be used.

Status

Resolved

[L-03] token/Shezmu#_swapTax - Lack of records of failure cases

Description

The `_swapTax` function internally calls the `guardianFeeReceiver.receiveReward` function but does not log a failure to call the internal function.

Recommendation

Add an event in the `catch` case.

Status

Resolved

[L-04] token/Shezmu#_swapTax, farming/Farming#_processRewards, farming/Zap#recoverETH - Lack of return value check

Description

There is an unchecked ETH transfer using the low-level call function. The return value of this call is not checked, which could lead to silent failures if the ETH transfer fails.

Recommendation

Add a check for the return value and if it's not true, revert the transaction.

Status

Resolved

[L-05] marketplace/ERC721Marketplace - Incorrect implementation of the `_getPremiumPrice` function

Description

the `_getPremiumPrice` function is implemented to always return 0, regardless of the input parameters. This implementation appears to be incorrect and may lead to unintended behaviour in the premium pricing mechanism of the marketplace.

Impact

- No Premium Pricing: The function always returns 0, effectively disabling any premium pricing mechanism in the marketplace.
- Potential Revenue Loss: If premium pricing was intended to generate additional revenue, this implementation results in lost income for the platform.
- Unfair Pricing: All items are treated equally without considering potential premium status, which may be unfair to sellers of high-value or rare items.
- Inconsistent Behaviour: The function signature suggests that premium pricing should depend on input parameters, but the current implementation ignores these inputs.

Recommendation

Review the intended logic for premium pricing and implement it correctly.

If premium pricing is not intended to be used, consider refactoring the code to eliminate any references to premium pricing or adding clear comments.

Status

Resolved

[L-06] marketplace/MarketplaceAuction#_withdrawBid - Inappropriate use of assert for external call results

Description

The `_withdrawBid` function uses the `assert` statement to check the result of an external call (ETH transfer). This is an inappropriate use of `assert` and can lead to severe consequences in case of transfer failure.

Recommendation

Replace the `assert` statement with a `require` statement and add proper error handling:

Status

Acknowledged

[L-07] marketplace/MarketplaceDiscount#_setOracle - Insufficient oracle validation

Description

The `_setOracle` function lacks proper validation for the oracle address. While it checks for non-zero addresses, it fails to verify if the provided oracle address actually implements the required `IMarketplaceOracle` interface. This oversight could lead to the setting of invalid oracles, potentially causing critical failures in price fetching operations.

Recommendation

Implement a validation function that checks if the provided oracle address is a contract and implements the required interface:

```
function _validateOracle(address oracle) internal view {
    require(oracle.code.length > 0, "Oracle must be a contract");
    require(IMarketplaceOracle(oracle).getFloorETH.selector ==
        bytes4(keccak256("getFloorETH()")), "Invalid oracle interface");
}
```

Status

Resolved

QA

[Q-01] oasis/usd/ERC20Vault#_liquidate, oasis/usd/ERC1155Vault#_liquidate - Unnecessary position.debtPortion update

Description

The `_liquidate` function updates `position.debtPortion` to 0 but it's not needed because `positions[_owner]` is deleted in the next line.

Recommendation

Remove the `position.debtPortion` update.

Status

Resolved

[Q-02] marketplace/ERC721Marketplace, marketplace/ERC1155Marketplace - Unused role declaration

Description

The contracts declare the `WHITELISTED_ROLE` but it's never used.

Recommendation

Remove the role declaration in the contracts.

Status

Resolved

[Q-03] farming/Farming#getPoolInfo - `365 * 86400` could be replaced with `365 days` for readability

Description

The `getPoolInfo` function uses `365 * 86400` while `apr` calculation.

`365 * 86400` could be replaced with `365 days` to improve code readability.

Recommendation

Replace `365 * 86400` with `365 days`.

Status

Resolved

[Q-04] liquidations/ERC20/ERC20Liquidator, liquidations/ERC721/ERC721Liquidator, liquidations/ERC1155/ERC1155Liquidator - Unused error declaration

Description

The contracts declare the `InsufficientBalance` errors but they are never used.

Recommendation

Remove the unused error or use the error in the `liquidate` function.

Status

Resolved

Gas

[G-01] oasis/migration/CollateralUnlockerUSD - stablecoin, usdc, and curveStableSwapNG variables could be made constant

Description

The above variables are only set with hardcoded values during the contract deployment.

Recommendation

Make such variables constant.

Status

Resolved

[G-02] oasis/migration/CollateralUnlockerUSD#unlockCollateral - Gas consumption due to reading storage multiple times

Description

The `unlockCollateral` function reads `SWAP_AMOUNT` from storage multiple times which costs more gas.

Recommendation

Cache `SWAP_AMOUNT` into a memory variable and use the variable instead of reading the storage multiple times.

Status

Resolved

Centralisation

The Shezmu project values security and utility over decentralisation.

The owner executable functions within the protocol increase security and functionality but depend highly on internal team responsibility.

Conclusion

After Hashlocks analysis, the Shezmu project seems to have a sound and well-tested code base, now that our vulnerability findings have been resolved and acknowledged. Overall, most of the code is correctly ordered and follows industry best practices. The code is well commented on as well. To the best of our ability, Hashlock is not able to identify any further vulnerabilities.

Our Methodology

Hashlock strives to maintain a transparent working process and to make our audits a collaborative effort. The objective of our security audits is to improve the quality of systems and upcoming projects we review and to aim for sufficient remediation to help protect users and project leaders. Below is the methodology we use in our security audit process.

Manual Code Review:

In manually analysing all of the code, we seek to find any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behaviour when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our methodologies include manual code analysis, user interface interaction, and white box penetration testing. We consider the project's website, specifications, and whitepaper (if available) to attain a high-level understanding of what functionality the smart contract under review contains. We then communicate with the developers and founders to gain insight into their vision for the project. We install and deploy the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We undergo a robust, transparent process for analysing potential security vulnerabilities and seeing them through to successful remediation. When a potential issue is discovered, we immediately create an issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is vast because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with Acknowledged questions, and then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyse the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinised by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the contract details are made public.

Disclaimers

Hashlock's Disclaimer

Hashlock's team has analysed these smart contracts in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in the smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Hashlock is not responsible for the safety of any funds and is not in any way liable for the security of the project.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to attacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.

About Hashlock

Hashlock is an Australian-based company aiming to help facilitate the successful widespread adoption of distributed ledger technology. Our key services all have a focus on security, as well as projects that focus on streamlined adoption in the business sector.

Hashlock is excited to continue to grow its partnerships with developers and other web3-oriented companies to collaborate on secure innovation, helping businesses and decentralised entities alike.

Website: hashlock.com.au
Contact: info@hashlock.com.au