

SECURITY REVIEW OF SHEZMU (OASIS)



Summary

Auditors: 0xWeiss (Marc Weiss)

Client: Shezmu (Oasis)

Report Delivered: 13 March 2024

About 0xWeiss

0xWeiss is an independent security researcher. In-house auditor/security engineer in *Ambit Finance* and *Tapioca DAO*. Security Researcher at Paladin Blockchain Security and ASR at Spearbit DAO. Reach out on Twitter @[0xWeiss](#)

Protocol Summary

Shezmu is a cross-chain router dex and elastic-supply protocol with a unique and confident approach. Built with the goal of innovation and efficiency, Shezmu is redefining the DeFi blockchain landscape with a novel approach to maximizing rewards for its dedicated community.

Protocol Name	Shezmu (Oasis)
Language	Solidity
Codebase	https://github.com/Shezmu-Team/shezmu-contracts
Commit	09b2011757a87e42658818394be49e556ac1f303
Previous Audits	None

Audit Summary

Shezmu engaged **0xWeiss** to review the security of its codebase and consult architectural decisions.

A 2-week time-boxed security assessment was performed.

At the end, there were 25 issues identified.

All findings have been recorded in the following report. Notice that the examined smart contracts are not resistant to internal exploitation.

For a detailed understanding of risk severity, source code vulnerability, and potential attack vectors, refer to the complete audit report below.

Vulnerability Summary

Severity	Total	Pending	Acknowledged	Par. resolved	Resolved
● HIGH	6	0	0	0	6
● MEDIUM	5	0	0	0	5
● LOW	12	0	2	1	9
● INF	2	0	0	0	2

Severity Classification

Severity	Classification
● HIGH	Exploitable, causing loss/manipulation of assets or data.
● MEDIUM	Risk of future exploits that may or may not impact the smart contract execution.
● LOW	Minor code errors that may or may not impact the smart contract execution.
● INF	No impact issues. Code improvement

Methodology

























The auditing process pays special attention to the following considerations:


- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

AUDIT SCOPE

src/usd/ShezmuUSD.sol
src/usd/ERC721Vault.sol
src/usd/ERC1155ValueProvider.sol
src/usd/ERC20Vault.sol
src/usd/ERC20ValueProvider.sol
src/usd/ERC721ValueProvider.sol
src/usd/ERC1155Vault.sol
src/stability/ShezUSDStabilityPool.sol
src/utils/RateLib.sol
src/utils/AccessControlUpgradeable.sol
src/liquidations/ERC1155/ERC1155ShezmuAuction.sol
src/liquidations/ERC1155/ERC1155Liquidator.sol
src/liquidations/ERC721/ERC721ShezmuAuction.sol
src/liquidations/ERC721/ERC721Liquidator.sol

Findings and Resolutions

ID	Category	Severity	Status
SP-H1	Rate Manipulation	 HIGH	Resolved
ERC721V -H1	Incorrect accounting	 HIGH	Resolved
GLOBAL-H1	Access control	 HIGH	Resolved
GLOBAL -H2	Access control	 HIGH	Resolved
GLOBAL -H3	User Loss	 HIGH	Resolved
APD-H3	User Loss	 HIGH	Resolved
SP-M1	Rate manipulation	 MEDIUM	Resolved
SP-M2	Architectural error	 MEDIUM	Resolved
GLOBAL-M1	Incorrect accounting	 MEDIUM	Resolved
GLOBAL-M2	Architectural error	 MEDIUM	Resolved
GLOBAL-M3	Architectural error	 MEDIUM	Resolved
ERC721V-L1	Logic	 LOW	Resolved
ERC721V -L2	Un-used code	 LOW	Resolved
GLOBAL-L1	Logic	 LOW	Resolved
GLOBAL-L2	Re-entrancy	 LOW	Resolved
GLOBAL-L3	Architectural error	 LOW	Part. Resolved
GLOBAL -L4	Missing events	 LOW	Resolved
GLOBAL -L5	Logic	 LOW	Resolved
GLOBAL -L6	Input validation	 LOW	Resolved
GLOBAL -L7	Un-used errors	 LOW	Resolved
GLOBAL -L8	Architectural error	 LOW	Acknowledged
GLOBAL -L9	Logic	 LOW	Resolved
GLOBAL -L10	Logic	 LOW	Acknowledged
GLOBAL - INF1	Natspec	 INF	Resolved

GLOBAL- INF1	Gas	 INF	Resolved
-----------------	-----	--	----------

[SP-H1] First depositor attack in Stability Pools

Severity	Category	Status
● HIGH	Rate Manipulation	Resolved

Description

The way on how the Stability pools has been designed is vulnerable to first depositor attacks:

```
function deposit(uint256 amount) external nonReentrant {  
    uint256 share = amountToShare(amount);  
    shezUSD.safeTransferFrom(msg.sender, address(this), amount);  
    _mint(msg.sender, share);  
}
```

- A hacker back-runs the transaction of an ERC4626 pool creation.
- The hacker mints for themselves one share: `deposit(1)`. Thus, `totalAsset()==1`, `totalSupply()==1`.
- The hacker front-runs the deposit of the victim who wants to deposit 20,000 USDT (20,000.000000).
- The hacker inflates the denominator right in front of the victim: `asset.transfer(20_000e6)`. Now `totalAsset()==20_000e6 + 1`, `totalSupply()==1`.
- Next, the victim's tx takes place. The victim gets $1 * 20_000e6 / (20_000e6 + 1) == 0$ shares. The victim gets zero shares.
- The hacker burns their share and gets all the money.

Recommendation

Implement the `uniswapv2` option of DEAD Shares. Though it does not fully mitigate the issue, I would advise the protocol to be the first depositor themselves, and deposit through flashbots (private mempool). Monitor also if another address has deposited previously.

Resolution

Fixed

[ERC721V-H1] Missing debt accrual when repurchasing NFTs.

Severity	Category	Status
● HIGH	Incorrect accounting	Resolved

Description

When calling `_repurchase` on the ERC721Vault contract, `totalDebtPortion` and `totalDebtAmount` are used to re-calculate the global debt of the protocol:

```
uint256 _totalDebtPortion = totalDebtPortion;
uint256 _totalDebtAmount = totalDebtAmount;
uint256 _plusPortion = _calculatePortion(
    _totalDebtPortion,
    _newDebtAmount,
    _totalDebtAmount
);
```

This will be incorrect and will cause all further debt calculations to be inaccurate too.

Recommendation

Do call the `accrue()` function before repurchasing:

```
function repurchase(
    uint256 _nftIndex,
    uint256 _repayAmount
) external nonReentrant {
+   accrue();
    _repurchase(msg.sender, _nftIndex, _repayAmount);
}
```

Resolution

Fixed

[GLOBAL-H1] Missing access control on newAuction.

Severity	Category	Status
● HIGH	Access control	Resolved

Description

newAuction is a function made so that whitelisted addresses can create auctions post liquidation to auction their erc721/1155 tokens. Though the modifier in *newAuction* is missing and anyone can create an auction.

Recommendation

Add the access control to the function:

```
- function newAuction(address _owner,ERC1155Upgradeable _nft,uint256 _id
x,uint256 _amount,uint256 _minBid) external{
+ function newAuction(address _owner,ERC1155Upgradeable _nft,uint256 _id
x,uint256 _amount,uint256 _minBid) external onlyRole(WHITELISTED_ROLE) {
```

Resolution

Fixed

[GLOBAL-H2] Ownership can be taken over by anyone.

Severity	Category	Status
● HIGH	Access control	Resolved

Description

The function *finalizeUpgrade* doesn't make sense as it only grants a role to the contract, changes the *auctionDuration* and updates the admin role. These 3 functionalities can be performed independently. However, the problem is not the lack of functionality of the function, it is that the function.

```
function finalizeUpgrade(
    address _admin,
    uint256 _auctionDuration
) external {
    bytes32 _role = keccak256('UPGRADED');
    if (hasRole(_role, address(this))) revert();

    auctionDuration = _auctionDuration;

    _grantRole(_role, address(this));
    _grantRole(DEFAULT_ADMIN_ROLE, _admin);
}
```

has no access control or whatsoever, allowing anyone to be the *DEFAULT_ADMIN_ROLE* and take over the contract.

Recommendation

Add the access control to the function:

```
- function finalizeUpgrade( address _admin,uint256 _auctionDuration) external {
+ function finalizeUpgrade( address _admin,uint256 _auctionDuration) external onlyRole(DEFAULT_ADMIN_ROLE) {
```

Resolution

Fixed

[GLOBAL-H3] Unsold NFTs will not be retrievable by creators.

Severity	Category	Status
 HIGH	User loss	Resolved

Description

Currently, in both auctions, if an NFT is unsold after the auction duration has ended, there is a function that supposedly should allow the creator of the auction (the previous owner of the NFT) to recover their NFT:

```
function withdrawUnsoldNFT(  
    uint256 _auctionIndex  
    ) external onlyRole(DEFAULT_ADMIN_ROLE) {
```

As the function has the *DEFAULT_ADMIN_ROLE* role and the check, the NFT will be able to be withdrawn only by the *DEFAULT_ADMIN_ROLE* and not the actual creator of the auction.

Recommendation

Add a check so that the auction creator is the one that is able to withdraw their NFT to the *withdrawUnsoldNFT* functions accross the different ERC721-1155 Standards:

```
function withdrawUnsoldNFT(  
    uint256 _auctionIndex  
-    ) external onlyRole(DEFAULT_ADMIN_ROLE) {  
+    ) external {  
    Auction storage auction = auctions[_auctionIndex];  
+ if (auction.creator != msg.sender) revert Unauthorized();
```

Resolution

Fixed

[SP-M1] Non-standardized vault + front-runnable shares.

Severity	Category	Status
● MEDIUM	Rate manipulation	Resolved

Description

Currently, the *ShezUSDStabilityPool* contract uses a non-standard pool, instead of implementing the battletested *erc4626* standard. This is important to prevent rounding errors, inflation and front-running attacks. Additionally, Shezmu is using the *balanceOf()* instead of virtual shares.

Recommendation

Refactor the stability pool to integrated with the *erc4626* standard from OpenZeppelin that uses virtual shares. Additionally follow the correct rounding principles to round in favor of the protocol. You can find good information in the following [article](#).

Resolution

Fixed

[SP-M2] Not having reserves can cause withdrawals to fail.

Severity	Category	Status
● MEDIUM	Architectural error	Resolved

Description

Currently, the *ShezUSDStabilityPool* contract has a *borrowForLiquidation* function that is used to borrow *shezUSD* to liquidate. However, there is no reserve of funds that have to stay in the contract at all times (idle) to cover possible withdrawals, causing them to fail if enough funds are borrowed.

Recommendation

Add a reserve variable that at least those funds can't be borrowed to count for possible withdrawals.

Resolution

Fixed

[GLOBAL-M1] Not accruing the vaults before updating debtInterestApr will miss-calculate the additionalInterest.

Severity	Category	Status
● MEDIUM	Incorrect accounting	Resolved

Description

Not accruing the vaults before updating debtInterestApr will mis-calculate the additionalInterest. This will impact future calculations across the codebase, impacting the borrowing and repaying functions.

The liabilities and market state should be accrued at the current liabilities before updating the interest apr, as the liabilities that should already be accrued are going to be accrued with the new apr, which would be incorrect.

Recommendation

Add the accrue call when updating the settings.

```
function setSettings(  
    VaultSettings calldata _settings  
) external onlyRole(SETTER_ROLE) {  
+     accrue();  
    settings = _settings;  
}
```

Resolution

Fixed

[GLOBAL-M2] Dust positions can be difficult to liquidate.

Severity	Category	Status
● MEDIUM	Architectural error	Resolved

Description

A malicious user can borrow amounts only worth a few wei.

This would affect the liquidators, as the gas price of liquidating the position can be larger than the position itself.

Same happens when repaying. A user can repay almost all their liabilities and leave a dust amount, so it becomes hard to liquidate too.

Recommendation

Add a minimum amount that can be borrowed.

```
function _borrow(address _account, uint256 _amount) internal {  
-     if (_amount == 0) revert InvalidAmount(_amount);  
+     if (_amount < minBorrowableAmount) revert InvalidAmount(_amount);  
}
```

Add another check when repaying, so that you can't leave less than this minimum amount without being repaid. If so, you either repay all, or repay less.

Resolution

Fixed

[GLOBAL-M3] Any token can be auctioned.

Severity	Category	Status
● MEDIUM	Architectural error	Resolved

Description

Currently, on auctions there is no validation or whatsoever that the `erc1155s` and the `erc721s` being auctioned are any of the whitelisted assets that exist from the vaults.

Meaning, any NFT can be auctioned, even scam NFT contracts with custom logic can be used to revert transfers, re-enter, etc.

```
_nft.safeTransferFrom(msg.sender, address(this), _idx, _amount, '0x');  
  
emit NewAuction(auctionsLength - 1, _nft, _idx, _amount, _startTime);
```


Recommendation

Add a global whitelist system through a registry, where you will keep all the contracts stored there. Anything from Vaults, Liquidators, Auction contracts, and most importantly, assets “whitelisted” that can be used in the system.

Resolution

Fixed

[ERC721V-L1] Normalization of the repay amount.

Severity	Category	Status
 LOW	Logic	Resolved

Description

When repurchasing your NFT, if you are trying to repay more than your whole debt to repurchase, the transaction will revert instead of normalizing you specified amount to the actual amount to repay.

Recommendation

Assign the whole debt amount to the repay amount variable if the users are trying to repay more than what they should.

```
+ if (_repayAmount == 0) revert InvalidAmount(_repayAmount);  
- if (_repayAmount > _debtAmount || _repayAmount == 0) revert InvalidAmount(_repayAmount);  
+ if(_repayAmount > _debtAmount) _repayAmount = _debtAmount;
```

Resolution

Fixed

[ERC721V-L2] Un-used whitelist role

Severity	Category	Status
 LOW	Un-used code	Resolved

Description

The whitelist role on the erc721vault contract is unused.

```
bytes32 public constant WHITELISTED_ROLE = keccak256('WHITELISTED_ROLE');
```

Recommendation

Use the role in the create new auction function.

Resolution

Fixed

[GLOBAL-L1] Incorrect assignment allows to enter incorrect nft addresses.

Severity	Category	Status
● LOW	Logic	Resolved

Description

When adding NFT Vaults on the liquidators, the incorrect param is passed. Currently, it allows to specify an arbitrary NFT address, while it should be fetching the address from the corresponding vault.

Recommendation

Change the following line in the liquidators:


```
function addNFTVault(
    ERC1155Vault _nftVault,
    address _nft,
    uint256 _tokenIndex
) external onlyOwner {
    if (address(_nftVault) == address(0) || _nft == address(0))
        revert ZeroAddress();

    vaultInfo[_nftVault] = VaultInfo(
        IERC20Upgradeable(_nftVault.stablecoin()),
-       _nft,
+       IERC1155Upgradeable(_nftVault.tokenContract()),
        _tokenIndex
    );
}
```

Resolution

Fixed

[GLOBAL-L2] Event re-entrancy can be given.

Severity	Category	Status
 LOW	Re-entrancy	Resolved

Description

Across the entire codebase, the pattern, execute a external call and emit an event after, is given. This allows users to basically skip event emission by reentering on the external call:

```
(bool _sent, ) = payable(msg.sender).call{
    value: auction.bids[_highestBidder]
}('');
assert(_sent);

emit ETHClaimed(_auctionIndex);
```

Recommendation

Emit the events before all the external calls.

```
+     emit ETHClaimed(_auctionIndex);
(bool _sent, ) = payable(msg.sender).call{
    value: auction.bids[_highestBidder]
}('');
assert(_sent);

-     emit ETHClaimed(_auctionIndex);
```

Resolution

Fixed

[GLOBAL-L3] Inefficient architecture.

Severity	Category	Status
● LOW	Architectural error	Partially Resolved

Description

- Currently, there are 3 different vault contracts, one supporting erc20s, the second one, erc721s and the third one erc1155s. The problem is that these 3 contracts are extremely similar, though don't have a base contract, to keep composability and a more efficient architecture.
- ERC1155 Vaults are architected so that it exists one vault per every single erc1155 id. This can become problematic as there might be too many vaults in the future.

Recommendation

- Create an abstract base contract for the 3 vaults. Additionally, if needed, do the same for the liquidation contracts too.
- Refactor the erc1155 to include all the erc1155 ids.

Resolution

Partially Fixed. The abstract contract for the different vault standards has been implemented. The erc1155 refactoring, hasn't.

[GLOBAL-L4] Missing events in functions that update key state changes.

Severity	Category	Status
● LOW	Missing events	Resolved

Description

Across the whole codebase there are several functions that do change ownership and permissions, or they update key states of the codebase, but they miss events.

Some examples of functions missing events on the vaults:

setBaseLiquidationLimitRate() *setBaseCreditLimitRate()*

Recommendation

Grep for all the setter functions that change relevant state and if they do not emit an event, declare one.

As an example, on how to log the old and new value:

```
function setBaseCreditLimitRate(
    RateLib.Rate memory _baseCreditLimitRate
) external onlyRole(DEFAULT_ADMIN_ROLE) {
    _validateRateBelowOne(_baseCreditLimitRate);
+   emit newBaseCreditLimitRate(baseCreditLimitRate, _baseCreditLimitR
ate);
    baseCreditLimitRate = _baseCreditLimitRate;
}
```

Resolution

Fixed

[GLOBAL-L5] Users with no collateral can be added to the indexes.

Severity	Category	Status
● LOW	Logic	Resolved

Description

Users can specify 0 as the amount when adding collateral, and they will be added to the *userIndexes* mapping:

```
function _addCollateral(address _account, uint256 _colAmount) internal {
    tokenContract.safeTransferFrom(_account, address(this), _colAmount
);
    Position storage position = positions[_account];
    if (!userIndexes.contains(_account)) {
        userIndexes.add(_account);
    }
    position.collateral += _colAmount;
    emit CollateralAdded(_account, _colAmount);
}
```

Recommendation

Check that collateral is not 0

```
function _addCollateral(address _account, uint256 _colAmount) internal {
+   if (_colAmount == 0) revert InvalidAmount(_colAmount);
    tokenContract.safeTransferFrom(_account, address(this), _colAmount
);
}
```

Resolution

Fixed

[GLOBAL-L6] Missing validation when updating the settings.

Severity	Category	Status
● LOW	Input validation	Resolved

Description

When updating the setting in the vaults, when initializing the erc20vault, there is validation so that all the rates in settings are within the valid ranges.

Unfortunately, this validation is missing on the *setSettings* function.

Recommendation

Add the same validation you already have in the constructor:

```
function setSettings(
    VaultSettings calldata _settings
) external onlyRole(SETTER_ROLE) {
+   if (
+       !_settings.debtInterestApr.isValid() ||
+       !_settings.debtInterestApr.isBelowOne()
+   ) revert RateLib.InvalidRate();


+   if (
+       !_settings.organizationFeeRate.isValid() ||
+       !_settings.organizationFeeRate.isBelowOne()
+   ) revert RateLib.InvalidRate();

    settings = _settings;
}
```

Resolution

Fixed

[GLOBAL-L7] Un-used errors in vaults.

Severity	Category	Status
 LOW	Un-used errors	Resolved

Description

The errors *PositionLiquidated*, *Unauthorized*, *ZeroAddress* and *InvalidOracleResults* are un-used in the *erc20Vault* contract.

Recommendation

Delete the declaration of the un-used errors.

Resolution

Fixed

[GLOBAL-L8] Transfer-tax tokens are not supported.

Severity	Category	Status
 LOW	Architectural error	Acknowledged

Description

The whole architecture has several spots that will not work with transfer-tax tokens.
This will break the entire system.

Recommendation

Consider not using such tokens.

Resolution

Acknowledged, such tokens will not be used.

[GLOBAL-L9] yearly interest will be slightly inaccurate.

Severity	Category	Status
● LOW	Logic	Resolved

Description

When calculating the additional interest, we are dividing the numerator between 365 days, which it is inaccurate:

```
return
    (elapsedTime * totalDebt * settings.debtInterestApr.numerator) /
    settings.debtInterestApr.denominator /
    365 days;
```

Recommendation


The average length of the calendar year in days now becomes: $(3 \times 365 + 366)/4 = 365.25$ days.

```
return
    (elapsedTime * totalDebt * settings.debtInterestApr.numerator) /
    settings.debtInterestApr.denominator /
-   365 days;
+   365.25 days;
```

Resolution

Fixed

[GLOBAL-L10] Incorrect healthiness check.

Severity	Category	Status
 LOW	Logic	Acknowledged

Description

Currently, the `_removeCollateral` function allows users to remove collateral that they previously deposited in the contract. There is a missing check so that users are not able to “withdraw” enough collateral to leave their position liquidatable.

As you can see, there is currently a check so that the positions’ debt, after withdrawing the collateral is not smaller than the credit limit of that account:

```
uint256 _creditLimit = _getCreditLimit(
    _account,
    position.collateral - _colAmount
);

if (_debtAmount > _creditLimit) revert InsufficientCollateral();

position.collateral -= _colAmount;
```

Though the check should be a different one, as you should be checking that the position is healthy liquidation wise, instead of the credit limit.

Recommendation

Change the checks:

```
-     uint256 _creditLimit = _getCreditLimit(
-         _account,
-         position.collateral - _colAmount
-     );

-     if (_debtAmount > _creditLimit) revert InsufficientCollateral();
+     uint256 liquidationLimit = _getLiquidationLimit(
+         _account,
+         position.collateral - _colAmount
+     );
+     if (_debtAmount > liquidationLimit) revert InsufficientCollateral
+ ();

    position.collateral -= _colAmount;
```

Resolution

Acknowledged as credit limit should be stricter than liquidation limit.

[GLOBAL-INF1] Inconsistent compiler version declaration.

Severity	Category	Status
● INF	Logic	Resolved

Description

Across the different contracts from the codebase, the compiler version declaration is inconsistent.

For instance, in ERC1155Liquidator contract, the declared version is pragma solidity ^0.8.4; . However, in the other contracts in scope like ERC721Vault the declared version is pragma solidity 0.8.17.

Recommendation

Even though the compiler version will end up being the same, for consistency, change the declaration of all the contracts to be the exact same version.

Resolution

Fixed

[GLOBAL-INF2] Un-used imports across the codebase.

Severity	Category	Status
● INF	Un-used imports	Resolved

Description

- In *ERC1155Vault* and *ERC20Vault*, import `'../../interfaces/IChainLinkV3Aggregator.sol'`;
- In *ERC20ValueProvider*, import `'@openzeppelin/contracts-upgradeable/token/ERC20/Utils/SafeERC20Upgradeable.sol'`;
- In *ERC1155ShezmuAuction*, import `'@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol'`;

Recommendation

Remove those import declarations.

Resolution

Fixed

DISCLAIMER

This report is not, nor should be considered, an “endorsement” or “disapproval” of any project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Marc Weiss to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technology’s proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intended to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Blockchain technology and cryptographic assets present a high level of ongoing risk.

My position is that each company and individual are responsible for their own due diligence and continuous security. My goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze. Therefore, I do not guarantee the explicit security of the audited smart contract, regardless of the verdict.