SOLIDProof

Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY

jAssets

by BLKSWN PTE. LTD

AUDIT

SECURITY ASSESSMENT

06. February, 2025

SolidProof_io

@solidproof_io

Introduction

SolidProof.io is a brand of the officially registered company Future Visions Deutschland. We're mainly focused on Blocskchain Security, such as Smart Contract Audits and KYC verification for project teams.
Solidproof.io assesses potential security issues in the smart contracts implementations, reviews for potential inconsistencies between the code base and the whitepaper/documentation, and provides suggestions for improvement.

Disclaimer

SolidProof.io reports are not, nor should they be considered, an "endorsement" or "disapproval" of any particular project or team. These reports are not, nor should they be considered, an indication of the economics or value of any "product" or "asset" created by any team. SolidProof.io does not cover testing or auditing the integration with external contracts or services (such as Unicrypt, Uniswap, PancakeSwap, etc.).

SolidProof.io Audits do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analysed, nor do they provide any indication of the technology proprietors. SolidProof Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

SolidProof.io Reports represent an extensive auditing process intending 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. SolidProof's position is that each company and individual are responsible for their own due diligence and continuous security. SolidProof in no way claims any guarantee of the security or functionality of the technology we agree to analyse.

Project Overview

Summary

Project Name	JAssets
Website	https://docs.jellyverse.org/protocol-library/jassets-by- blkswn
About the project	jAssets is a collateralized debt platform. Users can lock up collateral (specific ERC20 tokens, selected by the governance) and issue jAssets (jUSD, jAAPL, jTSLA, etc.) to their own address and subsequently transfer those tokens to any other address. The individual collateralized debt positions are called Troves.
Chain	Sei
Language	Solidity
Codebase Link	Provided as files
Commit	N/A
Unit Tests	Provided

Social Medias

Telegram	N/A
Twitter	https://x.com/BLKSWN_STUDIO
Facebook	N/A
Instagram	N/A
Github	N/A
Reddit	N/A
Medium	N/A
Discord	N/A
Youtube	N/A
TikTok	N/A
LinkedIn	N/A

Audit Summary

Version	Delivery Date	Changelog
v1.0	09. October 2024	• Layout Project • Automated- /Manual-Security Testing • Summary
v1.2	06. February 2025	• Reaudit

Note - The following audit report presents a comprehensive security analysis of the smart contract utilized in the project that includes malicious outside manipulation of the contract's functions. This analysis did not include functional testing (or unit testing) of the contract/s logic. We cannot guarantee 100% logical correctness of the contract as we did not functionally test it. This includes internal calculations in the formulae used in the contract.

File Overview

The Team provided us with the files that should be tested in the security assessment. This audit covered the following files listed below with an SHA-1 Hash.

File Name	SHA-1 Hash
Interfaces/ITroveManager.sol	702b786e171f99bf09fe77c6834e53446224bedf
Interfaces/IStakingOperations.sol	f32aeaaf56d526d4e49ece517d8df19a4e6175da
Interfaces/ISwapPair.sol	94a9de80ddb320de30ab79cdce208c47bcada853
Interfaces/IPriceFeed.sol	5a0c08149fe505c0fcd14693354f37a8e9633d14
Interfaces/IReservePool.sol	eb3514c8ed051aaf291bce6d2596a2aca9ba7573
Interfaces/IDebtToken.sol	6d27466ecea0e60c543ad517fcaa87cac894b549
Interfaces/IHintHelpers.sol	8438af44ea5d587778c84d82e1fd630c1c452f35
Interfaces/ITokenManager.sol	2e6eca2cc29297d2aded1d112f2840d5c0a7da91
Interfaces/ISwapERC20.sol	4e1042ed17b6accb286e28d0a9ec3478f72135c6
Interfaces/IStakingVestingOperations.sol	ce190ca2a553fde6ef7c3b568ffcc6fe46f177af
Interfaces/ILiquidationOperations.sol	228989f81703b234a8bb38be5870bd66baa5fe3c
Interfaces/IAlternativePriceFeed.sol	26dd72276c87a8c1b42109617ed5aa5572c40d22
Interfaces/IBase.sol	2ac655da8cb4916b60baf2cd60b05bc81ce4368d
Interfaces/ICollSurplusPool.sol	fd1f3efd126c78105a63da4cf19aa7a93e12f9de
Interfaces/IRedemptionOperations.sol	fc54b960f8ac1f7a20aa4ab1bb06ddba07cfa12c
Interfaces/IStoragePool.sol	4de295007d809e31f13b0631612d14f542837aa7
Interfaces/IBorrowerOperations.sol	795f17ae16feb67d662cfea40e055dda6940db74
Interfaces/IBBase.sol	dfc08a12479214aa353ee2f61a32715d1d12c5c4
Interfaces/IDynamicFee.sol	843c66ad373d646588c2b36940e124e41fe0a48f
Interfaces/ISwapOperations.sol	7897ff56407c8090975e7736d64042c3989ea829
Interfaces/ISwapCallee.sol	c585b02cb4ad1c50e7d0a7ae238745738eaf1082
Interfaces/ISortedTroves.sol	111eb14edfe36ccb0fbd0ed841b5f82c3cea5ee0
PriceFeed.sol	6c38c9a7d5a4da8abc383f1f315ebd0b3b0545d0
LiquidationOperations.sol	fb48244351b30df7e7abb19f492fb9bf906d07c5
TroveManager.sol	aa3a9f083d1f9cbec72ea1775ce53da1c8f39e2c

DynamicFee.sol	9b1c9cbf409ca4f49132e8a4cab5847ecc23e070
BorrowerOperations.sol	a9740c710610519811cd2b8fe03ef93f067ea150
StoragePool.sol	9dd98b65d3b993ec313a6ac25fe8fcdf0f052e41
SwapPair.sol	db86b02982575fa0110e10487f5939f6b53fc4f6
AlternativePriceFeed.sol	0a05c2b27c3b73650253aecca24b85ec9e8c7751
CollSurplusPool.sol	d3dbbeab6dd4de5a1e75010734d90af14a944eda
Dependencies/LiquityMath.sol	fbd3b6171829b9b96965f1619a633db3750db761
Dependencies/UQ112x112.sol	b2e60c78033d52510fc07b8a0cab7467dfa14f79
Dependencies/LiquityBase.sol	347cb6fb1389c88c6b18fc67be98be030f912f52
Dependencies/TrustlessPermit.sol	544bfd7dee92b0551988dce226bba52f29c55887
Dependencies/IERC2612.sol	ebd49c8b06cac4ab0ec3da20bc4ea130d38fda45
Dependencies/CheckContract.sol	2daa3aaa82603cf31aa1b1c5eb804e5dc80897a6
Dependencies/IBalancerV2Pool.sol	157bb81262abdbd30ab4c9e6565421e602c99ba9
Dependencies/IBalancerV2Vault.sol	fb885e3bbe1c3eda124a3e03d057a7379f19d6ea
TokenManager.sol	683bfc34dd27094b3864066120eae278eef01e5a
SwapERC20.sol	dd4f2354ae64761a74f685cdf18816c53564eb4a
SwapOperations.sol	8fabf25aebb4337cc85567d6606bd6efb45ce854
StakingOperations.sol	035e5ba7c8ce6b150e2b9c8cff0f0aad60af13cb
RedemptionOperations.sol	c5d4a7bbab403036582bce5404a54017234a77e6
StakingVestingOperations.sol	cead47741f20055bc2edf339e769fe4c5f070ba3
SortedTroves.sol	70eedd0f5f5cd32e1c67c5ead36bca33479cae21
DebtToken.sol	02b1f8ad6655e3bac3ab1784764c182b79ebf545
ReservePool.sol	314d725c81399a080856bc2fedcb79824e4d9eea
HintHelpers.sol	2affab19bca778ab9461923111671f8129cd23b4

Please note: Files with a different hash value than in this table have been modified after the security check, either intentionally or unintentionally. A different hash value may (but need not) indicate a changed state or potential vulnerability that was not the subject of this scan.

Imported packages

Used code from other Frameworks/Smart Contracts (direct imports).

Dependency / Import Path	Count
@openzeppelin/contracts/access/Ownable.sol	15
@openzeppelin/contracts/token/ERC20/ERC20.sol	1
@openzeppelin/contracts/token/ERC20/IERC20.sol	13
@openzeppelin/contracts/token/ERC20/extensions/ERC20Permit.sol	1
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	8
@openzeppelin/contracts/token/ERC20/extensions/IERC20Permit.sol	1
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	7
@openzeppelin/contracts/utils/Address.sol	1
@openzeppelin/contracts/utils/cryptography/ECDSA.sol	1
@openzeppelin/contracts/utils/cryptography/MessageHashUtils.sol	1
@openzeppelin/contracts/utils/math/Math.sol	1
@pythnetwork/pyth-sdk-solidity/IPyth.sol	1
@pythnetwork/pyth-sdk-solidity/MockPyth.sol	1
@pythnetwork/pyth-sdk-solidity/PythStructs.sol	1

Note for Investors: We only audited contracts mentioned in the scope above. All contracts related to the project apart from that are not a part of the audit, and we cannot comment on its security and are not responsible for it in any way.

Audit Information

Vulnerability & Risk Level

Risk represents the probability that a certain source threat will exploit vulnerability and the impact of that event on the organization or system. The risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 - 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon aspossible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to check the repository for security-related issues, code quality, and compliance with specifications and best practices. To this end, our team of experienced pen-testers and smart contract developers reviewed the code line by line and documented any issues discovered.

We check every file manually. We use automated tools only so that they help us achieve faster and better results.

Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
    a. Review the specifications, sources, and instructions provided to SolidProof to ensure we understand the smart contract's size, scope, and functionality.
    b. Manual review of the code, i.e., reading the source code line by line to identify potential vulnerabilities.
    c. Comparison to the specification, i.e., verifying that the code does what is described in the specifications, sources, and instructions provided to SolidProof.

2. Testing and automated analysis that includes the following:
    a. Test coverage analysis determines whether test cases cover code and how much code is executed when those test cases are executed.
    b. Symbolic execution is analysing a program to determine what inputs cause each part of a program to execute.

3. Review best practices, i.e., smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on best practices, recommendations, and research from industry and academia.

4. Concrete, itemized and actionable recommendations to help you secure your smart contracts.

Overall Security

Upgradeability

Contract is not an upgradeable	✅ Deployer cannot update the contract with new functionalities
Description	The contract is not an upgradeable contract. The deployer is not able to change or add any functionalities to the contract after deploying.
Comment	N/A

Ownership

Contract ownership is not renounced	❌ **The ownership is not renounced**
Description	The owner has not renounced the ownership that means that the owner retains control over the contract's operations, including the ability to execute functions that may impact the contract's users or stakeholders. This can lead to several potential issues, including: • Centralizations • The owner has significant control over contract's operations
Example	We assume that you have funds in the contract and it has been audited by any security audit firm. Now the audit has passed. After that, the deployer can upgrade the contract to allow him to transfer the funds you purchased without any approval from you. This has the consequence that your funds can be taken by the creator.
Comment	N/A

Note - If the contract is not deployed then we would consider the ownership to be not renounced. Moreover, if there are no ownership functionalities then the ownership is automatically considered renounced.

Ownership Privileges

These functions can be dangerous. Please note that abuse can lead to financial loss. We have a guide where you can learn more about these Functions.

Minting tokens

Minting tokens refers to the process of creating new tokens in a cryptocurrency or blockchain network. This process is typically performed by the project's owner or designated authority, who can add new tokens to the network's total supply.

Contract has the ability to mint new tokens	❌ **The contract has the ability to mint new tokens**
Description	The contract has the ability to mint new tokens.
Comment	The contract contains the functionality in which the trove manager, borrower operation of the contract can mint an unlimited amount of tokens to any wallet which is not recommended as this can manipulate the total supply of the token and price of the token will also be fluctuated because of this functionality. There must be a fixed max total supply so that the tokens total supply will not be changed.

File, Line/s: L230-234
Codebase: DebtToken.sol

```
function mint(address _account, uint256 _amount) external override {
    _requireCallerIsBorrowerOperationsOrTroveManager();
    _requireMintingEnabled();
    _mint(_account, _amount);
}
```

Burning Tokens without Allowance

Burning tokens is the process of permanently destroying a certain number of tokens, reducing the total supply of a cryptocurrency or token. This is usually done to increase the value of the remaining tokens, as the reduced supply can create scarcity and potentially drive up demand.

Contract owner cannot burn tokens	✅ **The owner cannot burn tokens**
Description	The owner is not able to burn tokens without any allowances.
Comment	N/A

Blacklist addresses

Blacklisting addresses in smart contracts is the process of adding a certain address to a blacklist, effectively preventing them from accessing or participating in certain functionalities or transactions within the contract. This can be useful in preventing fraudulent or malicious activities, such as hacking attempts or money laundering.

Contract owner cannot blacklist addresses	✅ **The owner cannot blacklist addresses**
Description	The owner is not able to blacklist addresses to lock funds.
Comment	N/A

Fees and Tax

In some smart contracts, the owner or creator of the contract can set fees for certain actions or operations within the contract. These fees can be used to cover the contract's cost, such as paying for gas fees or compensating the contract's owner for their time and effort in developing and maintaining the contract.

Contract owner cannot set fees more than 25%	✅ **The owner cannot levy unfair taxes**
Description	The owner is not able to set the fees above 25%
Comment	N/A

Lock User Funds

In a smart contract, locking refers to the process of restricting access to certain tokens or assets for a specified period of time. When tokens or assets are locked in a smart contract, they cannot be transferred or used until the lock-up period has expired or certain conditions have been met.

Owner cannot lock the contract	✅ **The owner cannot lock the contract**
Description	The owner is not able to lock the contract by any functions or updating any variables.
Comment	N/A

External/Public functions

External/public functions are functions that can be called from outside of a contract, i.e., they can be accessed by other contracts or external accounts on the blockchain. These functions are specified using the function declaration's external or public visibility modifier.

State variables

State variables are variables that are stored on the blockchain as part of the contract's state. They are declared at the contract level and can be accessed and modified by any function within the contract. State variables can be defined with a visibility modifier, such as public, private, or internal, which determines the access level of the variable.

Components

📝Contracts	📚Libraries	🔍Interfaces	🎨Abstract
29	3	27	0

Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

🌐Public	💰Payable
514	43

External	Internal	Private	Pure	View
477	404	17	49	293

StateVariables

Total	🌐Public
207	176

Capabilities

Solidity Versions observed	Transfers ETH	💰 Can Receive Funds	🖥️ Uses Assembly	💣 Has Destroyable Contracts
`0.8.20` `^0.8.0`		Yes	Yes (3 asm blocks)	

📤 Transfers ETH	⚡ Low-Level Calls	👥 DelegateCall	🧱 Uses Hash Functions	🪛 ECRecover	🌀 New/Create/Create2
			yes		

Inheritance Graph

An inheritance graph is a graphical representation of the inheritance hierarchy among contracts. In object-oriented programming, inheritance is a mechanism that allows one class (or contract, in the case of Solidity) to inherit properties and methods from another class. It shows the relationships between different contracts and how they are related to each other through inheritance.

Centralization Privileges

Centralization can arise when one or more parties have privileged access or control over the contract's functionality, data, or decision-making. This can occur, for example, if a single entity controls the contract or if certain participants have special permissions or abilities that others do not.

In the project, some authorities have access to the following functions:

File	Privileges
AlternativePriceFeed.sol	• The owner can update the price feed address only once. • The owner can update the fallback price in the contract. • The owner can add the balancer price pool in the contract. • The owner can update the fallback trusted timespan for the token in the contract. • The owner can update the balance price pool for the token.
BorrowerOperations.sol	• The owner can update the trove manager, storage pool, stability pool, price feed, token manager, swap operations, troves, and Coll surplus contract address.
CollSurplusPool.sol	• The owner can update the liquidation and borrower operations contract address.
DebtToken.sol	• The token manager can update the name and symbol of the token. • The token manager can update the stock token and exchange rate. • The token manager can mint debt tokens to any arbitrary wallet. • The stability pool manager address can transfer tokens from the sender wallet to the pool address.
HintHelpers.sol	• The owner can update the sorted troves, trove manager, redemption operation, and price feed contract address.
LiquidationOperations.sol	• The owner can update the trove manager, storage pool, price feed, token manager, stability pool manager, coll surplus pool contract address.
PriceFeed.sol	• The owner can set the pyth and token manager contract address. • The token manager can add new oracle IDs in the contract. • The owner can update the alternative price feed contract address.

File	Privileges
RedemptionOperations.sol	• The owner can update the trove manager, storage pool, price feed, token manager, sorted troves, hint helpers contract address.
ReservePool.sol	• The owner can update the stability pool manager address, price feed, and token manager contract address. • The owner can update the relative stable cap value in the contract to any arbitrary number. • The owner can update the gov reserve ca value to any arbitrary number in the contract.
SortedTroves.sol	• The owner can update the trove manager, borrower operation, and redemption operation contract address.
StabilityPool.sol	• The stability pool manager address can send the gains to the depositor wallet. • The stability pool manager address withdraw all depositor's accumulated gains to depositor. • The stability pool manager address can withdraw gains from the contract.
StabilityPoolManager.sol	• The owner can update the liquidation operations, price feed, storage pool, reserve pool, token manager contract address. • The token manager can add the stability pools in the contract.
StakingOperations.sol	• The owner can set the swap operations, token manager contract address. • The token manager can set the rewards per second value. • The token manager or swap operations contract address can set pool in the contract.
StoragePool.sol	• The owner can set the borrower operations, trove manager, redemption operations, liquidation operations, stability pool and price feed contract address. • The borrower operations, trove manager, stability pool manager, redemption operations, liquidation operations contract address can add, subtract, withdraw tokens from the contract.
SwapOperations.sol	• The owner can create pair in the contract. • The owner can set the swap base fees not more than 10%. • The owner can update the swap gov fee to 100%. • The owner can et the dynamic fee address.
SwapPair.sol	• The swap operations can update the token0, token1, token manager, price feed, staking operation contract address.

File	Privileges
TokenManager.sol	• The owner can update the stability pool manager, staking operations, price feed, gov payout address in the contract. • The owner can update the rewards per second value to any arbitrary amount in the staking operation contract address. • The owner can set the burn redistribute address. • The owner can set the staking alloc point in the staking operations contract address. • The owner can enable/disable minting in the contract. • The owner can disable or enable the debt minting. • The owner can update the name and symbol of the debt tokens. • The owner can update the next stock split value in the debt token address. • The owner can set the stock exchange parameters in the debt token contract address. • The owner can update the oracle ID in the price feed contract address. • The owner can update the debt in the contract without the oracle ID. • The owner can update the coll token with or without the oracle ID. • The owner can update the coll token with supported collateral ratio.
StakingVestingOperations.sol	• The owner can update the stakingOPS contract address.

File	Privileges
TroveManager.sol	• The owner can update the borrower, redemption operations, liquidation operation, storage pool, price feed sorted troves, token manager, reserve pool contract address. • The owner can enable/disable liquidation. • The owner can enable/disable redeeming. • The owner can enable/disable minting on closed hours. • The owner can set the borrowing fee floor upto 100%. • The owner can update the borrowing fee Gov discount upto 100%. • The owner can update the borrowing interest rate upto 100%. • The owner can update the max debt as collateral upto 100%. • The borrower operations, redemption operation, liquidation operation address can set the trove status for the borrowers. • The borrower operations, redemption operation, liquidation operation address can update the borrower's stake based on their latest collateral value. • The borrower operations, redemption operation, liquidation operation address can remove the total stakes sum and their sakes to zero. • The borrower operations, redemption operation, liquidation operation address can update the snapshots of system total stakes and collateral. • The borrower operations, redemption operation, liquidation operation address can apply for the pending rewards. • The borrower operations, redemption operation, liquidation operation address can update the trove reward snapshots. • The borrower operations, redemption operation, liquidation operation address can increase/decrease trove call. • The borrower operations, redemption operation, liquidation operation address can increase/decrease trove debt.

Recommendations

To avoid potential hacking risks, the client should manage the private key of the privileged account with care. Additionally, we recommend enhancing the security practices of centralized privileges or roles in the protocol through a decentralized mechanism or smart-contract-based accounts, such as multi-signature wallets.

Here are some suggestions of what the client can do:

- Consider using multi-signature wallets: Multi-signature wallets require multiple parties to sign off on a transaction before it can be executed, providing an extra layer of security, e.g. Gnosis Safe
- Use of a timelock at least with a latency of, e.g. 48-72 hours for awareness of privileged operations
- Introduce a DAO/Governance/Voting module to increase transparency and user involvement
- Consider Renouncing the ownership so that the owner can no longer modify any state variables of the contract. Make sure to set up everything before renouncing.

Audit Results

Critical issues

No critical issues

High issues

No high issues

Medium issues

#1 | Missing 'require' check.

File	Severity	Location	Status
AlternativePriceFees.sol	Medium	L162	ACK

Description - The contract contains the functionality in which the price calculation is done which the help of the non-target USD and target weight and balance in the contract in this function the value of nontarget weight and balance should not be zero otherwise the functionality will fail and the price calculation will not be possible.

#2 | Missing Threshold

File	Severity	Location	Status
ReservePool.sol	Medium	L57-60, L62-65	ACK

Description - The contract contains the functionality in which the owner can update any arbitrary value in the relatively stable cap or gov reserve cap, including zero, which is not recommended as this can fail the functionalities if the value is set to zero. Also, in the staking operations contract, the token manager can update any arbitrary value in the rewards per second value in the contract, which can set the rewards value to zero. There must be a check so that the rewards in the contract should not be zero.

#3 | Manipulated or Irrelevant _priceUpdateData.

File	Severity	Location	Status
PriceFeed.sol	Medium	L128-133	ACK

Description - The updatePythPrices function in the PriceFeed contract lacks validation for _priceUpdateData, making it vulnerable to processing manipulated, irrelevant, or outdated data. This can result in inaccurate pricing, potentially leading

to financial losses or exploitation. The issue arises because _priceUpdateData is passed directly to the pyth.updatePriceFeeds function without verifying its structure, timestamps, or association with registered tokens.

Remediation - To mitigate this, validation should be implemented to ensure the data format is correct, timestamps are recent, and the oracle IDs correspond to known tokens. A fallback mechanism using altFeed should provide redundancy in case of invalid data. Limiting update frequency, flagging unusual data for manual review, and logging update details can further enhance security. These measures prevent incorrect price updates, maintain system integrity, and protect against malicious attempts, ensuring the accuracy and reliability of the pricing mechanism.

#4 | Missing Non-reentrant check.

File	Severity	Location	Status
StakingVestingOperations.sol	Medium	L85-109, L111-113, L115-118	ACK

Description - The contract contains the functionality in which the claim function (and similar functions like withdraw), making an external call to transfer ERC20 tokens using the transfer function. Since the contract hasn't yet marked the claim as completed (i.e., updated the claimed variable), the attacker can exploit this to re-enter the claim function, repeatedly withdrawing tokens. The transfer happens first (interaction with an external contract) before the internal state is updated. The nonReentrant modifier ensures that any attempts to call the claim function again during execution are blocked, providing an additional safeguard. Therefore, It is recommended to do the check-effect-transaction method or use the non-reentrant modifier to prevent the code from this issue.

#5 | Possible front-running.

File	Severity	Location	Status
RedemptionOperations.sol	Medium	L79-203	ACK

Description - The function adjusts the collateral ratios (CR) of troves based on their current state and applies redemptions accordingly. If a large redemption is about to occur, it might lower the Total Collateral Ratio (TCR) of the system or impact the collateral available for subsequent redemptions. A front-runner could preemptively redeem stablecoins before the TCR decreases or before the system enters Recovery Mode, where certain conditions might apply (e.g., no redemptions are allowed).

Remediation - Implementing fair ordering protocols, like batch auctions or Time-Bound Mempool Access, could prevent front-runners from gaining an advantage by observing pending transactions.

Low issues

#1 | Missing Events

File	Severity	Location	Status
StabilityPoolManager.sol	Low	L188-199, L201-211, L213-220	ACK
SwapOperations.sol	Low	L104-107, L113-116, L123-126	ACK
SwapPair.sol	Low	L46-60	ACK

Description - Make sure to emit events for all the critical parameter changes in the contract to ensure the transparency and trackability of all the state variable changes.

#2 | Stricter Collateral and Total System Ratio (TCR) Requirements.

File	Severity	Location	Status
BorrowerOperations.sol	Low	L166-219	ACK

Description - In the BorrowerOperations contract introduce stricter enforcement of Initial Collateral Ratio (ICR) and Total Collateral Ratio (TCR) to ensure system stability. These updates require ICR to be above IMCR (minimum collateral ratio) and CCR (critical collateral ratio) in recovery mode, which helps prevent under-collateralized positions and protects against systemic insolvency. While this enhances protocol resilience by enforcing a higher standard of financial health, it reduces usability by making it harder for borrowers to maintain their positions, potentially leading to increased liquidations in volatile market conditions.

Remediation - To balance resilience and usability, the protocol can implement gradual enforcement mechanisms and real-time alerts to notify users when their ICR is approaching critical levels. Introducing a grace period for collateral adjustments and making TCR dynamically adjustable based on market conditions can improve flexibility. These measures ensure system security while maintaining a smoother user experience.

Informational issues

#1 | NatSpec documentation missing

File	Severity	Location	Status
All	Informational	—	ACK

Description - If you started to comment on your code, comment on all other functions, variables etc.

#2 | Unlimited Reserve Pool Withdrawal and Collateral Seizure in Liquidation.

File	Severity	Location	Status
LiquidationOperations.sol	Informational	L315-325	ACK

Description - The _compensateLossViaReservePool(), which allows the protocol to withdraw funds from the ReservePool when a liquidated trove has a collateral ratio (CR) below 100%. While this change aims to stabilize the system by covering shortfalls, it lacks limitations on how much can be withdrawn, creating a risk of Reserve Pool depletion. Additionally, borrowers with CR below 100% lose all collateral, even if excess remains after covering their debt. This is unfair to borrowers who could still reclaim some funds but are instead fully liquidated with no recovery option. Furthermore, there is no governance control over these withdrawals, meaning large amounts can be taken from the Reserve Pool without oversight, reducing transparency and increasing centralization risks.

Remediation - To ensure fairness and stability, the function should be modified to prevent excessive Reserve Pool depletion and allow borrowers to reclaim any remaining collateral after liquidation. First, a withdrawal cap should be introduced, limiting the amount that can be used from the Reserve Pool per liquidation (e.g., 30% of total Reserve Pool funds). Second, the liquidation logic should be updated to allow borrowers to reclaim excess collateral, even if their trove is under-collateralized, instead of sending all funds to the Reserve Pool. Finally, governance approval should be required for large withdrawals (e.g., any withdrawal exceeding 20% of the Reserve Pool) to increase transparency and decentralization.

Legend for the Issue Status

Attribute or Symbol	Meaning
Open	The issue is not fixed by the project team.
Fixed	The issue is fixed by the project team.
Acknowledged(ACK)	The issue has been acknowledged or declared as part of business logic.

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY