Seneca - SenecaDefi

Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **November 30th, 2023 - December 8th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	12/05/2023
0.2	Document Updates	12/07/2023
0.3	Draft Review	12/08/2023
0.4	Draft Review	12/08/2023
1.0	Remediation Plan	12/14/2023
1.1	Remediation Plan Review	12/14/2023
1.2	Remediation Plan Review	12/15/2023

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

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Seneca engaged Halborn to conduct a security assessment on their smart contracts beginning on November 30th, 2023 and ending on December 8th, 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 around one week for the engagement and assigned a full-time security engineer to verify 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 Seneca team.

1.3 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 code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during 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.
- 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)	1
	Partial (R:P)	0.5
	Full (R:F)	0.25
Scope (s)	Changed (S:C)	1.25
	Unchanged (S:U)	1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

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

2.4 SCOPE

IN-SCOPE CODE & COMMITS:

- Repository: SenecaDefi/Seneca
  - Commit ID: 2efe67f6fa9cff42b60ac959b2bc30b063705485
  - Smart contracts **in scope**:
    - contracts/Chamber.sol
    - contracts/senUSD_OFT.sol

OUT-OF-SCOPE:

- Third-party libraries and dependencies.
- Economic attacks.

REMEDIATION COMMIT ID:

- 5bf575e3d619c9f57cc2dfaac2f5dc859e28b27d

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	2	7

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) USE OF APPROVE METHOD TO DIRECTLY APPROVE MAX AMOUNT	Medium (4.5)	SOLVED - 12/14/2023
(HAL-02) USE OF UNSAFE APPROVE METHOD IN AGGREGATEDTLSD CONTRACT	Low (3.1)	SOLVED - 12/14/2023
(HAL-03) MISSING A ZERO CHECK FOR CHANGE BORROW LIMIT	Low (2.5)	SOLVED - 12/14/2023
(HAL-04) CONTRACT PAUSE FEATURE MISSING	Informational (1.0)	SOLVED - 12/14/2023
(HAL-05) 2-STEP TRANSFER OWNERSHIP MISSING	Informational (1.0)	SOLVED - 12/14/2023
(HAL-06) USE OF CUSTOM ERRORS MISSING	Informational (1.0)	SOLVED - 12/14/2023
(HAL-07) INCONSISTENCY BETWEEN FILE NAME AND CONTRACT NAME	Informational (1.0)	SOLVED - 12/14/2023
(HAL-08) FLOATING PRAGMA	Informational (0.0)	SOLVED - 12/14/2023
(HAL-09) CACHING LENGTH IN FOR LOOPS	Informational (0.0)	SOLVED - 12/14/2023
(HAL-10) LONG LITERAL UINT256 USED IN CONSTANTS LIBRARY	Informational (0.0)	SOLVED - 12/14/2023

FINDINGS & TECH DETAILS

4.1 (HAL-01) USE OF APPROVE METHOD TO DIRECTLY APPROVE MAX AMOUNT - MEDIUM (4.5)

Description:

The ChamberContract contract approves the max amount of **senUSD** tokens instead of just approving the specific amount when needed to be sent to bentoBox. This behavior could lead to security issues and potential losses in the case of a security breach.

Code Location:

Listing 1: Chamber.sol (Line 121)

```
115 function init(bytes calldata data) public virtual payable override
    ↳ {
116     require(address(collateral) == address(0), "Chamber: already
    ↳ initialized");
117     (collateral, oracle, oracleData, accruedInterest.
    ↳ INTEREST_PER_SECOND, LIQUIDATION_MULTIPLIER, COLLATERIZATION_RATE,
    ↳  BORROW_OPENING_FEE) = abi.decode(data, (IERC20, IOracle, bytes,
    ↳ uint64, uint256, uint256, uint256));
118     borrowLimit = BorrowCap(type(uint128).max, type(uint128).max);
119     require(address(collateral) != address(0), "Chamber: bad pair"
    ↳ );
120
121     senUSD.approve(address(bentoBox), type(uint256).max);
122
123     blacklisted[address(bentoBox)] = true;
124     blacklisted[address(this)] = true;
125     blacklisted[Ownable(address(bentoBox)).owner()] = true;
126
127     (, exchangeRate) = oracle.get(oracleData);
128
129     accumulate();
130 }
```

BVSS:

AO:A/AC:M/AX:M/C:N/I:N/A:N/D:C/Y:N/R:N/S:U (4.5)

Recommendation:

It is recommended to approve only the minimum amount necessary, or set approval to zero after the operations for a contract to perform its intended functions. This minimizes risks, reduces potential losses in case of a security breach, and follows the principle of least privilege.

Remediation Plan:

SOLVED: The Seneca team solved the issue in the following commit id.

Commit ID: 5bf575e3d619c9f57cc2dfaac2f5dc859e28b27d

4.2 (HAL-02) USE OF UNSAFE APPROVE METHOD IN AGGREGATEDTLSD CONTRACT - LOW (3.1)

Description:

The init function within the contract uses the approve method from the ERC20 standard to set the allowance for bentoBox. However, this implementation does not use the safeApprove method available in OpenZeppelin 's SafeERC20 library. The use of plain approve might lead to potential issues due to the allowance manipulation vulnerability, known as approval race condition.

Code Location:

Listing 2: Chamber.sol (Line 121)

```
115 function init(bytes calldata data) public virtual payable override {
116     require(address(collateral) == address(0), "Chamber: already initialized");
117     (collateral, oracle, oracleData, accruedInterest.INTEREST_PER_SECOND, LIQUIDATION_MULTIPLIER, COLLATERIZATION_RATE, BORROW_OPENING_FEE) = abi.decode(data, (IERC20, IOracle, bytes, uint64, uint256, uint256, uint256));
118     borrowLimit = BorrowCap(type(uint128).max, type(uint128).max);
119     require(address(collateral) != address(0), "Chamber: bad pair");
120
121     senUSD.approve(address(bentoBox), type(uint256).max);
122
123     blacklisted[address(bentoBox)] = true;
124     blacklisted[address(this)] = true;
125     ...
```

BVSS:

AO:A/AC:L/AX:L/C:L/I:L/A:N/D:N/Y:N/R:N/S:U (3.1)

Recommendation:

To mitigate the potential risks associated with the approve method, you should consider using the safeApprove method from OpenZeppelin's SafeERC20 library. This will ensure that the contract’s token operations are secure and resistant to known vulnerabilities.

Remediation Plan:

SOLVED: The Seneca team solved the issue in the following commit id.

Commit ID: 5bf575e3d619c9f57cc2dfaac2f5dc859e28b27d

4.3 (HAL-03) MISSING A ZERO CHECK FOR CHANGE BORROW LIMIT - LOW (2.5)

Description:

The function changeBorrowLimit allows the owner of the master contract to set the maximum borrow amount that totalBorrow can have. Thus, whenever a user executes a borrow transaction, cannot be borrowed if the limit is set to zero. However, when setting this storage variable, a check that checks it is not being set to zero is missing. Ensuring no disruption of the service.

Code Location:

Listing 3: Chamber.sol (Line 618)

```
617 function changeBorrowLimit(uint128 newBorrowLimit, uint128 perAddressPart) public onlyMasterContractOwner {
618     borrowLimit = BorrowCap(newBorrowLimit, perAddressPart);
619     emit LogChangeBorrowLimit(newBorrowLimit, perAddressPart);
620 }
```

BVSS:

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

Recommendation:

The use of a requirement ensuring the maximum borrow amount is greater than 0 is recommended when updating the storage calling changeBorrowLimit function.

Remediation Plan:

SOLVED: The Seneca team solved the issue in the following commit id.

Commit ID: 5bf575e3d619c9f57cc2dfaac2f5dc859e28b27d

4.4 (HAL-04) CONTRACT PAUSE FEATURE MISSING - INFORMATIONAL (1.0)

Description:

It was identified that no high-privileged user can pause any of the scoped contracts. In the event of a security incident, the owner would not be able to stop any plausible malicious actions. Pausing the contract can also lead to more considered decisions.

BVSS:

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

Recommendation:

Consider adding the pausable functionality to the contract.

Remediation Plan:

SOLVED: The Seneca team solved the issue in the following commit id.

Commit ID: 5bf575e3d619c9f57cc2dfaac2f5dc859e28b27d

4.5 (HAL-05) 2-STEP TRANSFER OWNERSHIP MISSING - INFORMATIONAL (1.0)

Description:

The code does not implement a two-step ownership transfer pattern. This practice is recommended when admin users have heavy responsibilities, such as the ability to mint tokens, freeze or unfreeze user accounts and set system configurations. It may happen that when transferring ownership of a contract, an error is made in the address. If the request were submitted, the contract would be lost forever. With this pattern, contract owners can submit a transfer request; however, this is not final until accepted by the new owner. If they realize they have made a mistake, they can stop it at any time before accepting it by calling cancelRequest.

Code Location:

Listing 4: Chamber.sol

```
contract ChamberContract is Ownable, IMasterContract {
```

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 recommended to implement a two-step process where the owner nominates an account, and the nominated account needs to call an acceptOwnership() function for the transfer of the ownership to succeed fully. This ensures the nominated EOA account is valid and active.

Remediation Plan:

SOLVED: The Seneca team solved the issue in the following commit id.

Commit ID: 5bf575e3d619c9f57cc2dfaac2f5dc859e28b27d

4.6 (HAL-06) USE OF CUSTOM ERRORS MISSING - INFORMATIONAL (1.0)

Description:

Failed operations in this contract are reverted with an accompanying message selected from a set of hard-coded strings.

In EVM, emitting a hard-coded string in an error message costs ~50 more gas than emitting a custom error. Additionally, hard-coded strings increase the gas required to deploy the contract.

Code Location:

Listing 5: Chamber.sol

```
require(msg.sender == masterContract.owner(), "Caller is not the owner");
```

Listing 6: Chamber.sol

```
require(address(collateral) == address(0), "Chamber: already initialized");
```

Listing 7: Chamber.sol

```
require(address(collateral) != address(0), "Chamber: bad pair");
```

Listing 8: Chamber.sol

```
require(_isSolvent(msg.sender, _exchangeRate), "Chamber: user insolvent");
```

Listing 9: Chamber.sol

```
require(share <= bentoBox.balanceOf(token, address(this)).sub(total), "Chamber: Skim too much");
```

Listing 10: Chamber.sol

```
require(totalBorrow.elastic <= cap.total, "Borrow Limit reached");
```

Listing 11: Chamber.sol

```
require(newBorrowPart <= cap.borrowPartPerAddress, "Borrow Limit reached");
```

Listing 12: Chamber.sol

```
require(!blacklisted[callee], "Chamber: can't call");
```

Listing 13: Chamber.sol

```
require(success, "Chamber: call failed");
```

Listing 14: Chamber.sol

```
require((!must_update || updated) && rate > minRate && (maxRate == 0 || rate < maxRate), "Chamber: rate not ok");
```

Listing 15: Chamber.sol

```
require(_isSolvent(msg.sender, _exchangeRate), "Chamber: user insolvent");
```

Listing 16: Chamber.sol

```
require(newInterestRate < oldInterestRate + oldInterestRate * 3 / 4 || newInterestRate <= ONE_PERCENT_RATE(), "Interest rate increase > 75%");


```

```
require(lastInterestUpdate + 3 days < block.timestamp, "Update only every 3 days");
```

Listing 17: Chamber.sol

```
require(callee != address(0), 'invalid callee');
require(callee != address(bentoBox) && callee != address(this), "invalid callee");
```

Listing 18: senUSD_OFT.sol

```
require(_to != address(0), "SENUSD: no mint to zero address");
```

BVSS:

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

Recommendation:

Custom errors are available from Solidity version 0.8.4 up. Consider replacing all revert strings with custom errors. Usage of custom errors should look like this:

Listing 19

```
error CustomError();

// ...

if (condition)
    revert CustomError();
```

Remediation Plan:

SOLVED: The Seneca team solved the issue in the following commit id.

Commit ID: 5bf575e3d619c9f57cc2dfaac2f5dc859e28b27d

4.7 (HAL-07) INCONSISTENCY BETWEEN FILE NAME AND CONTRACT NAME - INFORMATIONAL (1.0)

Description:

The file name and the contract name within the Solidity file are inconsistent. This can lead to confusion and potential errors when trying to interact with or deploy the contract. In Solidity, it is a best practice to keep the contract name and the file name the same for clarity and ease of management.

Code Location:

Listing 20: senUSD_OFT.sol

```
contract SenecaUSD is OFTWithFee {
```

Listing 21: Chamber.sol

```
contract ChamberContract is Ownable, IMasterContract {
```

BVSS:

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

Recommendation:

Rename either the file or the contract so that they match. If the contract name is SenecaUSD, then the file name should ideally be SenecaUSD.sol.

Remediation Plan:

SOLVED: The Seneca team solved the issue in the following commit id.

Commit ID: 5bf575e3d619c9f57cc2dfaac2f5dc859e28b27d

4.8 (HAL-08) FLOATING PRAGMA - INFORMATIONAL (0.0)

Description:

Contracts should be deployed with the same compiler version and flags used during development and testing. Locking the pragma helps to ensure that contracts do not accidentally get deployed using another pragma. For example, an outdated pragma version might introduce bugs that affect the contract system negatively.

Code Location:

The ChamberContract contract is using the `pragma solidity >=0.8.0;` floating pragma.

BVSS:

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

Recommendation:

Consider locking the pragma version in the smart contracts. It is not recommended to use a floating pragma in production.

For example: `pragma solidity 0.8.15;`

Remediation Plan:

SOLVED: The Seneca team solved the issue in the following commit id.

Commit ID: 5bf575e3d619c9f57cc2dfaac2f5dc859e28b27d

4.9 (HAL-09) CACHING LENGTH IN FOR LOOPS - INFORMATIONAL (0.0)

Description:

In a for loop, the length of an array can be put in a temporary variable to save some gas.

In the above cases, the solidity compiler will always read the length of the array during each iteration. That is,

- if it is a storage array, this is an extra sload operation (100 additional extra gas (EIP-2929) for each iteration except for the first),
- if it is a memory array, this is an extra mload operation (3 additional gas for each iteration except for the first),
- if it is a calldata array, this is an extra calldataload operation (3 additional gas for each iteration except for the first)

Code Location:

Listing 22: Chamber.sol (Line 412)

```
405 function performOperations(
406     uint8[] calldata actions,
407     uint256[] calldata values,
408     bytes[] calldata datas
409 ) external payable returns (uint256 value1, uint256 value2) {
410     OperationStatus memory status;
411 
412     for (uint256 i = 0; i < actions.length; i++) {
413         uint8 action = actions[i];
414         if (!status.hasAccrued && action < 10) {
415             accumulate();
416             status.hasAccrued = true;
417         }
418         ...
```

Listing 23: Chamber.sol (Line 516)

```
499 function liquidate(
500     address[] memory users,
501     uint256[] memory maxBorrowParts,
502     address to,
503     ISwapperV2 swapper,
504     bytes memory swapperData
505 ) public virtual {
506     // Oracle can fail but we still need to allow liquidations
507     (, uint256 _exchangeRate) = updatePrice();
508     accumulate();
509
510     uint256 allCollateralShare;
511     uint256 allBorrowAmount;
512     uint256 allBorrowPart;
513     Rebase memory bentoBoxTotals = bentoBox.totals(collateral);
514     _beforeUsersLiquidated(users, maxBorrowParts);
515
516     for (uint256 i = 0; i < users.length; i++) {
517         address user = users[i];
518         if (!_isSolvent(user, _exchangeRate)) {
519         ...
```

BVSS:

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

Recommendation:

In a for loop, store the length of an array in a temporary variable.

Remediation Plan:

SOLVED: The Seneca team solved the issue in the following commit id.

Commit ID: 5bf575e3d619c9f57cc2dfaac2f5dc859e28b27d

4.10 (HAL-10) LONG LITERAL UINT256 USED IN CONSTANTS LIBRARY - INFORMATIONAL (0.0)

Description:

Critical protocol parameters are set within the Constants.sol library. Those parameters 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 24: Constants.sol

```
library Constants {

    uint64 public constant PERCENT_RATE = 317097920;

    // Interest
    uint16 public constant BASIS_POINTS_DENOM = 10000;

    // Core
    uint256 public constant COLLATERIZATION_RATE_PRECISION = 100000;

    // Rates
    uint256 public constant EXCHANGE_RATE_PRECISION = 1000000000000000000;
    uint256 public constant LIQUIDATION_MULTIPLIER_PRECISION = 100000;

    // Fees
    uint256 public constant BORROW_OPENING_FEE_PRECISION = 100000;
    ...
```

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 (125e16, 13e17, etc.) is recommended instead.

Remediation Plan:

SOLVED: The Seneca team solved the issue in the following commit id.

Commit ID: 5bf575e3d619c9f57cc2dfaac2f5dc859e28b27d

AUTOMATED TESTING

5.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their ABIs and binary format, Slither was run against the 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.

Results:

contracts/Chamber.sol

```
INFO:Detectors:
ChamberContract._bentoDeposit(bytes,uint256,uint256,uint256) (src/Chamber.sol#342-352) sends eth to arbitrary user
        Dangerous calls:
        - bentoBox.deposit{value: value}(token,msg.sender,to,uint256(amount),uint256(share)) (src/Chamber.sol#351)
ChamberContract._call(uint256,bytes,uint256,uint256) (src/Chamber.sol#367-389) sends eth to arbitrary user
        Dangerous calls:
        - (success,returnData) = callee.call{value: value}(callData) (src/Chamber.sol#386)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
Reentrancy in ChamberContract.performOperations(uint8[],uint256[],bytes[]) (src/Chamber.sol#405-482):
        External calls:
        - depositCollateral(to,skim,_num(share,value1,value2)) (src/Chamber.sol#420)
                - bentoBox.transfer(token,msg.sender,address(this),share) (src/Chamber.sol#223)
        - _repay(to_scope_0,skim_scope_1,_num(part,value1,value2)) (src/Chamber.sol#423)
                - bentoBox.transfer(senUSD,address(bentoBox),address(this),share) (src/Chamber.sol#313)
                - bentoBox.transfer(senUSD,msg.sender,address(this),share) (src/Chamber.sol#313)
        - _withdrawCollateral(to_scope_3,_num(share_scope_2,value1,value2)) (src/Chamber.sol#426)
                - bentoBox.transfer(collateral,address(this),to,share) (src/Chamber.sol#255)
        - (value1,value2) = _borrow(to_scope_4,_num(amount,value1,value2)) (src/Chamber.sol#430)
                - bentoBox.transfer(senUSD,address(this),to,share) (src/Chamber.sol#290)
        - (updated,rate) = updatePrice() (src/Chamber.sol#434)
                - (updated,rate) = oracle.get(oracleData) (src/Chamber.sol#196)
        - bentoBox.setMasterContractApproval(user,_masterContract,approved,v,r,s) (src/Chamber.sol#439)
        - (value1,value2) = _bentoDeposit(datas[i],values[i],value1,value2) (src/Chamber.sol#441)
                - bentoBox.deposit{value: value}(token,msg.sender,to,uint256(amount),uint256(share)) (src/Chamber.sol#351)
        - (value1,value2) = _bentoWithdraw(datas[i],value1,value2) (src/Chamber.sol#443)
                - bentoBox.withdraw(token,msg.sender,to,_num(amount,value1,value2),_num(share,value1,value2)) (src/Chamber.sol#361)
        - bentoBox.transfer(token,msg.sender,to_scope_5,_num(share_scope_6,value1,value2)) (src/Chamber.sol#446)
        - bentoBox.transferMultiple(token_scope_7,msg.sender,tos,shares) (src/Chamber.sol#449)
        - (returnData,returnValues) = _call(values[i],datas[i],value1,value2) (src/Chamber.sol#451)
                - (success,returnData) = callee.call{value: value}(callData) (src/Chamber.sol#386)
        - _operationLiquidate(datas[i]) (src/Chamber.sol#465)
                - (updated,rate) = oracle.get(oracleData) (src/Chamber.sol#196)
                - bentoBox.transfer(collateral,address(this),to,allCollateralShare) (src/Chamber.sol#561)
                - swapper.swap(address(collateral),address(senUSD),msg.sender,allBorrowShare,allCollateralShare,swapperData) (src/Chamber.sol#563)
                - bentoBox.transfer(senUSD,msg.sender,address(this),allBorrowShare) (src/Chamber.sol#567)
        External calls sending eth:
        - (value1,value2) = _bentoDeposit(datas[i],values[i],value1,value2) (src/Chamber.sol#441)
                - bentoBox.deposit{value: value}(token,msg.sender,to,uint256(amount),uint256(share)) (src/Chamber.sol#351)
        - (returnData,returnValues) = _call(values[i],datas[i],value1,value2) (src/Chamber.sol#451)
                - (success,returnData) = callee.call{value: value}(callData) (src/Chamber.sol#386)
        State variables written after the call(s):
        - accumulate() (src/Chamber.sol#415)
                - accruedInterest = _accruedInterest (src/Chamber.sol#144)
                - accruedInterest = _accruedInterest (src/Chamber.sol#154)
        ChamberContract.accruedInterest (src/Chamber.sol#92) can be used in cross function reentrancies:
        - ChamberContract._borrow(address,uint256) (src/Chamber.sol#272-293)
        - ChamberContract.accruedInterest (src/Chamber.sol#92)
        - ChamberContract.accumulate() (src/Chamber.sol#133-157)
        - ChamberContract.changeInterestRate(uint16) (src/Chamber.sol#601-612)
        - ChamberContract.init(bytes) (src/Chamber.sol#115-130)
        - ChamberContract.liquidate(address[],uint256[],address,ISwapperV2,bytes) (src/Chamber.sol#499-568)
        - ChamberContract.withdrawFees() (src/Chamber.sol#571-580)
        - (value1,value2) = _borrow(to_scope_4,_num(amount,value1,value2)) (src/Chamber.sol#430)
                - accruedInterest.feesEarned = uint128(accruedInterest.feesEarned.add(uint128(feeAmount))) (src/Chamber.sol#280)
        ChamberContract.accruedInterest (src/Chamber.sol#92) can be used in cross function reentrancies:
        - ChamberContract._borrow(address,uint256) (src/Chamber.sol#272-293)
        - ChamberContract.accruedInterest (src/Chamber.sol#92)
        - ChamberContract.accumulate() (src/Chamber.sol#133-157)
        - ChamberContract.changeInterestRate(uint16) (src/Chamber.sol#601-612)
        - ChamberContract.init(bytes) (src/Chamber.sol#115-130)
        - ChamberContract.liquidate(address[],uint256[],address,ISwapperV2,bytes) (src/Chamber.sol#499-568)
        - ChamberContract.withdrawFees() (src/Chamber.sol#571-580)
        - _operationLiquidate(datas[i]) (src/Chamber.sol#465)
                - accruedInterest = _accruedInterest (src/Chamber.sol#144)
                - accruedInterest = _accruedInterest (src/Chamber.sol#154)
                - accruedInterest.feesEarned = uint128(accruedInterest.feesEarned.add(distributionAmount)) (src/Chamber.sol#554)
        ChamberContract.accruedInterest (src/Chamber.sol#92) can be used in cross function reentrancies:
        - ChamberContract._borrow(address,uint256) (src/Chamber.sol#272-293)
        - ChamberContract.accruedInterest (src/Chamber.sol#92)
        - ChamberContract.accumulate() (src/Chamber.sol#133-157)
        - ChamberContract.changeInterestRate(uint16) (src/Chamber.sol#601-612)
```

contracts/senUSD_OFT.sol

```
INFO:Detectors:
BytesLib.concatStorage(bytes,bytes) (src/libraries/BytesLib.sol#86-212) performs a multiplication on the result of a division:
	- sstore(uint256,uint256)(_preBytes,fslot_concatStorage_asm_0 + mload(uint256)(_postBytes + 0x20) / 0x100 ** 32 - mlength_concatStorage_asm_0 * 0x100 ** 32 - newlength_concatStorage_asm_0 + mlength_concatSt
BytesLib.concatStorage(bytes,bytes) (src/libraries/BytesLib.sol#86-212) performs a multiplication on the result of a division:
	- sstore(uint256,uint256)(sc_concatStorage_asm_0,mload(uint256)(mc_concatStorage_asm_0) / mask_concatStorage_asm_0 * mask_concatStorage_asm_0) (src/libraries/BytesLib.sol#175)
BytesLib.concatStorage(bytes,bytes) (src/libraries/BytesLib.sol#86-212) performs a multiplication on the result of a division:
	- sstore(uint256,uint256)(sc_concatStorage_asm_0,mload(uint256)(mc_concatStorage_asm_0) / mask_concatStorage_asm_0 * mask_concatStorage_asm_0) (src/libraries/BytesLib.sol#209)
BytesLib.equalStorage(bytes,bytes) (src/libraries/BytesLib.sol#421-486) performs a multiplication on the result of a division:
	- fslot_equalStorage_asm_0 = fslot_equalStorage_asm_0 / 0x100 * 0x100 (src/libraries/BytesLib.sol#441)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
OFTCoreV2._estimateSendFee(uint16,bytes32,uint256,bool,bytes) (src/token/oft/v2/OFTCoreV2.sol#71-81) ignores return value by lzEndpoint.estimateFees(_dstChainId,address(this),payload,_useZro,_adapterParams) (src/to
OFTCoreV2._estimateSendAndCallFee(uint16,bytes32,uint256,bytes,uint64,bool,bytes) (src/token/oft/v2/OFTCoreV2.sol#83-95) ignores return value by lzEndpoint.estimateFees(_dstChainId,address(this),payload,_useZro,_ad
SenecaUSD.mintToBentoBox(address,uint256,IBentoBoxV1) (src/senUSD_OFT.sol#83-90) ignores return value by bentoBox.deposit(IERC20(address(this)),address(bentoBox),clone,amount,0) (src/senUSD_OFT.sol#89)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
OFTWithFee.constructor(string,string,uint8,address)._name (src/token/oft/v2/fee/OFTWithFee.sol#12) shadows:
	- ERC20._name (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#42) (state variable)
OFTWithFee.constructor(string,string,uint8,address)._symbol (src/token/oft/v2/fee/OFTWithFee.sol#12) shadows:
	- ERC20._symbol (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#43) (state variable)
OFTWithFee.constructor(string,string,uint8,address).decimals (src/token/oft/v2/fee/OFTWithFee.sol#13) shadows:
	- ERC20.decimals() (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#87-89) (function)
	- IERC20Metadata.decimals() (lib/openzeppelin-contracts/contracts/token/ERC20/extensions/IERC20Metadata.sol#27) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
LzApp.setPrecrime(address)._precrime (src/lzApp/LzApp.sol#154) lacks a zero-check on :
		- precrime = _precrime (src/lzApp/LzApp.sol#155)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in OFTCoreV2._send(address,uint16,bytes32,uint256,address,address,bytes) (src/token/oft/v2/OFTCoreV2.sol#114-133):
	External calls:
	- _lzSend(_dstChainId,lzPayload,_refundAddress,_zroPaymentAddress,_adapterParams,msg.value) (src/token/oft/v2/OFTCoreV2.sol#130)
		- lzEndpoint.send{value: _nativeFee}(_dstChainId,trustedRemote,_payload,_refundAddress,_zroPaymentAddress,_adapterParams) (src/lzApp/LzApp.sol#73)
	Event emitted after the call(s):
	- SendToChain(_dstChainId,_from,_toAddress,amount) (src/token/oft/v2/OFTCoreV2.sol#132)
Reentrancy in OFTCoreV2._sendAndCall(address,uint16,bytes32,uint256,bytes,uint64,address,address,bytes) (src/token/oft/v2/OFTCoreV2.sol#152-174):
	External calls:
	- _lzSend(_dstChainId,lzPayload,_refundAddress,_zroPaymentAddress,_adapterParams,msg.value) (src/token/oft/v2/OFTCoreV2.sol#171)
		- lzEndpoint.send{value: _nativeFee}(_dstChainId,trustedRemote,_payload,_refundAddress,_zroPaymentAddress,_adapterParams) (src/lzApp/LzApp.sol#73)
	Event emitted after the call(s):
	- SendToChain(_dstChainId,_from,_toAddress,amount) (src/token/oft/v2/OFTCoreV2.sol#173)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
SenecaUSD.mint(address,uint256) (src/senUSD_OFT.sol#69-81) uses timestamp for comparisons
	Dangerous comparisons:
	- lastMint.time < block.timestamp - MINTING_PERIOD (src/senUSD_OFT.sol#72)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
BytesLib.concat(bytes,bytes) (src/libraries/BytesLib.sol#12-84) uses assembly
	- INLINE ASM (src/libraries/BytesLib.sol#15-81)
BytesLib.concatStorage(bytes,bytes) (src/libraries/BytesLib.sol#86-212) uses assembly
	- INLINE ASM (src/libraries/BytesLib.sol#87-211)
BytesLib.slice(bytes,uint256,uint256) (src/libraries/BytesLib.sol#214-277) uses assembly
	- INLINE ASM (src/libraries/BytesLib.sol#224-274)
BytesLib.toAddress(bytes,uint256) (src/libraries/BytesLib.sol#279-288) uses assembly
	- INLINE ASM (src/libraries/BytesLib.sol#283-285)
BytesLib.toUint8(bytes,uint256) (src/libraries/BytesLib.sol#290-299) uses assembly
	- INLINE ASM (src/libraries/BytesLib.sol#294-296)
BytesLib.toUint16(bytes,uint256) (src/libraries/BytesLib.sol#301-310) uses assembly
	- INLINE ASM (src/libraries/BytesLib.sol#305-307)
BytesLib.toUint32(bytes,uint256) (src/libraries/BytesLib.sol#312-321) uses assembly
	- INLINE ASM (src/libraries/BytesLib.sol#316-318)
BytesLib.toUint64(bytes,uint256) (src/libraries/BytesLib.sol#323-332) uses assembly
	- INLINE ASM (src/libraries/BytesLib.sol#327-329)
BytesLib.toUint96(bytes,uint256) (src/libraries/BytesLib.sol#334-343) uses assembly
	- INLINE ASM (src/libraries/BytesLib.sol#338-340)
BytesLib.toUint128(bytes,uint256) (src/libraries/BytesLib.sol#345-354) uses assembly
	- INLINE ASM (src/libraries/BytesLib.sol#349-351)
BytesLib.toUint256(bytes,uint256) (src/libraries/BytesLib.sol#356-365) uses assembly
	- INLINE ASM (src/libraries/BytesLib.sol#360-362)
BytesLib.toBytes32(bytes,uint256) (src/libraries/BytesLib.sol#367-376) uses assembly
	- INLINE ASM (src/libraries/BytesLib.sol#371-373)
```

All the issues flagged by Slither were manually reviewed by Halborn. Reported issues were either considered as false positives or are already included in the report findings.

THANK YOU FOR CHOOSING

HALBORN