Palladium Labs

Audit Report

Fri Jun 06 2025

contact@bitslab.xyz https://twitter.com/scalebit_

ScaleBit

Palladium Labs Audit Report

1 Executive Summary

1.1 Project Information

Description	Palladium is a decentralized finance protocol implementing a Collateralized Debt Position (CDP) system.
Type	Stablecoin
Auditors	ScaleBit
Timeline	Fri May 23 2025 - Fri May 30 2025
Languages	Solidity
Platform	Bitcoin Network
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/PalladiumLabs/Palladium-SmartContracts
Commits	954b46b3128898eddb3181efa5514b238cc9c947 97f18d7a22c3bfe073c117d8fa3e1f0a88bed93a

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
CSP	contracts/CollSurplusPool.sol	9980218a46282836ba9fcae54afc1a9272affb4c
IBO	contracts/Interfaces/IBorrowerOperations.sol	15ca9d4225ddce297de39e20a525394e4232079c
IST	contracts/Interfaces/ISortedTroves.sol	b9e809f97a0e0a6ace1ac1ff9c9c1d4dba2253fb
IERC2P	contracts/Interfaces/IERC2612Permit.sol	4bf9c24d84d79fab7f56467dc58f3530ec521bc1
IAP	contracts/Interfaces/IActivePool.sol	9fec177aa7981f7a28bee13be54a3aa4afdf3249
IDP	contracts/Interfaces/IDefaultPool.sol	57bda58e773dae461794d24afd727bb6ac8a8d52
IDE	contracts/Interfaces/IDeposit.sol	edd2f648572b8c00e255901dcc1f1c8a6b728266
IPDMS	contracts/Interfaces/IPDMStaking.sol	3373c7cc5d78a9aae8a4faa626a5c1319220d8fe
ITMO	contracts/Interfaces/ITroveManagerOperations.sol	aa0473a6489d74fea5b621258d3260b6e13ae021
IPB	contracts/Interfaces/IPalladiumBase.sol	1d98f39bf480fe020ad49dc943131b5657d6fc49

IPO	contracts/Interfaces/IPool.sol	721635fd9526d7f735012e9d9c18ae2c649c0b7c
ICSP	contracts/Interfaces/ICollSurplusPool.sol	1c0ab4c4c57c851cf9e25b643d0eacefbdbe1fcf
ISP	contracts/Interfaces/IStabilityPool.sol	f9a5422a35c10dcd4e9a9400fdcf5f3486fd3610
IDT	contracts/Interfaces/IDebtToken.sol	579c322eca4ead778eb0c5d54468e3877ed8a914
IPF	contracts/Interfaces/IPriceFeed.sol	6035b8a8389b75074dcb3eeafdbd57c40ec5c8ef
ICI	contracts/Interfaces/ICommunityIssuance.sol	779b35396c7d33a6b8bb2e1546850e1784630720
IFC	contracts/Interfaces/IFeeCollector.sol	4ba23f8fb4b824c63db314a67ce27ceb22784f16
ITM	contracts/Interfaces/ITroveManager.sol	8ee0123ec7aed961f2161e88312d78bd541daef6
IERC2D	contracts/Interfaces/IERC20Decimals.sol	7920086d6058ab6ace0613b9d65b14abbd2d3f25
IAC	contracts/Interfaces/IAdminContract.sol	c552ec0f8f36a9122ef7d077d2a0368d31d9562f
WE2UPA	contracts/Pricing/WstEth2UsdPriceAggregator.sol	c42e6560026d15ccb93bbb42390719e1a1908242
API3PI	contracts/Pricing/API3ProxyInterface.sol	75fe032d29ba565cdb5e08b43c1226ef3dfe4ffe

PFL2	contracts/Pricing/PriceFeedL2.sol	25f4a01bea45cacd3407349a491dec9f1dd48591
FPA	contracts/Pricing/FixedPriceAggregator.sol	e4c0c90ca014e3217de92768b4d04f8199c12ee4
SE2EPA	contracts/Pricing/SfrxEth2EthPriceAggregator.sol	455fc79eee6cee332b65d83457932d43299746f8
SPO	contracts/StabilityPool.sol	40aee19d503fc37b0ff8052423856cffce3b11e9
BOP	contracts/BorrowerOperations.sol	bcd809a10c741c64124789acdff8ade308e59528
ADD	contracts/Addresses.sol	d6b02af6519702347e180c095794366eb1a33c32
PFE	contracts/PriceFeed.sol	4bf57c8f75865d4229e8d1b16f8b6ab6422d1598
ACO	contracts/Dependencies/AddressesConfigurable.sol	b3f96e8d0787c3c6bffe5546228ac946c9bfc1c8
ERC2P	contracts/Dependencies/ERC20Permit.sol	f6309b125e601bac8ca80b6160eea0bca95c4514
PBA	contracts/Dependencies/PalladiumBase.sol	b5b69c21e826032f9b88fc3b860d1678d26e1a70
AMA	contracts/Dependencies/AddressesMainnet.sol	ae89323a092460d76c109a4d0126c5d054483ce2
BMA	contracts/Dependencies/BaseMath.sol	f73260a2a29b9f325f8ae4c1236e2d9a8a820aac

STR	contracts/Dependencies/SafetyTransfer.sol	ff3f611947ef922d32eaeb0c6273db66ee6757bd
PMA	contracts/Dependencies/PalladiumMath.sol	dc628af27cc5a8b988375ad21986a90cb1c5b872
TMO	contracts/TroveManagerOperations.sol	a6fc5187fe15b7665e3205967fe79b1a14b914b6
ACO1	contracts/AdminContract.sol	284ae7fa88e74451fae73d08dc6abb12867c0f9a
CIS	contracts/PDM/CommunityIssuance.sol	647e2eeba8bd7ef3e310926b5bb0f1ab38b0a103
PDMS	contracts/PDM/PDMStaking.sol	096be521ce8aa0a8a027c353c556d5f5be964e6d
PDMT	contracts/PDM/PDMToken.sol	843c267d4ccd963257ec4bb63cf96fe210519309
LPDM	contracts/PDM/LockedPDM.sol	627970e72c2ff8f2c7dd20f36ce5da5460d3f1d7
FCO	contracts/FeeCollector.sol	d5b592ee0014466b43a04a8cae05eb85afd6b04d
DTO	contracts/DebtToken.sol	7131c276bdd8b45a8c8d0412815c02e49e7de13b
GPO	contracts/GasPool.sol	21560619990d476af1df57aa6f50692ac0265553
TIM	contracts/Timelock.sol	db2bb5cb8b03cbaaf562fc6affe4ec1e18c67f9f

STR1	contracts/SortedTroves.sol	c548aa5086b91fea72a2a243b5d5fe86bdf5effd
DPO	contracts/DefaultPool.sol	20458a3d2f562f94d9bd6c4a46ac6f0756752a12
APO	contracts/ActivePool.sol	d7c87f1710eb45d0a326e0df0a54890bad1e3b3f
TMA	contracts/TroveManager.sol	fc5273843547af61eec5f707d5ff5dcec72c917b

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	7	4	3
Informational	1	1	0
Minor	3	2	1
Medium	1	0	1
Major	2	1	1
Critical	0	0	0

1.4 ScaleBit Audit Breakdown

ScaleBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow
- Number of rounding errors
- Unchecked External Call
- Unchecked CALL Return Values
- Functionality Checks
- Reentrancy
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic issues
- Gas usage
- Fallback function usage
- tx.origin authentication
- Replay attacks
- Coding style issues

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by `Palladium Labs` to identify any potential issues and vulnerabilities in the source code of the `Palladium` smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.
The Palladium Smart Contracts is a fork from Gravita. This audit focuses on the changes Palladium has made to the Gravita codebase:

- Vessels have been renamed to Trove across the codebase
- Few parameters have been updated in FeeManager.sol to make it a one-time fee model (which was earlier a refundable fee model)

During the audit, we identified 7 issues of varying severity, listed below.

ID	Title	Severity	Status
ACO-1	Centralization Risk	Major	Fixed
ACO-2	Insufficient Parameters Validation	Minor	Fixed
ACO-3	Missing Check for Address	Minor	Fixed
ACO-4	Lack of Event Emit	Informational	Fixed
AMA-1	Addresses Not Updated	Major	Acknowledged
APO-1	Single-step Ownership Transfer Can be Dangerous	Minor	Acknowledged
SPO-1	`Initialize` Could Be Front-Run	Medium	Acknowledged

3 Participant Process

Here are the relevant actors with their respective abilities within the Palladium Labs Smart Contract :

Admin

- Admin can finalize setup initialization through the `setSetupIsInitialized()` function.
- Admin can add new collateral types through the `addNewCollateral()` function.
- Admin can configure collateral parameters through the `setCollateralParameters()` function.
- Admin can activate or deactivate collaterals through the `setIsActive()` function.
- Admin can set borrowing fee rates through the `setBorrowingFee()` function.
- Admin can set Critical Collateralization Ratio through the `setCCR()` function.
- Admin can set Minimum Collateralization Ratio through the `setMCR()` function.
- Admin can set minimum net debt threshold through the `setMinNetDebt()` function.
- Admin can set minting cap limits through the `setMintCap()` function.
- Admin can set percent divisor values through the `setPercentDivisor()` function.
- Admin can set redemption fee floor rates through the `setRedemptionFeeFloor()` function.
- Admin can set redemption block timestamps through the `setRedemptionBlockTimestamp()` function.
- Admin can authorize contract upgrades through the `authorizeUpgrade()` function.
- Admin can set or update primary and fallback oracles for an asset through the `setOracle()` function.
- Admin can transfer contract ownership through the `transferOwnership()` function.
- Admin can renounce contract ownership through the `renounceOwnership()` function.

- Admin can stop or resume minting for a specific collateral asset through the `emergencyStopMinting()` function.
- Admin can add an address to the whitelist through the `addWhitelist()` function.
- Admin can remove an address from the whitelist through the `removeWhitelist()` function.
- Admin can set a new delay by queuing and executing a transaction that calls the `setDelay()` function in `Timelock` contract.
- Admin can set a pending admin by queuing and executing a transaction that calls the `setPendingAdmin()` function in `Timelock` contract.
- Admin can queue transactions by calling the `queueTransaction()` function in `Timelock` contract.
- Admin can cancel queued transactions by calling the `cancelTransaction()` function in `Timelock` contract.
- Admin can execute queued transactions by calling the `executeTransaction()` function in `Timelock` contract.
- PendingAdmin can accept the admin role by calling the `acceptAdmin()` function in `Timelock` contract.

Whitelisted Contract

- Whitelisted contracts can mint new PUSD tokens through the `mintFromWhitelistedContract()` function.
- Whitelisted contracts can burn PUSD tokens through the `burnFromWhitelistedContract()` function.

BorrowerOperations

- BorrowerOperations can mint PUSD tokens for an account (when not blocked) through the `mint()` function.
- BorrowerOperations can burn PUSD tokens from an account through the `burn()` function.

- BorrowerOperations can increase borrower debt through the `increaseDebt()` function.
- BorrowerOperations can decrease borrower debt through the `decreaseDebt()` function.
- BorrowerOperations can close borrower debt through the `closeDebt()` function.
- BorrowerOperations can transfer assets to other contracts through the `sendAsset()` function.
- BorrowerOperations can notify the contract of received ERC20 tokens through the `receivedERC20()` function.
- BorrowerOperations can insert nodes through the `insert()` function.
- BorrowerOperations can reposition nodes through the `reInsert()` function.
- BorrowerOperations can add trove owners to arrays through the `addTroveOwnerToArray()` function.
- BorrowerOperations can update trove reward snapshots through the `updateTroveRewardSnapshots()` function.
- BorrowerOperations can update stakes and total stakes through the `updateStakeAndTotalStakes()` function.
- BorrowerOperations can set trove statuses through the `setTroveStatus()` function.
- BorrowerOperations can increase trove collateral through the `increaseTroveColl()` function.
- BorrowerOperations can decrease trove collateral through the `decreaseTroveColl()` function.
- BorrowerOperations can increase trove debt through the `increaseTroveDebt()` function.
- BorrowerOperations can decrease trove debt through the `decreaseTroveDebt()` function.

StabilityPool

- StabilityPool can transfer PUSD tokens from a sender to itself through the `sendToPool()` function.
- StabilityPool can return PUSD tokens from the pool to a receiver through the `returnFromPool()` function.
- StabilityPool can burn PUSD tokens from an account through the `burn()` function.

TroveManager

- TroveManager can burn PUSD tokens from an account through the `burn()` function.
- TroveManager can return PUSD tokens from the pool to a receiver through the `returnFromPool()` function.
- TroveManager can liquidate borrower debt through the `liquidateDebt()` function.
- TroveManager can handle redemption fees through the `handleRedemptionFee()` function.
- TroveManager can decrease borrower debt through the `decreaseDebt()` function.
- TroveManager can close borrower debt through the `closeDebt()` function.
- TroveManager can increase debt amount through the `increaseDebt()` function.
- TroveManager can transfer assets to other contracts through the `sendAsset()` function.
- TroveManager can insert nodes through the `insert()` function.
- TroveManager can remove nodes through the `remove()` function.
- TroveManager can reposition nodes through the `reInsert()` function.
- TroveManager can offset debt during liquidations through the `offset()` function.

TroveManagerOperations

- TroveManagerOperations can transfer assets to other contracts through the `sendAsset()` function.
- TroveManagerOperations can execute full redemptions through the `executeFullRedemption()` function.

- TroveManagerOperations can execute partial redemptions through the `executePartialRedemption()` function.
- TroveManagerOperations can finalize redemptions through the `finalizeRedemption()` function.
- TroveManagerOperations can update base rates from redemptions through the `updateBaseRateFromRedemption()` function.
- TroveManagerOperations can move pending trove rewards to the active pool through the `movePendingTroveRewardsToActivePool()` function.
- TroveManagerOperations can redistribute debt and collateral through the `redistributeDebtAndColl()` function.
- TroveManagerOperations can update system snapshots excluding collateral remainders through the `updateSystemSnapshots_excludeCollRemainder()` function.
- TroveManagerOperations can close troves via liquidation through the `closeTroveLiquidation()` function.
- TroveManagerOperations can send gas compensation through the `sendGasCompensation()` function.
- TroveManagerOperations can apply pending rewards through the `applyPendingRewards()` function.
- TroveManagerOperations can remove stakes through the `removeStake()` function.
- TroveManagerOperations can close troves through the `closeTrove()` function.

DefaultPool

- DefaultPool can notify the contract of received ERC20 tokens through the `receivedERC20()` function.

StabilityPool

- StabilityPool can decrease debt amount through the `decreaseDebt()` function.
- StabilityPool can receive assets through the `sendAsset()` function.

AdminContract

- AdminContract can add new collateral types through the `addCollateralType()` function.

ActivePool

- ActivePool can update collateral balances through the `receivedERC20()` function.

Timelock

- Timelock can set the redemption softening parameter through the `setRedemptionSofteningParam()` function.

User

- User can open a trove through the `openTrove()` function.
- User can adjust their trove through the `adjustTrove()` function.
- User can close their trove through the `closeTrove()` function.
- User can liquidate an undercollateralized trove through the `liquidate()` function.
- User can liquidate a sequence of undercollateralized troves through the `liquidateTroves()` function.
- User can liquidate a custom list of troves through the `batchLiquidateTroves()` function.
- User can redeem collateral by burning debt tokens through the `redeemCollateral()` function.
- User can provide debt tokens to the Stability Pool through the `provideToSP()` function.
- User can withdraw debt tokens and claim collateral gains through the `withdrawFromSP()` function.
- User can transfer debt tokens to a valid recipient (not zero address and not the token contract itself) through the `transfer()` function.
- User can transfer debt tokens from another account (with allowance) to a valid recipient through the `transferFrom()` function.

- User can approve other addresses to spend debt tokens on their behalf through the `approve()` function.
- User can use off-chain signatures to authorize other addresses to use their debt tokens through the `permit()` function.
- User can give up existing role permissions through the `renounceRole()` function.

4 Findings

ACO-1 Centralization Risk

Severity: Major

Status: Fixed

Code Location:

contracts/AdminContract.sol;

contracts/ActivePool.sol;

contracts/DefaultPool.sol;

contracts/TroveManager.sol;

contracts/FeeCollector.sol

Descriptions:

Centralization risk was identified in the smart contract:

- Admin can update collateral parameters and various debt related parameters.
- Admin can modify any user's debt and collateral data.
- Admin can increase the balance of any type of collateral.
- Admin can transfer any amount of collateral to any address.
- Admin can mint and burn any number of debt tokens at any address.

Suggestion:

It is recommended that measures be taken to reduce the risk of centralization, such as a multi-signature mechanism.

Resolution:

The project explained that multisig wallets will be used for admin functionalities.

ACO-2 Insufficient Parameters Validation

Severity: Minor

Status: Fixed

Code Location:

contracts/AdminContract.sol#142-143

Descriptions:

The `setCollateralParameters()` function does not check whether `MCR` is less than `CCR`. If `MCR` is set to be greater than `CCR`, it may cause the liquidation mechanism to fail.

Suggestion:

It is recommended to add validation for `MCR` and `CCR` to ensure that the set `MCR` is less than `CCR`.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

ACO-3 Missing Check for Address

Severity: Minor

Status: Fixed

Code Location:

contracts/AdminContract.sol#324;

contracts/ActivePool.sol#131;

contracts/BorrowerOperations.sol#656;

contracts/FeeCollector.sol#328;

contracts/PriceFeed.sol#211;

contracts/SortedTroves.sol#415;

contracts/StabilityPool.sol#900;

contracts/DefaultPool.sol#94;

contracts/TroveManager.sol#702;

contracts/TroveManagerOperations.sol#969;

contracts/CollSurplusPool.sol#93

Descriptions:

In the `authorzeUpgrade()` function, there is no check to see if the entire address parameter of `newImplementation` is not set to 0 address. If it is set to 0 address incorrectly, it can cause a denial of service issue.

Suggestion:

It is recommended to add validity checks for `newImplementation`, such as ensuring it is a known role or a non-zero address.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

ACO-4 Lack of Event Emit

Severity: Informational

Status: Fixed

Code Location:

contracts/AdminContract.sol#152

Descriptions:

Functions such as `setIsActive()` lack logs, making the contract's activities difficult to track.

Suggestion:

It is recommended to add event emission for this operation.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

AMA-1 Addresses Not Updated

Severity: Major

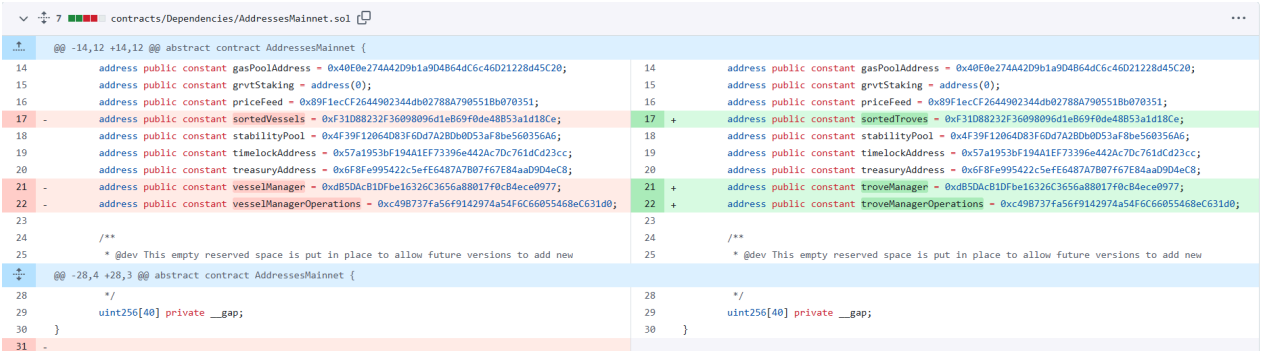
Status: Acknowledged

Code Location:

contracts/Dependencies/AddressesMainnet.sol#14-22

Descriptions:

This project is modified based on the Gravita project, but the main network address of this project has not been updated and instead uses the address of the Gravita project. This may result in assets being transferred to the wrong address or causing denial of service issues after project deployment.



```
7 contracts/Dependencies/AddressesMainnet.sol
@@ -14,12 +14,12 @@ abstract contract AddressesMainnet {
        address public constant gasPoolAddress = 0x40E0e274A42D9b1a9D4B64dC6c46D21228d45C20;
        address public constant grvtStaking = address(0);
        address public constant priceFeed = 0x89F1ecCF2644902344db02788A790551Bb070351;
-       address public constant sortedVessels = 0xF31D88232F36098096d1eB69f0de48B53a1d18Ce;
+       address public constant sortedTroves = 0xF31D88232F36098096d1eB69f0de48B53a1d18Ce;
        address public constant stabilityPool = 0x4F39F12064D83F6Dd7A2BDb0D53aF8be560356A6;
        address public constant timelockAddress = 0x57a1953bF194A1EF73396e442Ac7Dc761dCd23cc;
        address public constant treasuryAddress = 0x6F8Fe995422c5efE6487A7B07f67E84aaD9D4eC8;
-       address public constant vesselManager = 0xdB5DAcB1DFbe16326C3656a88017f0cB4ece0977;
-       address public constant vesselManagerOperations = 0xc49B737fa56f9142974a54F6C66055468eC631d0;

        /**
         * @dev This empty reserved space is put in place to allow future versions to add new
@@ -28,4 +28,3 @@ abstract contract AddressesMainnet {
         */
        uint256[40] private __gap;
    }
-
```

Suggestion:

It is recommended to update the mainnet address of this project.

Resolution:

The client confirmed addresses will be updated after Mainnet deployment.

APO-1 Single-step Ownership Transfer Can be Dangerous

Severity: Minor

Status: Acknowledged

Code Location:

contracts/ActivePool.sol#20;

contracts/DefaultPool.sol#20;

contracts/CollSurplusPool.sol#13;

contracts/FeeCollector.sol#15;

contracts/AdminContract.sol#14;

contracts/SortedTroves.sol#43;

contracts/PDM/PDMStaking.sol#16;

contracts/PDM/CommunityIssuance.sol#14

Descriptions:

The `transferOwnership()` function inherited from the `OwnableUpgradeable` contract carries the risk of single step permission transfer. Once called, the new owner will immediately gain permission. If the new owner's address is entered incorrectly or if there are security issues with the private key, it may lead to the contract being in an unmanaged state.

Suggestion:

It is recommended to use a two-step permission transfer mechanism. Reference: (https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/master/contracts/access/Ownable2StepUpgradeable.sol).

SPO-1 `Initialize` Could Be Front-Run

Severity: Medium

Status: Acknowledged

Code Location:

contracts/StabilityPool.sol#209;

contracts/CollSurplusPool.sol#25;

contracts/FeeCollector.sol#34;

contracts/DefaultPool.sol#30;

contracts/SortedTroves.sol#67;

contracts/ActivePool.sol#67;

contracts/TroveManager.sol#103;

contracts/PDM/PDMStaking.sol#52;

contracts/BorrowerOperations.sol#60;

contracts/TroveManagerOperations.sol#37;

contracts/AdminContract.sol#86;

contracts/PriceFeed.sol#40;

contracts/PDM/CommunityIssuance.sol#50

Descriptions:

In the contract, by calling the `initialize` function to initialize the contracts, there is a potential issue that malicious attackers preemptively call the initialize function to initialize and there is no access control verification for the initialize functions.

Suggestion:

It is suggested that the `initialize` function can be called only by privileged addresses or in the same transaction immediately after the contract is created to avoid being maliciously called by the attacker.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.