ERD - Ethereum Reserve Dollar

Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **June 7th, 2023 - July 12th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	07/07/2023	Omar Alshaeb
0.2	Document Updates	07/07/2023	Omar Alshaeb
0.3	Document Updates	07/12/2023	Omar Alshaeb
0.4	Draft Review	07/13/2023	Gokberk Gulgun
0.5	Draft Review	07/13/2023	Gabi Urrutia
0.6	Document Updates	07/17/2023	Omar Alshaeb
1.0	Remediation Plan	07/17/2023	Omar Alshaeb
1.1	Remediation Plan Review	07/17/2023	Gabi Urrutia
1.2	Document Updates	07/18/2023	Omar Alshaeb
1.3	Document Updates Review	07/18/2023	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Gokberk Gulgun	Halborn	Gokberk.Gulgun@halborn.com
Omar Alshaeb	Halborn	Omar.Alshaeb@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

ERD is a decentralized protocol that allows Ether/LSD (Liquid Staking Derivatives) holders to obtain maximum liquidity against their collateral while paying low interest. After locking up ETH/wrapper ETH as collateral in a smart contract and creating an individual position called a trove, the user can get instant liquidity by minting EUSD, a USD-pegged stablecoin. Each trove is required to be collateralized at a minimum of 110%. Any owner of EUSD can redeem their stablecoins for the underlying collateral at any time. The redemption mechanism along with algorithmically adjusted fees guarantee a minimum stablecoin value of USD 1.

ERD engaged Halborn to conduct a security assessment on their smart contracts beginning on June 7th, 2023 and ending on July 12th, 2023. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 ASSESSMENT SUMMARY

The team at Halborn was provided five weeks for the engagement and assigned a full-time security engineer to assessment the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to:

- Ensure that smart contract functions operate as intended.
- Identify potential security issues with the smart contracts.

In summary, Halborn identified some security risks that were successfully addressed by the ERD team.

1.3 SCOPE

1. IN-SCOPE:

The security assessment was scoped to the following smart contracts:

- contracts/Dependencies/*
- contracts/Interfaces/*
- contracts/Oracles/*
- contracts/ActivePool.sol
- contracts/BorrowerOperations.sol
- contracts/CollSurplusPool.sol
- contracts/CollateralManager.sol
- contracts/CommunityIssuance.sol
- contracts/DataTypes.sol
- contracts/DefaultPool.sol
- contracts/EToken.sol
- contracts/EUSDToken.sol
- contracts/Errors.sol
- contracts/GasPool.sol
- contracts/HintHelpers.sol
- contracts/LiquidityIncentive.sol
- contracts/Migrations.sol
- contracts/MultiTroveGetter.sol
- contracts/PriceFeed.sol
- contracts/SortedTroves.sol
- contracts/StabilityPool.sol
- contracts/Treasury.sol
- contracts/TroveDebt.sol
- contracts/TroveInterestRateStrategy.sol
- contracts/TroveLogic.sol
- contracts/TroveManager.sol
- contracts/TroveManagerDataTypes.sol
- contracts/TroveManagerLiquidations.sol
- contracts/TroveManagerRedemptions.sol

Commit ID: c46e664f30a3ed28a0420afd11788b045527a39a

2. **REMEDIATION PR/COMMITS:**

- Fix Commit ID (HAL-01): 6657817edcc30b48e41836756a3f41fa34ef779d
- Fix Commit ID (HAL-02): 95ad8f438291ec082f34dab97dc57ecf2494209c
- Fix Commit ID (HAL-03): 93c803ae22a7676e05a1fa6ec884589de28fd619
- Fix Commit ID (HAL-04): 1961c7fc04181f770468d575b5b402a07b8ab239
- Fix Commit ID (HAL-05): 0aaf1539e5897aca96034f20f82a0ec1a8d45182
- Fix Commit ID (HAL-06): f77899108075aef9f90e1e31ab4e1ab22c20d89c

1.4 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the bridge code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the assessment:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions. (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment. (Foundry)

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 IMPACT

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

EXECUTIVE OVERVIEW

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	1	4

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) PRICE MANIPULATION RISK IN STETHORACLE CONTRACT	Medium (5.0)	SOLVED - 07/17/2023
(HAL-02) NON-TRANSFERABLE OWNER IN MIGRATIONS CONTRACT	Low (2.5)	SOLVED - 07/17/2023
(HAL-03) POSSIBLE DOS DUE TO COLLATERALMANAGER.COLLATERALSUPPORT SIZE	Informational (1.0)	SOLVED - 07/17/2023
(HAL-04) MISSING A CAP FOR EUSD GAS COMPENSATION	Informational (0.0)	SOLVED - 07/17/2023
(HAL-05) LONG LITERAL UINT256 USED IN COLLATERALMANAGER	Informational (0.0)	SOLVED - 07/17/2023
(HAL-06) MISSING REQUIREISCONTRACT CHECK IN COLLATERALMANAGER.SETADDRESSES()	Informational (0.0)	SOLVED - 07/18/2023

FINDINGS & TECH DETAILS

4.1 (HAL-01) PRICE MANIPULATION RISK IN STETHORACLE CONTRACT - MEDIUM (5.0)

Description:

If the owner's private key of the contract StETHOracle.sol gets stolen, or the owner himself acts maliciously, it is possible to directly manipulate the price oracle by calling the setPrice() function and updating the lastGoodPrice storage variable without using Chainlink. Hence, all parts of the protocol using fetchPrice_view() would get as a result an incorrect price for the token.

Code Location:

Listing 1: StETHOracle.sol (Line 339)

```
338 function setPrice(uint _price) external onlyOwner {
339     lastGoodPrice = _price;
340     emit LastGoodPriceUpdated(_price);
341 }
```

Listing 2: StETHOracle.sol (Line 139)

```
138 function fetchPrice_view() external view override returns (uint256
    ↳ ) {
139     return lastGoodPrice;
140 }
```

Proof of Concept:

1. The owner calls setPrice() and significantly decreases the token price.
2. All active troves now have ICR < MCR, hence can be liquidated.
3. Liquidate all troves and distribute all the rewards.
4. Set the correct token price again.

BVSS:

AO:A/AC:L/AX:L/C:N/I:H/A:N/D:M/Y:M/R:P/S:U (5.0)

Recommendation:

Updating the lastGoodPrice storage variable is only recommended by requesting it to Chainlink and checking if the response is acceptable.

Remediation Plan:

SOLVED: The ERD team solved the issue with the following commit ID.

Commit ID : 6657817edcc30b48e41836756a3f41fa34ef779d

4.2 (HAL-02) NON-TRANSFERABLE OWNER IN MIGRATIONS CONTRACT - LOW (2.5)

Description:

The owner of the Migrations.sol contract is set in the constructor() and cannot be changed anymore. If there is any issue with the owner account, the contract can be left useless without being able to change the ownership to a new address.

Code Location:

Listing 3: Migrations.sol (Line 14)

```
13 constructor() {
14     owner = msg.sender;
15 }
```

Listing 4: Migrations.sol (Line 10)

```
 9 modifier restricted() {
10     if (msg.sender == owner) _;
11 }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:M/D:N/Y:N/R:P/S:U (2.5)

Recommendation:

The use of Ownable contract from OpenZeppelin is recommended to handle the ownership of the contract.

Remediation Plan:

SOLVED: The ERD team solved the issue with the following commit ID.

Commit ID : 95ad8f438291ec082f34dab97dc57ecf2494209c

4.3 (HAL-03) POSSIBLE DOS DUE TO COLLATERALMANAGER.COLLATERALSUPPORT SIZE - INFORMATIONAL (1.0)

Description:

The owner of the CollateralManager.sol contract can add new collateral tokens which will be supported by the protocol. When adding support for new collaterals, there is no limit for the current amount of collaterals supported, and as the addresses of the collaterals are pushed to an array (collateralSupport), the size of this array can grow considerably over time.

Hence, when the protocol calls priceUpdate() to update the price of all collaterals supported by the protocol, it iterates over all the collaterals fetching their price from their oracles. In the case the size of the array has grown significantly, it could be possible the price update will revert due to reaching the transaction gas limit.

Code Location:

Listing 5: CollateralManager.sol (Line 127)

```
function addCollateral(
    address _collateral,
    address _oracle,
    address _eTokenAddress,
    uint256 _ratio
) external override onlyOwner {
    require(!getIsSupport(_collateral), Errors.CM_COLL_EXISTS);
    _requireRatioLegal(_ratio);

    collateralParams[_collateral] = DataTypes.CollateralParams(
        _ratio,
        _eTokenAddress,
        _oracle,
        DataTypes.CollStatus(1),
        collateralsCount
```

```
    );
    collateralSupport.push(_collateral);
    collateralsCount = collateralsCount.add(1);
}
```

Listing 6: CollateralManager.sol (Line 246)

```
242 function priceUpdate() public override {
243     if (collateralsCount < 2) {
244         return;
245     }
246     for (uint256 i = 1; i < collateralsCount; ) {
247         IOracle(collateralParams[collateralSupport[i]].oracle).
    ↳ fetchPrice();
248         unchecked {
249             i++;
250         }
251     }
252 }
```

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:C/D:N/Y:N/R:P/S:U (1.0)

Recommendation:

It is strongly recommended to set a cap for the amount of collaterals supported by the protocol.

Remediation Plan:

SOLVED: The ERD team solved the issue with the following commit ID.

Commit ID : 93c803ae22a7676e05a1fa6ec884589de28fd619

4.4 (HAL-04) MISSING A CAP FOR EUSD GAS COMPENSATION - INFORMATIONAL (0.0)

Description:

The owner of the CollateralManager.sol contract, when setting the EUSD_GAS_COMPENSATION protocol parameter within setGasCompensation() function, there are no checks regarding the quantity being set.

Code Location:

Listing 7: CollateralManager.sol (Line 756)

```
function setGasCompensation(uint256 _gas) external override
    onlyOwner {
    EUSD_GAS_COMPENSATION = _gas;
}
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:P/S:U (0.0)

Recommendation:

An important parameter for the protocol as is EUSD_GAS_COMPENSATION, which plays a vital role, it is strongly recommended to set a cap for it.

Remediation Plan:

SOLVED: The ERD team solved the issue with the following commit ID.

Commit ID : 1961c7fc04181f770468d575b5b402a07b8ab239

4.5 (HAL-05) LONG LITERAL UINT256 USED IN COLLATERALMANAGER - INFORMATIONAL (0.0)

Description:

Critical protocol parameters are set within the initialize() function of CollateralManager.sol contract. Specifically, MCR (minimum collateral ratio) and CCR (critical collateral ratio) are set using a long literal. This can lead to confusion on the percentages configured for the correct functionality of the whole protocol.

Code Location:

Listing 8: CollateralManager.sol (Lines 63,64)

```
60 function initialize() public initializer {
61     __Ownable_init();
62     BOOTSTRAP_PERIOD = 14 days;
63     MCR = 1100000000000000000; // 110%
64     CCR = 1300000000000000000; // 130%
65     EUSD_GAS_COMPENSATION = 200e18;
66     MIN_NET_DEBT = 1800e18;
67     BORROWING_FEE_FLOOR = (DECIMAL_PRECISION / 10000) * 75; //
   ↳ 0.75%
68
69     REDEMPTION_FEE_FLOOR = (DECIMAL_PRECISION / 10000) * 75; //
   ↳ 0.75%
70     RECOVERY_FEE = (DECIMAL_PRECISION / 10000) * 25; // 0.25%
71     MAX_BORROWING_FEE = (DECIMAL_PRECISION / 100) * 5; // 5%
72 }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:P/S:U (0.0)

Recommendation:

To avoid any confusion or human errors while setting those parameters, the use of exponentiation (11e17, 13e17, etc.) is recommended instead.

Remediation Plan:

SOLVED: The ERD team solved the issue with the following commit ID.

Commit ID : 0aaf1539e5897aca96034f20f82a0ec1a8d45182

4.6 (HAL-06) MISSING REQUIREISCONTRACT CHECK IN COLLATERALMANAGER.SETADDRESSES() - INFORMATIONAL (0.0)

Description:

In the CollateralManager.sol contract, when setting the contract addresses using setAddresses(), the _requireIsContract() check is missing for _troveManagerRedemptionsAddress.

Code Location:

Listing 9: CollateralManager.sol (Line 98)

```
function setAddresses(
    address _activePoolAddress,
    address _borrowerOperationsAddress,
    address _defaultPoolAddress,
    address _priceFeedAddress,
    address _troveManagerAddress,
    address _troveManagerRedemptionsAddress,
    address _wethAddress
) external override onlyOwner {
    _requireIsContract(_activePoolAddress);
    _requireIsContract(_borrowerOperationsAddress);
    _requireIsContract(_defaultPoolAddress);
    _requireIsContract(_priceFeedAddress);
    _requireIsContract(_wethAddress);
    _requireIsContract(_troveManagerAddress);

    borrowerOperationsAddress = _borrowerOperationsAddress;
    activePool = IActivePool(_activePoolAddress);
    defaultPool = IDefaultPool(_defaultPoolAddress);
    priceFeed = IPriceFeed(_priceFeedAddress);
    wethAddress = _wethAddress;

    troveManager = ITroveManager(_troveManagerAddress);

```

```
    troveManagerRedemptionsAddress = _troveManagerRedemptionsAddress;

    emit ActivePoolAddressChanged(_activePoolAddress);
    emit BorrowerOperationsAddressChanged(_borrowerOperationsAddress);
    emit DefaultPoolAddressChanged(_defaultPoolAddress);
    emit PriceFeedAddressChanged(_priceFeedAddress);
    emit TroveManagerAddressChanged(_troveManagerAddress);
    emit TroveManagerRedemptionsAddressChanged(
        _troveManagerRedemptionsAddress
    );
    emit WETHAddressChanged(_wethAddress);
}
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:P/S:U (0.0)

Recommendation:

When the address needs to be a contract, as is the case with _troveManagerRedemptionsAddress, it is recommended to check it as done for the rest of the parameters.

Remediation Plan:

SOLVED: The ERD team solved the issue with the following commit ID.

Commit ID : f77899108075aef9f90e1e31ab4e1ab22c20d89c

MANUAL TESTING

The main goal of the manual testing performed during this assessment was to test all the functionalities regarding the ERD stablecoin overall protocol, focusing on the following points/scenarios:

1. Tests focused on borrowing EUSD and adding collateral to the troves (as a borrower of the protocol and using multiple collateral tokens)

- Open a new trove with ERC20 tokens as collateral.
- Open a new trove with ETH as collateral.
- Add more collateral to the troves.
- Check how the new ICR is calculated.
- Check how the new index of the trove is calculated and reinserted.

2. Tests focused on repaying EUSD and withdrawing collateral from the troves (as a borrower of the protocol and using multiple collateral tokens)

- Repay 50% of the trove debt.
- Withdraw some collateral from the trove.
- Check how the new ICR is calculated.
- Check how the new index of the trove is calculated and reinserted.
- Check if exist a situation where the borrower cannot repay the debt.

```
├─ [408] collateralManager::MIN_NET_DEBT() [staticcall]
│   └─ ← 1800000000000000000000
├─ [605] eUSDToken::balanceOf(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]) [staticcall]
│   └─ ← 2000000000000000000000
├─ [227628] borrowerOperations::repayEUSD(20000000000000000000, borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436], borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436])
│   ├─ [247] priceFeedTestnet::fetchPrice()
│   │   └─ ← 2000000000000000000000
│   ├─ [446] collateralManager::priceUpdate()
│   │   └─ ← ()
│   ├─ [14287] troveManager::checkRecoveryMode(2000000000000000000000) [staticcall]
│   │   ├─ [2114] troveDebt::totalSupply() [staticcall]
│   │   │   ├─ [674] troveManager::getTroveNormalizedDebt() [staticcall]
│   │   │   │   └─ ← 1000000000000000000000000000
│   │   │   └─ ← 2015000000000000000000
│   │   ├─ [337] defaultPool::getEUSDDebt() [staticcall]
│   │   │   └─ ← 0
│   │   ├─ [605] eUSDToken::balanceOf(gasPool: [0x9A743489af33E59C9CD78203b311596F80ea36A9]) [staticcall]
│   │   │   └─ ← 200000000000000000000
│   │   ├─ [5755] collateralManager::getEntireCollValue(2000000000000000000000) [staticcall]
│   │   │   ├─ [398] eToken::totalSupply() [staticcall]
│   │   │   │   └─ ← 16000000000000000000
│   │   │   ├─ [534] WETH::balanceOf(defaultPool: [0xC6f8f4FA75A4F1f4Ff289c000AB71Bb2ea26b9B8]) [staticcall]
│   │   │   │   └─ ← 0
│   │   │   └─ ← [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2], [16000000000000000000], 32000000000000000000000
│   │   ├─ [426] collateralManager::getCCR() [staticcall]
│   │   │   └─ ← 1300000000000000000
│   │   └─ ← false
│   ├─ [664] troveManager::getTroveStatus(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]) [staticcall]
│   │   └─ ← 1
│   ├─ [4178] troveManager::applyPendingRewards(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436])
│   │   ├─ [1120] collateralManager::getCollateralSupport() [staticcall]
│   │   │   └─ ← [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2]
│   │   └─ ← ()
│   ├─ [9074] troveManager::getTroveColls(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]) [staticcall]
│   │   ├─ [5546] collateralManager::getTroveColls(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]) [staticcall]
│   │   │   ├─ [660] eToken::balanceOf(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]) [staticcall]
│   │   │   │   └─ ← 16000000000000000000
│   │   │   ├─ [669] eToken::sharesOf(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]) [staticcall]
│   │   │   │   └─ ← 16000000000000000000
│   │   │   └─ ← [16000000000000000000], [16000000000000000000], [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2]
│   │   └─ ← [16000000000000000000], [16000000000000000000], [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2]
│   ├─ [3036] collateralManager::getValue([0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2], [16000000000000000000], 2000000000000000000000) [staticcall]
│   │   └─ ← 32000000000000000000000, [32000000000000000000000]
│   ├─ [2457] collateralManager::mintEToken([], [], borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436], 2000000000000000000000)
│   │   └─ ← [], 0
│   ├─ [2434] collateralManager::burnEToken([], [], borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436], 2000000000000000000000)
│   │   └─ ← [], 0
│   ├─ [4645] troveManager::getTroveDebt(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]) [staticcall]
│   │   ├─ [403] collateralManager::getEUSDGasCompensation() [staticcall]
│   │   │   └─ ← 200000000000000000000
│   │   ├─ [2435] troveDebt::balanceOf(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]) [staticcall]
│   │   │   ├─ [674] troveManager::getTroveNormalizedDebt() [staticcall]
│   │   │   │   └─ ← 1000000000000000000000000000
│   │   │   └─ ← 2015000000000000000000
│   │   └─ ← 2215000000000000000000
│   ├─ [383] collateralManager::getMCR() [staticcall]
│   │   └─ ← 1100000000000000000
│   ├─ [5755] collateralManager::getEntireCollValue(2000000000000000000000) [staticcall]
│   │   ├─ [398] eToken::totalSupply() [staticcall]
│   │   │   └─ ← 16000000000000000000
│   │   ├─ [534] WETH::balanceOf(defaultPool: [0xC6f8f4FA75A4F1f4Ff289c000AB71Bb2ea26b9B8]) [staticcall]
│   │   │   └─ ← 0
│   │   └─ ← [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2], [16000000000000000000], 32000000000000000000000
│   ├─ [2114] troveDebt::totalSupply() [staticcall]
│   │   ├─ [674] troveManager::getTroveNormalizedDebt() [staticcall]
│   │   │   └─ ← 1000000000000000000000000000
│   │   └─ ← 2015000000000000000000
│   ├─ [337] defaultPool::getEUSDDebt() [staticcall]
│   │   └─ ← 0
│   ├─ [605] eUSDToken::balanceOf(gasPool: [0x9A743489af33E59C9CD78203b311596F80ea36A9]) [staticcall]
│   │   └─ ← 200000000000000000000
```

3. Tests focused on providing liquidity to the stability pool (as an SP depositor of the protocol)

- Only one depositor on the system as liquidity provider to the stability pool.
- More than one depositor as liquidity providers in the protocol.
- Check the flow of the EUSD and collaterals tokens when active troves are updated.
- Check the flow of the EUSD and collaterals tokens when a trove is liquidated.

4. Tests focused on withdrawing liquidity from the stability pool (as an SP depositor of the protocol)

- Depositor of EUSD withdrawing liquidity from the stability pool.
- Check how the TCR is being affected.
- Check how the shares of depositors are being recalculated.

```
├─ [82034] stabilityPool::withdrawFromSP(2000000000000000000000)
│   ├─ [247] priceFeedTestnet::fetchPrice()
│   │   └─ ← 200000000000000000000
│   ├─ [446] collateralManager::priceUpdate()
│   │   └─ ← ()
│   ├─ [420] sortedTroves::getLast() [staticcall]
│   │   └─ ← borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]
│   ├─ [25706] troveManager::getCurrentICR(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436], 200000000000000000000) [staticcall]
│   │   ├─ [1120] collateralManager::getCollateralSupport() [staticcall]
│   │   │   └─ ← [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2]
│   │   ├─ [5546] collateralManager::getTroveColls(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]) [staticcall]
│   │   │   ├─ [660] eToken::balanceOf(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]) [staticcall]
│   │   │   │   └─ ← 16000000000000000000
│   │   │   ├─ [669] eToken::sharesOf(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]) [staticcall]
│   │   │   │   └─ ← 16000000000000000000
│   │   │   └─ ← [16000000000000000000], [16000000000000000000], [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2]
│   │   ├─ [1120] collateralManager::getCollateralSupport() [staticcall]
│   │   │   └─ ← [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2]
│   │   ├─ [403] collateralManager::getEUSDGasCompensation() [staticcall]
│   │   │   └─ ← 200000000000000000000
│   │   ├─ [2435] troveDebt::balanceOf(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]) [staticcall]
│   │   │   ├─ [674] troveManager::getTroveNormalizedDebt() [staticcall]
│   │   │   │   └─ ← 1000000000000000000000000000
│   │   │   └─ ← 2015000000000000000000
│   │   ├─ [3036] collateralManager::getValue([0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2], [16000000000000000000], 200000000000000000000) [staticcall]
│   │   │   └─ ← 3200000000000000000000, [3200000000000000000000]
│   │   └─ ← 1444695259593679458
│   ├─ [383] collateralManager::getMCR() [staticcall]
│   │   └─ ← 1100000000000000000
│   ├─ [243] communityIssuance::issue()
│   │   └─ ← 0
│   ├─ [1120] collateralManager::getCollateralSupport() [staticcall]
│   │   └─ ← [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2]
│   ├─ [2102] collateralManager::getAmounts([0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2], [0]) [staticcall]
│   │   └─ ← [0]
│   ├─ [315] communityIssuance::trigger(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436], 0)
│   │   └─ ← ()
│   ├─ emit GainPaidToDepositor(_depositor: borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436], _GAIN: 0)
│   ├─ emit FrontEndSnapshotUpdated(_frontEnd: 0x0000000000000000000000000000000000000000, _P: 0, _G: 0)
│   ├─ emit FrontEndStakeChanged(_frontEnd: 0x0000000000000000000000000000000000000000, _newFrontEndStake: 0, _depositor: borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436])
│   ├─ [20297] eUSDToken::returnFromPool(stabilityPool: [0xC30190a04DBFaF2FcbC4660E7baC8f8347280a17], borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436], 2000000000000000000000)
│   │   ├─ emit Transfer(from: stabilityPool: [0xC30190a04DBFaF2FcbC4660E7baC8f8347280a17], to: borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436], value: 2000000000000000000000)
│   │   └─ ← ()
│   ├─ emit StabilityPoolEUSDBalanceUpdated(_newBalance: 0)
│   ├─ [1120] collateralManager::getCollateralSupport() [staticcall]
│   │   └─ ← [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2]
│   ├─ emit DepositSnapshotUpdated(_depositor: borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436], _P: 0, _S: [], _G: 0)
│   ├─ emit UserDepositChanged(_depositor: borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436], _newDeposit: 0)
│   ├─ emit CollGainWithdrawn(_depositor: borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436], _collAmounts: [0], _EUSDLoss: 0)
│   └─ ← ()
├─ [0] console::log(################################################################) [staticcall]
│   └─ ← ()
├─ [247] priceFeedTestnet::fetchPrice() [staticcall]
│   └─ ← 200000000000000000000
├─ [9074] troveManager::getTroveColls(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]) [staticcall]
│   ├─ [5546] collateralManager::getTroveColls(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]) [staticcall]
│   │   ├─ [660] eToken::balanceOf(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]) [staticcall]
│   │   │   └─ ← 16000000000000000000
│   │   ├─ [669] eToken::sharesOf(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]) [staticcall]
│   │   │   └─ ← 16000000000000000000
│   │   └─ ← [16000000000000000000], [16000000000000000000], [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2]
│   └─ ← [16000000000000000000], [16000000000000000000], [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2]
├─ [3036] collateralManager::getValue([0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2], [16000000000000000000], 200000000000000000000) [staticcall]
│   └─ ← 3200000000000000000000, [3200000000000000000000]
├─ [4645] troveManager::getTroveDebt(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]) [staticcall]
│   ├─ [403] collateralManager::getEUSDGasCompensation() [staticcall]
│   │   └─ ← 200000000000000000000
│   ├─ [2435] troveDebt::balanceOf(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]) [staticcall]
│   │   ├─ [674] troveManager::getTroveNormalizedDebt() [staticcall]
│   │   │   └─ ← 1000000000000000000000000000
│   │   └─ ← 2015000000000000000000
```

MANUAL TESTING

5. Tests focused on liquidations (checking the mode of the protocol and the different scenarios for closing the troves and distributing the rewards)

- Basic checks when ICR < MCR, the trove can be liquidated.
- Basic checks when ICR > MCR and TCR < CCR, the trove can be liquidated as well.

```
├─ [3181] priceFeedTestnet::setPrice(1500000000000000000000)
│   └─ ← true
├─ [0] VM::startPrank(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436])
│   └─ ← ()
├─ [423185] troveManager::liquidate(borrower2: [0xD83d18A2EaE2440e272a53f86e617cd9F33c8D68])
│   ├─ [398615] troveManagerLiquidations::batchLiquidateTroves([0xD83d18A2EaE2440e272a53f86e617cd9F33c8D68], borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436])
│   │   ├─ [247] priceFeedTestnet::fetchPrice()
│   │   │   └─ ← 1500000000000000000000
│   │   ├─ [446] collateralManager::priceUpdate()
│   │   │   └─ ← ()
│   │   ├─ [2404] stabilityPool::getTotalEUSDDeposits() [staticcall]
│   │   │   └─ ← 0
│   │   ├─ [14287] troveManager::checkRecoveryMode(1500000000000000000000) [staticcall]
│   │   │   ├─ [2114] troveDebt::totalSupply() [staticcall]
│   │   │   │   ├─ [674] troveManager::getTroveNormalizedDebt() [staticcall]
│   │   │   │   │   └─ ← 1000000000000000000000000000
│   │   │   │   └─ ← 8060000000000000000000
│   │   │   ├─ [337] defaultPool::getEUSDDebt() [staticcall]
│   │   │   │   └─ ← 0
│   │   │   ├─ [605] eUSDToken::balanceOf(gasPool: [0x9A743489af33E59C9CD78203b311596F80ea36A9]) [staticcall]
│   │   │   │   └─ ← 400000000000000000000
│   │   │   ├─ [5755] collateralManager::getEntireCollValue(1500000000000000000000) [staticcall]
│   │   │   │   ├─ [398] eToken::totalSupply() [staticcall]
│   │   │   │   │   └─ ← 60000000000000000000
│   │   │   │   ├─ [534] WETH::balanceOf(defaultPool: [0xC6f8f4FA75A4F1f4Ff289c000AB71Bb2ea26b9B8]) [staticcall]
│   │   │   │   │   └─ ← 0
│   │   │   │   └─ ← [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2], [60000000000000000000], 90000000000000000000000
│   │   │   ├─ [426] collateralManager::getCCR() [staticcall]
│   │   │   │   └─ ← 1300000000000000000
│   │   │   └─ ← true
│   │   ├─ [2114] troveDebt::totalSupply() [staticcall]
│   │   │   ├─ [674] troveManager::getTroveNormalizedDebt() [staticcall]
│   │   │   │   └─ ← 1000000000000000000000000000
│   │   │   └─ ← 8060000000000000000000
│   │   ├─ [337] defaultPool::getEUSDDebt() [staticcall]
│   │   │   └─ ← 0
│   │   ├─ [605] eUSDToken::balanceOf(gasPool: [0x9A743489af33E59C9CD78203b311596F80ea36A9]) [staticcall]
│   │   │   └─ ← 400000000000000000000
│   │   ├─ [6015] activePool::getTotalCollateral() [staticcall]
│   │   │   ├─ [2788] troveManager::getCollateralSupport() [staticcall]
│   │   │   │   ├─ [1120] collateralManager::getCollateralSupport() [staticcall]
│   │   │   │   │   └─ ← [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2]
│   │   │   │   └─ ← [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2]
│   │   │   ├─ [534] WETH::balanceOf(activePool: [0x644aD51e26E1A66a13b7FBF110cf721E688A89a2]) [staticcall]
│   │   │   │   └─ ← 60000000000000000000
│   │   │   └─ ← 60000000000000000000, [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2], [60000000000000000000]
│   │   ├─ [6077] defaultPool::getTotalCollateral() [staticcall]
│   │   │   ├─ [2788] troveManager::getCollateralSupport() [staticcall]
│   │   │   │   ├─ [1120] collateralManager::getCollateralSupport() [staticcall]
│   │   │   │   │   └─ ← [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2]
│   │   │   │   └─ ← [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2]
│   │   │   ├─ [534] WETH::balanceOf(defaultPool: [0xC6f8f4FA75A4F1f4Ff289c000AB71Bb2ea26b9B8]) [staticcall]
│   │   │   │   └─ ← 0
│   │   │   └─ ← 0, [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2], [0]
│   │   ├─ [3036] collateralManager::getValue([0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2], [60000000000000000000], 1500000000000000000000) [staticcall]
│   │   │   └─ ← 90000000000000000000000, [90000000000000000000000]
│   │   ├─ [383] collateralManager::getMCR() [staticcall]
│   │   │   └─ ← 1100000000000000000
│   │   ├─ [664] troveManager::getTroveStatus(borrower2: [0xD83d18A2EaE2440e272a53f86e617cd9F33c8D68]) [staticcall]
│   │   │   └─ ← 1
│   │   ├─ [25706] troveManager::getCurrentICR(borrower2: [0xD83d18A2EaE2440e272a53f86e617cd9F33c8D68], 1500000000000000000000) [staticcall]
│   │   │   ├─ [1120] collateralManager::getCollateralSupport() [staticcall]
│   │   │   │   └─ ← [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2]
│   │   │   ├─ [5546] collateralManager::getTroveColls(borrower2: [0xD83d18A2EaE2440e272a53f86e617cd9F33c8D68]) [staticcall]
│   │   │   │   ├─ [660] eToken::balanceOf(borrower2: [0xD83d18A2EaE2440e272a53f86e617cd9F33c8D68]) [staticcall]
│   │   │   │   │   └─ ← 30000000000000000000
│   │   │   │   ├─ [669] eToken::sharesOf(borrower2: [0xD83d18A2EaE2440e272a53f86e617cd9F33c8D68]) [staticcall]
│   │   │   │   │   └─ ← 30000000000000000000
│   │   │   │   └─ ← [30000000000000000000], [30000000000000000000], [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2]
│   │   │   ├─ [1120] collateralManager::getCollateralSupport() [staticcall]
│   │   │   │   └─ ← [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2]
```

MANUAL TESTING

6. Tests depending on the current mode of the protocol (normal or recovery mode, to check which protocol actions are permitted and not permitted during recovery mode)

- Check if actions that mint EUSD are not permitted during recovery mode.
- Check if actions that burn EUSD are incentivized for users during recovery mode.

```
├─ [3181] priceFeedTestnet::setPrice(1500000000000000000000)
│   └─ ← true
├─ [0] VM::startPrank(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436])
│   └─ ← ()
├─ [70288] borrowerOperations::withdrawColl([0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2], [1000000000000000000], borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436], borrower: [0x05650444aCe15A01762Bd97EE8fDeb49
│   ├─ [737] collateralManager::getIsSupport(WETH: [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2]) [staticcall]
│   │   └─ ← true
│   ├─ [3269] collateralManager::getShares([0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2], [1000000000000000000]) [staticcall]
│   │   ├─ [335] eToken::getShare(1000000000000000000) [staticcall]
│   │   │   └─ ← 1000000000000000000
│   │   └─ ← [1000000000000000000]
│   ├─ [247] priceFeedTestnet::fetchPrice()
│   │   └─ ← 1500000000000000000000
│   ├─ [446] collateralManager::priceUpdate()
│   │   └─ ← ()
│   ├─ [14287] troveManager::checkRecoveryMode(1500000000000000000000) [staticcall]
│   │   ├─ [2114] troveDebt::totalSupply() [staticcall]
│   │   │   ├─ [674] troveManager::getTroveNormalizedDebt() [staticcall]
│   │   │   │   └─ ← 1000000000000000000000000000
│   │   │   └─ ← 8060000000000000000000
│   │   ├─ [337] defaultPool::getEUSDDebt() [staticcall]
│   │   │   └─ ← 0
│   │   ├─ [605] eUSDToken::balanceOf(gasPool: [0x9A743489af33E59C9CD78203b311596F80ea36A9]) [staticcall]
│   │   │   └─ ← 400000000000000000000
│   │   ├─ [5755] collateralManager::getEntireCollValue(1500000000000000000000) [staticcall]
│   │   │   ├─ [398] eToken::totalSupply() [staticcall]
│   │   │   │   └─ ← 6000000000000000000
│   │   │   ├─ [534] WETH::balanceOf(defaultPool: [0xC6f8f4FA75A4F1f4Ff289c000AB71Bb2ea26b9B8]) [staticcall]
│   │   │   │   └─ ← 0
│   │   │   └─ ← [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2], [6000000000000000000], 9000000000000000000000
│   │   ├─ [426] collateralManager::getCCR() [staticcall]
│   │   │   └─ ← 1300000000000000000
│   │   └─ ← true
│   ├─ [664] troveManager::getTroveStatus(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]) [staticcall]
│   │   └─ ← 1
│   ├─ [4178] troveManager::applyPendingRewards(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436])
│   │   ├─ [1120] collateralManager::getCollateralSupport() [staticcall]
│   │   │   └─ ← [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2]
│   │   └─ ← ()
│   ├─ [9074] troveManager::getTroveColls(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]) [staticcall]
│   │   ├─ [5546] collateralManager::getTroveColls(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]) [staticcall]
│   │   │   ├─ [660] eToken::balanceOf(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]) [staticcall]
│   │   │   │   └─ ← 3000000000000000000
│   │   │   ├─ [669] eToken::sharesOf(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]) [staticcall]
│   │   │   │   └─ ← 3000000000000000000
│   │   │   └─ ← [3000000000000000000], [3000000000000000000], [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2]
│   │   └─ ← [3000000000000000000], [3000000000000000000], [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2]
│   ├─ [3036] collateralManager::getValue([0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2], [3000000000000000000], 1500000000000000000000) [staticcall]
│   │   └─ ← 4500000000000000000000, [4500000000000000000000]
│   ├─ [2457] collateralManager::mintEToken([], [], borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436], 1500000000000000000000)
│   │   └─ ← [], 0
│   ├─ [8749] collateralManager::burnEToken([0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2], [1000000000000000000], borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436], 1500000000000000000000)
│   │   ├─ [4112] eToken::burn(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436], 1000000000000000000)
│   │   │   ├─ emit Transfer(from: borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436], to: 0x0000000000000000000000000000000000000000, value: 1000000000000000000)
│   │   │   └─ ← 1000000000000000000
│   │   └─ ← [1000000000000000000], 1500000000000000000000
│   ├─ [4645] troveManager::getTroveDebt(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]) [staticcall]
│   │   ├─ [403] collateralManager::getEUSDGasCompensation() [staticcall]
│   │   │   └─ ← 200000000000000000000
│   │   ├─ [2435] troveDebt::balanceOf(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]) [staticcall]
│   │   │   ├─ [674] troveManager::getTroveNormalizedDebt() [staticcall]
│   │   │   │   └─ ← 1000000000000000000000000000
│   │   │   └─ ← 4030000000000000000000
│   │   └─ ← 4230000000000000000000
│   └─ ← "16"
└─ ← "16"
```

7. Combine and perform integration tests with all the critical functionalities within the protocol (borrowers, depositors, liquidators, redeemers)

- Set 4 depositors as liquidity providers to the stability pool.
- Set 4 borrowers of EUSD.
- Set 2 redeemers of EUSD.
- Change the price of the collateral in the oracle.
- Redeemers redeem collateral.
- Troves being liquidated.
- Check the flow of EUSD and collateral tokens and check the priority collateral logic is properly working.

8. Deeply test all the possible cases of the system state and a specific trove being liquidated to ensure distributions of collaterals and rewards are correctly done as explained within the documentation (between depositors, active troves and the different pools).

- ICR < MCR & SP.EUSD >= trove.debt & TCR >= 130%
- ICR < MCR & SP.EUSD < trove.debt & TCR >= 130%
- ICR < MCR & SP.EUSD = 0 & TCR >= 130%
- ICR <=100% & TCR < 130%
- 100% < ICR < MCR & SP.EUSD > trove.debt & TCR < 130%
- 100% < ICR < MCR & SP.EUSD < trove.debt & TCR < 130%
- MCR <= ICR < TCR & SP.EUSD >= trove.debt & TCR < 130%

```
├─ [423907] troveManager::liquidateTroves(1)
│   ├─ [400096] troveManagerLiquidations::liquidateTroves(1, borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436])
│   │   ├─ [1120] collateralManager::getCollateralSupport() [staticcall]
│   │   │   └─ ← [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2]
│   │   ├─ [247] priceFeedTestnet::fetchPrice()
│   │   │   └─ ← 1500000000000000000000
│   │   ├─ [446] collateralManager::priceUpdate()
│   │   │   └─ ← ()
│   │   ├─ [2404] stabilityPool::getTotalEUSDDeposits() [staticcall]
│   │   │   └─ ← 0
│   │   ├─ [14287] troveManager::checkRecoveryMode(1500000000000000000000) [staticcall]
│   │   │   ├─ [2114] troveDebt::totalSupply() [staticcall]
│   │   │   │   ├─ [674] troveManager::getTroveNormalizedDebt() [staticcall]
│   │   │   │   │   └─ ← 1000000000000000000000000000
│   │   │   │   └─ ← 8060000000000000000000
│   │   │   ├─ [337] defaultPool::getEUSDDebt() [staticcall]
│   │   │   │   └─ ← 0
│   │   │   ├─ [605] eUSDToken::balanceOf(gasPool: [0x9A743489af33E59C9CD78203b311596F80ea36A9]) [staticcall]
│   │   │   │   └─ ← 400000000000000000000
│   │   │   ├─ [5755] collateralManager::getEntireCollValue(1500000000000000000000) [staticcall]
│   │   │   │   ├─ [398] eToken::totalSupply() [staticcall]
│   │   │   │   │   └─ ← 6000000000000000000
│   │   │   │   ├─ [534] WETH::balanceOf(defaultPool: [0xC6f8f4FA75A4F1f4Ff289c000AB71Bb2ea26b9B8]) [staticcall]
│   │   │   │   │   └─ ← 0
│   │   │   │   └─ ← [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2], [6000000000000000000], 9000000000000000000000
│   │   │   ├─ [426] collateralManager::getCCR() [staticcall]
│   │   │   │   └─ ← 1300000000000000000
│   │   │   └─ ← true
│   │   ├─ [2114] troveDebt::totalSupply() [staticcall]
│   │   │   ├─ [674] troveManager::getTroveNormalizedDebt() [staticcall]
│   │   │   │   └─ ← 1000000000000000000000000000
│   │   │   └─ ← 8060000000000000000000
│   │   ├─ [337] defaultPool::getEUSDDebt() [staticcall]
│   │   │   └─ ← 0
│   │   ├─ [605] eUSDToken::balanceOf(gasPool: [0x9A743489af33E59C9CD78203b311596F80ea36A9]) [staticcall]
│   │   │   └─ ← 400000000000000000000
│   │   ├─ [6015] activePool::getTotalCollateral() [staticcall]
│   │   │   ├─ [2788] troveManager::getCollateralSupport() [staticcall]
│   │   │   │   ├─ [1120] collateralManager::getCollateralSupport() [staticcall]
│   │   │   │   │   └─ ← [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2]
│   │   │   │   └─ ← [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2]
│   │   │   ├─ [534] WETH::balanceOf(activePool: [0x644aD51e26E1A66a13b7FBF110cf721E688A89a2]) [staticcall]
│   │   │   │   └─ ← 6000000000000000000
│   │   │   └─ ← 6000000000000000000, [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2], [6000000000000000000]
│   │   ├─ [6077] defaultPool::getTotalCollateral() [staticcall]
│   │   │   ├─ [2788] troveManager::getCollateralSupport() [staticcall]
│   │   │   │   ├─ [1120] collateralManager::getCollateralSupport() [staticcall]
│   │   │   │   │   └─ ← [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2]
│   │   │   │   └─ ← [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2]
│   │   │   ├─ [534] WETH::balanceOf(defaultPool: [0xC6f8f4FA75A4F1f4Ff289c000AB71Bb2ea26b9B8]) [staticcall]
│   │   │   │   └─ ← 0
│   │   │   └─ ← 0, [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2], [0]
│   │   ├─ [3036] collateralManager::getValue([0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2], [6000000000000000000], 1500000000000000000000) [staticcall]
│   │   │   └─ ← 9000000000000000000000, [9000000000000000000000]
│   │   ├─ [420] sortedTroves::getLast() [staticcall]
│   │   │   └─ ← borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]
│   │   ├─ [366] sortedTroves::getFirst() [staticcall]
│   │   │   └─ ← borrower2: [0xD83d18A2EaE2440e272a53f86e617cd9F33c8D68]
│   │   ├─ [383] collateralManager::getMCR() [staticcall]
│   │   │   └─ ← 1100000000000000000
│   │   ├─ [621] sortedTroves::getPrev(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]) [staticcall]
│   │   │   └─ ← borrower2: [0xD83d18A2EaE2440e272a53f86e617cd9F33c8D68]
│   │   ├─ [25706] troveManager::getCurrentICR(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436], 1500000000000000000000) [staticcall]
│   │   │   ├─ [1120] collateralManager::getCollateralSupport() [staticcall]
│   │   │   │   └─ ← [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2]
│   │   │   ├─ [5546] collateralManager::getTroveColls(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]) [staticcall]
│   │   │   │   ├─ [660] eToken::balanceOf(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]) [staticcall]
│   │   │   │   │   └─ ← 3000000000000000000
│   │   │   │   ├─ [669] eToken::sharesOf(borrower: [0x05650444aCe15A01762Bd97EE8fDeb495b3C2436]) [staticcall]
│   │   │   │   │   └─ ← 3000000000000000000
│   │   │   │   └─ ← [3000000000000000000], [3000000000000000000], [0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2]
```

9. Tests focused on using multiple collaterals instead of just one type, and analyze how the protocol handles the different user actions involved

- Add support for multiple collateral tokens.
- Check how the system handles when priority collateral logic is executed.
- Check how collaterals are calculated when opening a trove.
- Check how collaterals are recalculated when adding a specific collateral to an already active trove.
- Check how the system merges the ETH with collaterals into the protocol collaterals array.

10. Moreover, and very importantly, theoretically review all cases to make sure contracts have the correct business logic for the proper functionality of the stablecoin overall protocol

AUTOMATED TESTING

6.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the scoped contracts. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their ABI and binary formats, Slither was run on the all-scoped contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Slither results:

contracts/ActivePool.sol

```
ActivePool._sendETH(address,uint256) (src/ActivePool.sol#216-227) sends eth to arbitrary user
	Dangerous calls:
	- (success) = _to.call{value: amount}() (src/ActivePool.sol#220)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations

ActivePool.sendCollateral(address,address[],uint256[]).hasETH (src/ActivePool.sol#155) is a local variable never initialized
ActivePool.sendCollateral(address,address[],uint256[]).ETHAmount (src/ActivePool.sol#156) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

ActivePool.setAddresses(address,address,address,address,address,address,address,address,address,address)._borrowerOperationsAddress (src/ActivePool.sol#49) lacks a zero-check on :
		- borrowerOperationsAddress = _borrowerOperationsAddress (src/ActivePool.sol#71)
ActivePool.setAddresses(address,address,address,address,address,address,address,address,address,address)._troveManagerAddress (src/ActivePool.sol#50) lacks a zero-check on :
		- troveManagerAddress = _troveManagerAddress (src/ActivePool.sol#72)
ActivePool.setAddresses(address,address,address,address,address,address,address,address,address,address)._troveManagerLiquidationAddress (src/ActivePool.sol#51) lacks a zero-check on :
		- troveManagerLiquidationsAddress = _troveManagerLiquidationAddress (src/ActivePool.sol#73)
ActivePool.setAddresses(address,address,address,address,address,address,address,address,address,address)._troveManagerRedemptionsAddress (src/ActivePool.sol#52) lacks a zero-check on :
		- troveManagerRedemptionsAddress = _troveManagerRedemptionsAddress (src/ActivePool.sol#74)
ActivePool.setAddresses(address,address,address,address,address,address,address,address,address,address)._stabilityPoolAddress (src/ActivePool.sol#53) lacks a zero-check on :
		- stabilityPoolAddress = _stabilityPoolAddress (src/ActivePool.sol#75)
ActivePool.setAddresses(address,address,address,address,address,address,address,address,address,address)._defaultPoolAddress (src/ActivePool.sol#54) lacks a zero-check on :
		- defaultPoolAddress = _defaultPoolAddress (src/ActivePool.sol#76)
ActivePool.setAddresses(address,address,address,address,address,address,address,address,address,address)._treasuryAddress (src/ActivePool.sol#55) lacks a zero-check on :
		- treasuryAddress = _treasuryAddress (src/ActivePool.sol#77)
ActivePool.setAddresses(address,address,address,address,address,address,address,address,address,address)._liquidityIncentiveAddress (src/ActivePool.sol#56) lacks a zero-check on :
		- liquidityIncentiveAddress = _liquidityIncentiveAddress (src/ActivePool.sol#78)
ActivePool.setAddresses(address,address,address,address,address,address,address,address,address,address)._collSurplusPoolAddress (src/ActivePool.sol#57) lacks a zero-check on :
		- collSurplusPoolAddress = _collSurplusPoolAddress (src/ActivePool.sol#79)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
```

contracts/BorrowerOperations.sol

```
BorrowerOperations._singleTransferCollateralIntoActivePool(address,address,uint256) (src/BorrowerOperations.sol#795-832) use msg.value in a loop: WETH.deposit{value: msg.value}() (src/BorrowerOperations.sol#802)
BorrowerOperations._singleTransferCollateralIntoActivePool(address,address,uint256) (src/BorrowerOperations.sol#795-832) use msg.value in a loop: WETH.transferFrom(address(this),address(activePool),msg.value) (src/
BorrowerOperations._singleTransferCollateralIntoActivePool(address,address,uint256) (src/BorrowerOperations.sol#795-832) use msg.value in a loop: _amount > msg.value (src/BorrowerOperations.sol#808)
BorrowerOperations._singleTransferCollateralIntoActivePool(address,address,uint256) (src/BorrowerOperations.sol#795-832) use msg.value in a loop: SafeERC20Upgradeable.safeTransferFrom(IERC20Upgradeable(address(WETH
c/BorrowerOperations.sol#809-814)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#msgvalue-inside-a-loop

BorrowerOperations._singleTransferCollateralIntoActivePool(address,address,uint256) (src/BorrowerOperations.sol#795-832) ignores return value by WETH.transferFrom(address(this),address(activePool),msg.value) (src/B
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

ERDMath.calculateCompoundedInterest(uint256,uint40,uint256) (src/Dependencies/ERDMath.sol#250-280) performs a multiplication on the result of a division:
	-ratePerSecond = rate / SECONDS_PER_YEAR (src/Dependencies/ERDMath.sol#266)
	-WadRayMath.ray().add(ratePerSecond.mul(exp)).add(secondTerm).add(thirdTerm) (src/Dependencies/ERDMath.sol#276-279)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

ERDMath.calculateCompoundedInterest(uint256,uint40,uint256) (src/Dependencies/ERDMath.sol#250-280) uses a dangerous strict equality:
	- exp == 0 (src/Dependencies/ERDMath.sol#258)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

BorrowerOperations._removeZero(address[],uint256[]).index (src/BorrowerOperations.sol#934) is a local variable never initialized
BorrowerOperations.moveCollGainToTrove(address,address[],uint256[],address,address).params (src/BorrowerOperations.sol#442) is a local variable never initialized
BorrowerOperations.withdrawEUSD(uint256,address,address,uint256).params (src/BorrowerOperations.sol#408) is a local variable never initialized
BorrowerOperations._removeZero(address[],uint256[]).count (src/BorrowerOperations.sol#926) is a local variable never initialized
BorrowerOperations._activePoolAddColl(address,address[],uint256[]).i (src/BorrowerOperations.sol#787) is a local variable never initialized
BorrowerOperations._adjustTrove(BorrowerOperations.AdjustTrove_Params).vars (src/BorrowerOperations.sol#536) is a local variable never initialized
ERDMath._arrayIsNonzero(uint256[]).i (src/Dependencies/ERDMath.sol#196) is a local variable never initialized
BorrowerOperations._requireValidOpenTroveCollateral(address[],uint256[],uint256).i (src/BorrowerOperations.sol#971) is a local variable never initialized
BorrowerOperations._adjustArray(address[],uint256[],uint256).hasWETH (src/BorrowerOperations.sol#313) is a local variable never initialized
BorrowerOperations._requireNoWETHColls(address[]).i (src/BorrowerOperations.sol#1023) is a local variable never initialized
BorrowerOperations._adjustArray(address[],uint256[],uint256).index (src/BorrowerOperations.sol#314) is a local variable never initialized
BorrowerOperations.openTrove(address[],uint256[],uint256,uint256,address,address).vars (src/BorrowerOperations.sol#179) is a local variable never initialized
BorrowerOperations._requireNoOverlapColls(address[],address[]).i (src/BorrowerOperations.sol#1014) is a local variable never initialized
BorrowerOperations._requireValidAdjustCollateralAmounts(address[],uint256[],uint256,bool).i (src/BorrowerOperations.sol#998) is a local variable never initialized
BorrowerOperations.repayEUSD(uint256,address,address).params (src/BorrowerOperations.sol#424) is a local variable never initialized
BorrowerOperations._requireNoDuplicateColls(address[]).i (src/BorrowerOperations.sol#1033) is a local variable never initialized
BorrowerOperations.addColl(address[],uint256[],address,address).params (src/BorrowerOperations.sol#348) is a local variable never initialized
ERDMath._getArrayCopy(uint256[]).i (src/Dependencies/ERDMath.sol#210) is a local variable never initialized
BorrowerOperations._requireNoOverlapColls(address[],address[]).j (src/BorrowerOperations.sol#1015) is a local variable never initialized
BorrowerOperations.withdrawColl(address[],uint256[],address,address).params (src/BorrowerOperations.sol#380) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

BorrowerOperations.openTrove(address[],uint256[],uint256,uint256,address,address) (src/BorrowerOperations.sol#168-292) ignores return value by contractsCache.troveManager.increaseTroveDebt(msg.sender,vars.netDebt)
BorrowerOperations.closeTrove() (src/BorrowerOperations.sol#692-756) ignores return value by collateralManagerCached.burnEToken(collaterals,collAmounts,msg.sender,0) (src/BorrowerOperations.sol#715-720)
BorrowerOperations._updateTroveFromAdjustment(ITroveManager,address,uint256,bool) (src/BorrowerOperations.sol#859-872) ignores return value by _troveManager.increaseTroveDebt(_borrower,_debtChange) (src/BorrowerOpe
BorrowerOperations._updateTroveFromAdjustment(ITroveManager,address,uint256,bool) (src/BorrowerOperations.sol#859-872) ignores return value by _troveManager.decreaseTroveDebt(_borrower,_debtChange) (src/BorrowerOpe
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

BorrowerOperations.setAddresses(address,address,address,address,address,address,address,address,address,address)._stabilityPoolAddress (src/BorrowerOperations.sol#107) lacks a zero-check on :
		- stabilityPoolAddress = _stabilityPoolAddress (src/BorrowerOperations.sol#129)
BorrowerOperations.setAddresses(address,address,address,address,address,address,address,address,address,address)._gasPoolAddress (src/BorrowerOperations.sol#108) lacks a zero-check on :
		- gasPoolAddress = _gasPoolAddress (src/BorrowerOperations.sol#130)
BorrowerOperations.init(address,address,address)._treasuryAddress (src/BorrowerOperations.sol#150) lacks a zero-check on :
		- treasuryAddress = _treasuryAddress (src/BorrowerOperations.sol#158)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
```

contracts/CollateralManager.sol

```
ERDBase.collateralManager (src/Dependencies/ERDBase.sol#29) is never initialized. It is used in:
	- ERDBase.getEntireSystemColl(uint256) (src/Dependencies/ERDBase.sol#98-104)
ERDBase.eusdToken (src/Dependencies/ERDBase.sol#32) is never initialized. It is used in:
	- ERDBase.getEntireSystemDebt() (src/Dependencies/ERDBase.sol#106-115)
ERDBase.troveDebt (src/Dependencies/ERDBase.sol#33) is never initialized. It is used in:
	- ERDBase.getEntireSystemDebt() (src/Dependencies/ERDBase.sol#106-115)
ERDBase.gasPoolAddress (src/Dependencies/ERDBase.sol#37) is never initialized. It is used in:
	- ERDBase.getEntireSystemDebt() (src/Dependencies/ERDBase.sol#106-115)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables

CollateralManager.initialize() (src/CollateralManager.sol#60-72) performs a multiplication on the result of a division:
	-BORROWING_FEE_FLOOR = (DECIMAL_PRECISION / 10000) * 75 (src/CollateralManager.sol#67)
CollateralManager.initialize() (src/CollateralManager.sol#60-72) performs a multiplication on the result of a division:
	-REDEMPTION_FEE_FLOOR = (DECIMAL_PRECISION / 10000) * 75 (src/CollateralManager.sol#69)
CollateralManager.initialize() (src/CollateralManager.sol#60-72) performs a multiplication on the result of a division:
	-RECOVERY_FEE = (DECIMAL_PRECISION / 10000) * 25 (src/CollateralManager.sol#70)
CollateralManager.initialize() (src/CollateralManager.sol#60-72) performs a multiplication on the result of a division:
	-MAX_BORROWING_FEE = (DECIMAL_PRECISION / 100) * 5 (src/CollateralManager.sol#71)
CollateralManager._calcValue(address,uint256,uint256) (src/CollateralManager.sol#292-311) performs a multiplication on the result of a division:
	-value = coll_adjust.fetchPrice_view().mul(valueRatio).div(DECIMAL_PRECISION).mul(_amount).div(DECIMAL_PRECISION).mul(_price).div(DECIMAL_PRECISION) (src/CollateralManager.sol#300-307)
ERDMath.calculateCompoundedInterest(uint256,uint40,uint256) (src/Dependencies/ERDMath.sol#250-280) performs a multiplication on the result of a division:
	-ratePerSecond = rate / SECONDS_PER_YEAR (src/Dependencies/ERDMath.sol#266)
	-WadRayMath.ray().add(ratePerSecond.mul(exp)).add(secondTerm).add(thirdTerm) (src/Dependencies/ERDMath.sol#276-279)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

ERDMath.calculateCompoundedInterest(uint256,uint40,uint256) (src/Dependencies/ERDMath.sol#250-280) uses a dangerous strict equality:
	- exp == 0 (src/Dependencies/ERDMath.sol#258)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

ERDMath._getArrayCopy(uint256[]).i (src/Dependencies/ERDMath.sol#210) is a local variable never initialized
ERDMath._arrayIsNonzero(uint256[]).i (src/Dependencies/ERDMath.sol#196) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

CollateralManager.priceUpdate() (src/CollateralManager.sol#242-252) ignores return value by IOracle(collateralParams[collateralSupport[i]].oracle).fetchPrice() (src/CollateralManager.sol#247)
CollateralManager.validAdjustment(address,address,uint256) (src/CollateralManager.sol#606-633) ignores return value by IOracle(collateralParams[_collateral].oracle).fetchPrice() (src/CollateralManager.sol#616)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

contracts/PriceFeed.sol

```
ERDMath.calculateCompoundedInterest(uint256,uint40,uint256) (src/Dependencies/ERDMath.sol#250-280) performs a multiplication on the result of a division:
	-ratePerSecond = rate / SECONDS_PER_YEAR (src/Dependencies/ERDMath.sol#266)
	-WadRayMath.ray().add(ratePerSecond.mul(exp)).add(secondTerm).add(thirdTerm) (src/Dependencies/ERDMath.sol#276-279)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

ERDMath.calculateCompoundedInterest(uint256,uint40,uint256) (src/Dependencies/ERDMath.sol#250-280) uses a dangerous strict equality:
	- exp == 0 (src/Dependencies/ERDMath.sol#258)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

PriceFeed._getPrevChainlinkResponse(uint80,uint8).timestamp (src/PriceFeed.sol#678) is a local variable never initialized
ERDMath._getArrayCopy(uint256[]).i (src/Dependencies/ERDMath.sol#210) is a local variable never initialized
PriceFeed._getCurrentTellorResponse()._timestampRetrieved (src/PriceFeed.sol#616) is a local variable never initialized
ERDMath._arrayIsNonzero(uint256[]).i (src/Dependencies/ERDMath.sol#196) is a local variable never initialized
PriceFeed._getCurrentChainlinkResponse().timestamp (src/PriceFeed.sol#650) is a local variable never initialized
PriceFeed._getCurrentChainlinkResponse().decimals (src/PriceFeed.sol#637) is a local variable never initialized
PriceFeed._getPrevChainlinkResponse(uint80,uint8).roundId (src/PriceFeed.sol#675) is a local variable never initialized
PriceFeed._getPrevChainlinkResponse(uint80,uint8).answer (src/PriceFeed.sol#676) is a local variable never initialized
PriceFeed._getCurrentTellorResponse().ifRetrieve (src/PriceFeed.sol#614) is a local variable never initialized
PriceFeed._getCurrentChainlinkResponse().roundId (src/PriceFeed.sol#647) is a local variable never initialized
PriceFeed._scaleChainlinkPriceByDigits(uint256,uint256).price (src/PriceFeed.sol#556) is a local variable never initialized
PriceFeed._getCurrentTellorResponse().value (src/PriceFeed.sol#615) is a local variable never initialized
PriceFeed._getCurrentChainlinkResponse().answer (src/PriceFeed.sol#648) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

PriceFeed._getCurrentTellorResponse() (src/PriceFeed.sol#608-629) ignores return value by tellorCaller.getTellorCurrentValue(ETHUSD_TELLOR_REQ_ID) (src/PriceFeed.sol#613-628)
PriceFeed._getCurrentChainlinkResponse() (src/PriceFeed.sol#631-663) ignores return value by priceAggregator.decimals() (src/PriceFeed.sol#637-643)
PriceFeed._getCurrentChainlinkResponse() (src/PriceFeed.sol#631-663) ignores return value by priceAggregator.latestRoundData() (src/PriceFeed.sol#646-662)
PriceFeed._getPrevChainlinkResponse(uint80,uint8) (src/PriceFeed.sol#665-692) ignores return value by priceAggregator.getRoundData(_currentRoundId - 1) (src/PriceFeed.sol#674-691)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

Variable 'PriceFeed._getCurrentTellorResponse().ifRetrieve (src/PriceFeed.sol#614)' in PriceFeed._getCurrentTellorResponse() (src/PriceFeed.sol#608-629) potentially used before declaration: tellorResponse.ifRetriev
Variable 'PriceFeed._getCurrentTellorResponse().value (src/PriceFeed.sol#615)' in PriceFeed._getCurrentTellorResponse() (src/PriceFeed.sol#608-629) potentially used before declaration: tellorResponse.value = value
Variable 'PriceFeed._getCurrentTellorResponse()._timestampRetrieved (src/PriceFeed.sol#616)' in PriceFeed._getCurrentTellorResponse() (src/PriceFeed.sol#608-629) potentially used before declaration: tellorResponse.
Variable 'PriceFeed._getCurrentChainlinkResponse().decimals (src/PriceFeed.sol#637)' in PriceFeed._getCurrentChainlinkResponse() (src/PriceFeed.sol#631-663) potentially used before declaration: chainlinkResponse.de
Variable 'PriceFeed._getCurrentChainlinkResponse().roundId (src/PriceFeed.sol#647)' in PriceFeed._getCurrentChainlinkResponse() (src/PriceFeed.sol#631-663) potentially used before declaration: chainlinkResponse.rou
Variable 'PriceFeed._getCurrentChainlinkResponse().answer (src/PriceFeed.sol#648)' in PriceFeed._getCurrentChainlinkResponse() (src/PriceFeed.sol#631-663) potentially used before declaration: chainlinkResponse.answ
Variable 'PriceFeed._getCurrentChainlinkResponse().timestamp (src/PriceFeed.sol#650)' in PriceFeed._getCurrentChainlinkResponse() (src/PriceFeed.sol#631-663) potentially used before declaration: chainlinkResponse.t
Variable 'PriceFeed._getPrevChainlinkResponse(uint80,uint8).roundId (src/PriceFeed.sol#675)' in PriceFeed._getPrevChainlinkResponse(uint80,uint8) (src/PriceFeed.sol#665-692) potentially used before declaration: pre
682)
Variable 'PriceFeed._getPrevChainlinkResponse(uint80,uint8).answer (src/PriceFeed.sol#676)' in PriceFeed._getPrevChainlinkResponse(uint80,uint8) (src/PriceFeed.sol#665-692) potentially used before declaration: prev
)
Variable 'PriceFeed._getPrevChainlinkResponse(uint80,uint8).timestamp (src/PriceFeed.sol#678)' in PriceFeed._getPrevChainlinkResponse(uint80,uint8) (src/PriceFeed.sol#665-692) potentially used before declaration: p
d.sol#684)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables
```

contracts/StabilityPool.sol

```
StabilityPool._sendCollateralGainToDepositor(address[],uint256[]) (src/StabilityPool.sol#1048-1092) sends eth to arbitrary user
	Dangerous calls:
	- (success) = msg.sender.call{value: ETHAmount}() (src/StabilityPool.sol#1084)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations

ERDBase.defaultPool (src/Dependencies/ERDBase.sol#31) is never initialized. It is used in:
	- ERDBase.getEntireSystemColl() (src/Dependencies/ERDBase.sol#79-96)
	- ERDBase.getEntireSystemDebt() (src/Dependencies/ERDBase.sol#106-115)
ERDBase.troveDebt (src/Dependencies/ERDBase.sol#33) is never initialized. It is used in:
	- ERDBase.getEntireSystemDebt() (src/Dependencies/ERDBase.sol#106-115)
ERDBase.gasPoolAddress (src/Dependencies/ERDBase.sol#37) is never initialized. It is used in:
	- ERDBase.getEntireSystemDebt() (src/Dependencies/ERDBase.sol#106-115)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables

ERDMath.calculateCompoundedInterest(uint256,uint40,uint256) (src/Dependencies/ERDMath.sol#250-280) performs a multiplication on the result of a division:
	-ratePerSecond = rate / SECONDS_PER_YEAR (src/Dependencies/ERDMath.sol#266)
	-WadRayMath.ray().add(ratePerSecond.mul(exp)).add(secondTerm).add(thirdTerm) (src/Dependencies/ERDMath.sol#276-279)
StabilityPool._computeGainPerUnitStaked(uint256,uint256) (src/StabilityPool.sol#520-545) performs a multiplication on the result of a division:
	-gainPerUnitStaked = gainNumerator.div(_totalEUSDDeposits) (src/StabilityPool.sol#539)
	-lastGainError = gainNumerator.sub(gainPerUnitStaked.mul(_totalEUSDDeposits)) (src/StabilityPool.sol#540-542)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

ERDMath.calculateCompoundedInterest(uint256,uint40,uint256) (src/Dependencies/ERDMath.sol#250-280) uses a dangerous strict equality:
	- exp == 0 (src/Dependencies/ERDMath.sol#258)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

StabilityPool._updateFrontEndStakeAndSnapshots(address,uint256) (src/StabilityPool.sol#1186-1214) deletes IStabilityPool.Snapshots (src/Interfaces/IStabilityPool.sol#36-42) which contains a mapping:
	-delete frontEndSnapshots[_frontEnd] (src/StabilityPool.sol#1193)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#deletion-on-mapping-containing-a-structure
```

contracts/TroveManagerLiquidations.sol

```
ERDMath.calculateCompoundedInterest(uint256,uint40,uint256) (src/Dependencies/ERDMath.sol#250-280) performs a multiplication on the result of a division:
	-ratePerSecond = rate / SECONDS_PER_YEAR (src/Dependencies/ERDMath.sol#266)
	-WadRayMath.ray().add(ratePerSecond.mul(exp)).add(secondTerm).add(thirdTerm) (src/Dependencies/ERDMath.sol#276-279)
TroveManagerLiquidations._getOffsetAndRedistributionVals(uint256,uint256,address[],uint256[],uint256) (src/TroveManagerLiquidations.sol#388-465) performs a multiplication on the result of a division:
	-offset = _colls[i].mul(totalValueToSendToSP.mul(_100pct).div(values[i])).div(_100pct) (src/TroveManagerLiquidations.sol#448-450)
TroveManagerLiquidations._getCappedCollPortion(address[],uint256[],uint256,uint256) (src/TroveManagerLiquidations.sol#516-555) performs a multiplication on the result of a division:
	-offset = _colls[i].mul(portion.mul(_100pct).div(values[i])).div(_100pct) (src/TroveManagerLiquidations.sol#544-546)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

ERDMath.calculateCompoundedInterest(uint256,uint40,uint256) (src/Dependencies/ERDMath.sol#250-280) uses a dangerous strict equality:
	- exp == 0 (src/Dependencies/ERDMath.sol#258)
TroveManagerLiquidations._getCappedCollPortion(address[],uint256[],uint256,uint256) (src/TroveManagerLiquidations.sol#516-555) uses a dangerous strict equality:
	- values[i] == 0 || portion == 0 (src/TroveManagerLiquidations.sol#532)
TroveManagerLiquidations._getCappedCollPortion(address[],uint256[],uint256,uint256) (src/TroveManagerLiquidations.sol#516-555) uses a dangerous strict equality:
	- i == 0 (src/TroveManagerLiquidations.sol#534)
TroveManagerLiquidations._getCappedCollPortion(address[],uint256[],uint256,uint256) (src/TroveManagerLiquidations.sol#516-555) uses a dangerous strict equality:
	- i == 0 (src/TroveManagerLiquidations.sol#550)
TroveManagerLiquidations._getOffsetAndRedistributionVals(uint256,uint256,address[],uint256[],uint256) (src/TroveManagerLiquidations.sol#388-465) uses a dangerous strict equality:
	- i == 0 (src/TroveManagerLiquidations.sol#430)
TroveManagerLiquidations._getOffsetAndRedistributionVals(uint256,uint256,address[],uint256[],uint256) (src/TroveManagerLiquidations.sol#388-465) uses a dangerous strict equality:
	- totalValueToSendToSP == 0 (src/TroveManagerLiquidations.sol#435)
TroveManagerLiquidations._getOffsetAndRedistributionVals(uint256,uint256,address[],uint256[],uint256) (src/TroveManagerLiquidations.sol#388-465) uses a dangerous strict equality:
	- i == 0 (src/TroveManagerLiquidations.sol#438)
TroveManagerLiquidations._getOffsetAndRedistributionVals(uint256,uint256,address[],uint256[],uint256) (src/TroveManagerLiquidations.sol#388-465) uses a dangerous strict equality:
	- i == 0 (src/TroveManagerLiquidations.sol#455)
TroveManagerLiquidations._getTotalFromBatchLiquidate_RecoveryMode(ICollateralManager,IActivePool,IDefaultPool,uint256,uint256,address[]) (src/TroveManagerLiquidations.sol#968-1074) uses a dangerous strict equality:
	- vars.ICR >= mcr && vars.remainingEUSDInStabPool == 0 (src/TroveManagerLiquidations.sol#1001)
TroveManagerLiquidations._getTotalsFromLiquidateTrovesSequence_RecoveryMode(DataTypes.ContractsCache,uint256,uint256,uint256) (src/TroveManagerLiquidations.sol#690-796) uses a dangerous strict equality:
	- vars.ICR >= mcr && vars.remainingEUSDInStabPool == 0 (src/TroveManagerLiquidations.sol#722)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

TroveManagerLiquidations._getOffsetAndRedistributionVals(uint256,uint256,address[],uint256[],uint256) (src/TroveManagerLiquidations.sol#388-465) contains a tautology or contradiction:
	- i >= 0 (src/TroveManagerLiquidations.sol#426)
TroveManagerLiquidations._getCappedCollPortion(address[],uint256[],uint256,uint256) (src/TroveManagerLiquidations.sol#516-555) contains a tautology or contradiction:
	- i >= 0 (src/TroveManagerLiquidations.sol#531)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#tautology-or-contradiction

TroveManagerLiquidations.batchLiquidateTroves(address[],address).vars (src/TroveManagerLiquidations.sol#855) is a local variable never initialized
TroveManagerLiquidations._liquidateNormalMode(IActivePool,IDefaultPool,address,uint256,uint256).vars (src/TroveManagerLiquidations.sol#157) is a local variable never initialized
TroveManagerLiquidations._getTotalsFromBatchLiquidate_NormalMode(IActivePool,IDefaultPool,uint256,uint256,address[]).vars (src/TroveManagerLiquidations.sol#1083) is a local variable never initialized
TroveManagerLiquidations._getTotalsFromLiquidateTrovesSequence_RecoveryMode(DataTypes.ContractsCache,uint256,uint256,uint256).vars (src/TroveManagerLiquidations.sol#697) is a local variable never initialized
ERDMath._getArrayCopy(uint256[]).i (src/Dependencies/ERDMath.sol#210) is a local variable never initialized
TroveManagerLiquidations.liquidateTroves(uint256,address).vars (src/TroveManagerLiquidations.sol#579) is a local variable never initialized
TroveManagerLiquidations._liquidateRecoveryMode(IActivePool,IDefaultPool,address,uint256,uint256,uint256,uint256).vars (src/TroveManagerLiquidations.sol#227) is a local variable never initialized
ERDMath._arrayIsNonzero(uint256[]).i (src/Dependencies/ERDMath.sol#196) is a local variable never initialized
TroveManagerLiquidations._liquidateRecoveryMode(IActivePool,IDefaultPool,address,uint256,uint256,uint256,uint256).zeroVals (src/TroveManagerLiquidations.sol#378) is a local variable never initialized
TroveManagerLiquidations._getTotalFromBatchLiquidate_RecoveryMode(ICollateralManager,IActivePool,IDefaultPool,uint256,uint256,address[]).vars (src/TroveManagerLiquidations.sol#976) is a local variable never initial
TroveManagerLiquidations._getTotalsFromLiquidateTrovesSequence_NormalMode(IActivePool,IDefaultPool,uint256,uint256,uint256).vars (src/TroveManagerLiquidations.sol#805) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
```

contracts/TroveManagerRedemptions.sol

```
ERDMath.calculateCompoundedInterest(uint256,uint40,uint256) (src/Dependencies/ERDMath.sol#250-280) performs a multiplication on the result of a division:
	-ratePerSecond = rate / SECONDS_PER_YEAR (src/Dependencies/ERDMath.sol#266)
	-WadRayMath.ray().add(ratePerSecond.mul(exp)).add(secondTerm).add(thirdTerm) (src/Dependencies/ERDMath.sol#276-279)
TroveManagerRedemptions._calculateCollLot(uint256,address[],uint256[],uint256) (src/TroveManagerRedemptions.sol#232-284) performs a multiplication on the result of a division:
	-portion = EUSDLot.mul(_100pct).div(value) (src/TroveManagerRedemptions.sol#271)
	-offset = coll.mul(portion).div(_100pct) (src/TroveManagerRedemptions.sol#272)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

ERDMath.calculateCompoundedInterest(uint256,uint40,uint256) (src/Dependencies/ERDMath.sol#250-280) uses a dangerous strict equality:
	- exp == 0 (src/Dependencies/ERDMath.sol#258)
TroveManagerRedemptions._calculateCollLot(uint256,address[],uint256[],uint256) (src/TroveManagerRedemptions.sol#232-284) uses a dangerous strict equality:
	- EUSDLot == 0 (src/TroveManagerRedemptions.sol#258)
TroveManagerRedemptions._redeemCollateralFromTrove(DataTypes.ContractsCache,address,uint256,uint256,address,address,uint256) (src/TroveManagerRedemptions.sol#121-230) uses a dangerous strict equality:
	- newDebt == gas (src/TroveManagerRedemptions.sol#175)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

TroveManagerRedemptions._calculateCollLot(uint256,address[],uint256[],uint256) (src/TroveManagerRedemptions.sol#232-284) contains a tautology or contradiction:
	- i_scope_0 >= 0 (src/TroveManagerRedemptions.sol#254)
TroveManagerRedemptions._calculateCollLot(uint256,address[],uint256[],uint256) (src/TroveManagerRedemptions.sol#232-284) contains a tautology or contradiction:
	- i >= 0 (src/TroveManagerRedemptions.sol#246)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#tautology-or-contradiction

ERDMath._getArrayCopy(uint256[]).i (src/Dependencies/ERDMath.sol#210) is a local variable never initialized
ERDMath._arrayIsNonzero(uint256[]).i (src/Dependencies/ERDMath.sol#196) is a local variable never initialized
TroveManagerRedemptions.redeemCollateral(uint256,address,address,address,uint256,uint256,uint256,address).totals (src/TroveManagerRedemptions.sol#373) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

TroveManagerRedemptions._redeemCollateralFromTrove(DataTypes.ContractsCache,address,uint256,uint256,address,address,uint256) (src/TroveManagerRedemptions.sol#121-230) ignores return value by contractsCache.troveMan
erRedemptions.sol#214)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

- As a result of the tests carried out with the Slither tool, some results were obtained and reviewed by Halborn. Based on the results reviewed, some vulnerabilities were determined to be false positives. The actual vulnerabilities found by Slither are already included in the report findings.

6.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruits on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on all the contracts and sent the compiled results to the analyzers to locate any vulnerabilities.

MythX results:

Report for src/ActivePool.sol
https://dashboard.mythx.io/#/console/analyses/45ddfe0e-4b94-441b-b69e-2b7d07c34cfb

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.
22	(SWC-123) Requirement Violation	Low	Requirement violation.
134	(SWC-123) Requirement Violation	Low	Requirement violation.

Report for src/BorrowerOperations.sol
https://dashboard.mythx.io/#/console/analyses/c6b0323a-11fc-461f-9c5d-ef85e1a25d0a

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.
31	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.
33	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.

Report for src/CollateralManager.sol
https://dashboard.mythx.io/#/console/analyses/e264e5a0-6d27-498f-8b74-63070aa74b4a

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.

Report for src/PriceFeed.sol
https://dashboard.mythx.io/#/console/analyses/52c85e3c-cd7d-4d8b-bf4b-c12de8b92e28

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.
31	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.
32	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.

Report for src/StabilityPool.sol
https://dashboard.mythx.io/#/console/analyses/2ec42f1b-0f4b-48f1-9bc1-172440b06d60

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.

Report for src/TroveManagerLiquidations.sol
https://dashboard.mythx.io/#/console/analyses/026a7e21-222d-4434-983a-15f24244c7b7

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.
27	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.

Report for src/TroveManagerRedemptions.sol
https://dashboard.mythx.io/#/console/analyses/8fecc0dc-3434-4e26-8f5e-3edc344fb26b

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.
27	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.

- No major issues found by Mythx.

THANK YOU FOR CHOOSING

HALBORN