Stater-LendingData

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **April, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	04/12/2021	Gokberk Gulgun
0.2	Document Edits	04/12/2021	Gabi Urrutia
1.0	Final Edits	04/13/2021	Gabi Urrutia
1.1	Final Version	04/14/2021	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
Ataberk Yavuzer	Halborn	Ataberk.Yavuzer@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Stater engaged Halborn to conduct a security assessment on their Smart contract beginning on April 8th, 2021 and ending April 12th, 2021. The security assessment was scoped to the smart contract LendingData.sol . An audit of the security risk and implications regarding the changes introduced by the development team at Stater prior to its production release shortly following the assessments deadline.

Though this security audit's outcome is satisfactory, only the most essential aspects were tested and verified to achieve objectives and deliverables set in the scope due to time and resource constraints. It is essential to note the use of the best practices for secure smart-contract development.

1.2 AUDIT SUMMARY

The team at Halborn was provided four weeks for the engagement and assigned three full time security engineers to audit the security of the smart contract. The security engineers are blockchain and smart-contract security experts with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit to achieve the following:

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

In summary, Halborn identified few security risks, and recommends performing further testing to validate extended safety and correctness in context to the whole set of contracts. External threats, such as economic attacks, oracle attacks, and inter-contract functions and calls should be validated for expected logic and state.

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 the smart contract audit.While manual testing is recommended to uncover flaws in logic, process,and implementation; automated testing techniques help enhance coverage of smart contracts 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 audit:

- Research into architecture and purpose.
- Smart Contract manual code read 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.
- Static Analysis of security for scoped contract, and imported functions.(Slither)
- Testnet deployment (Truffle, Ganache)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident, and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. It’s quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that was used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.

3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

IN-SCOPE:

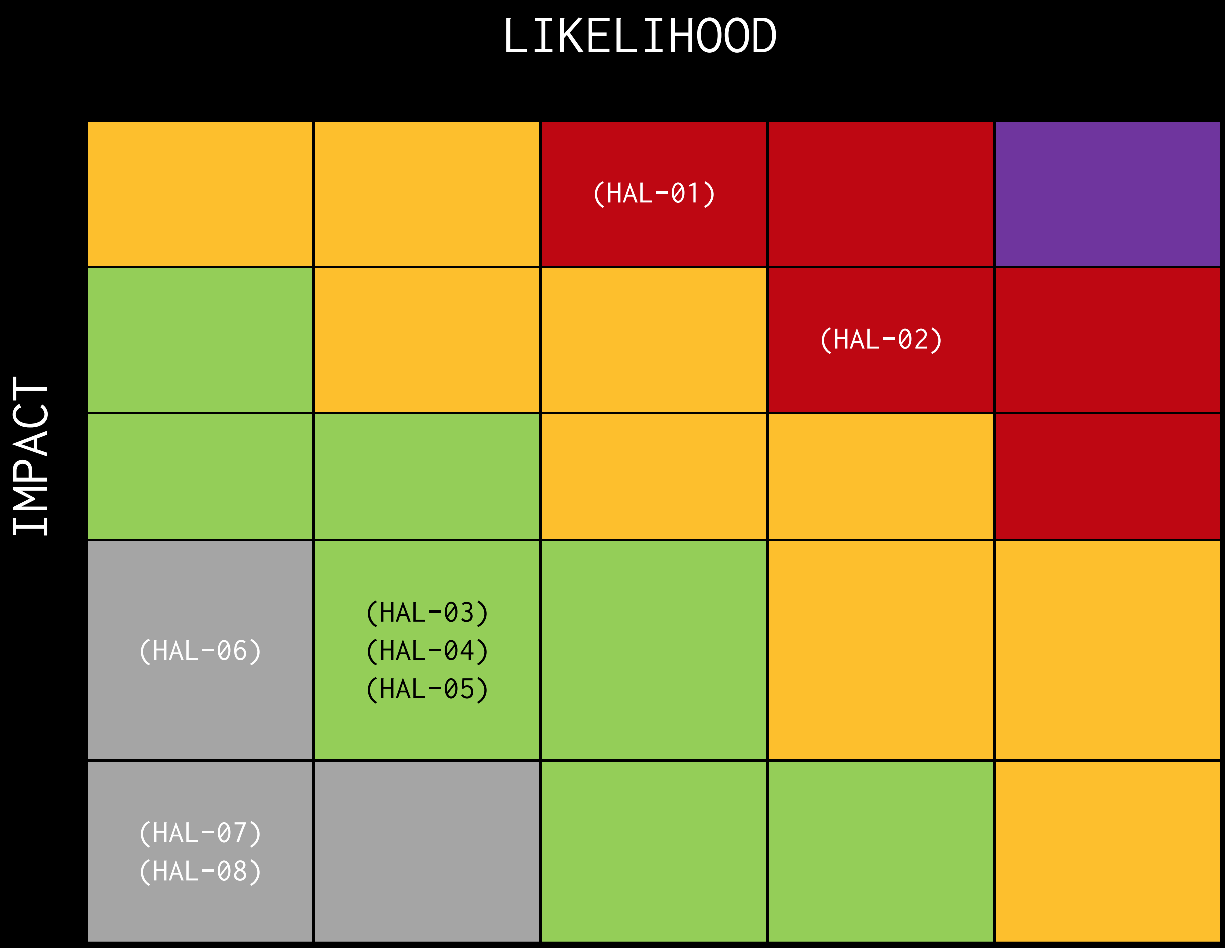
Code related to LendingData.sol smart contract

Specific commit of contract:

e48045700f6ef922c7f29239dc126b7433e1f052

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	2	0	3	3



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
OWNER CAN RENOUNCE OWNERSHIP	High	SOLVED: 04/14/2021
IMPROPER ROLE-BASED ACCESS CONTROL POLICY	High	SOLVED: 04/14/2021
LACK OF ADDRESS CONTROL ON THE FUNCTIONS	Low	FUTURE RELEASE UPDATE
USAGE OF BLOCK-TIMESTAMP	Low	SOLVED: 04/14/2021
IGNORE RETURN VALUES	Low	RISK ACCEPTED: 04/14/2021
POSSIBLE RE-ENTRANCY	Informational	RISK ACCEPTED: 04/14/2021
FOR LOOP OVER DYNAMIC ARRAY	Informational	RISK ACCEPTED: 04/14/2021
PRAGMA VERSION	Informational	RISK ACCEPTED: 04/14/2021
STATIC ANALYSIS		

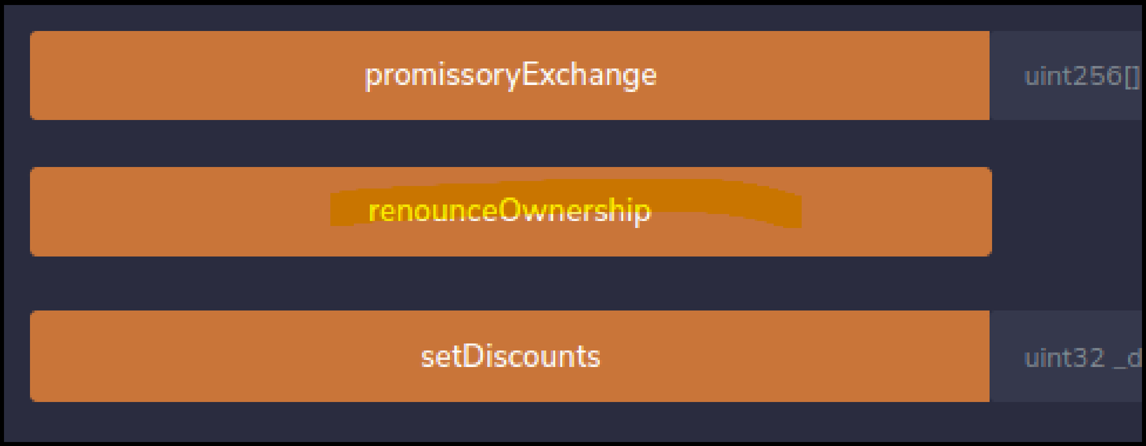
FINDINGS & TECH DETAILS

3.1 (HAL-01) OWNER CAN RENOUNCE OWNERSHIP - HIGH

Description:

The Owner of the contract is usually the account which deploys the contract. As a result, the Owner is able to perform some privileged actions. In LendingData.sol smart contract, the renounceOwnership function is used to renounce being Owner. Otherwise, if the ownership was not transferred before, the contract will never have an Owner, which is dangerous.

Function:



Recommendation:

It's recommended that the Owner is not able to call renounceOwnership without transferring the Ownership to other address before. In addition, if a multi-signature wallet is used, calling renounceOwnership function should be confirmed for two or more users.

Remediation Plan:

Solved: Stater team will use a multi-signature wallet for the deployment to the mainnet.

3.2 (HAL-02) IMPROPER ROLE-BASED ACCESS CONTROL POLICY - HIGH

Description:

Implementing a valid access control policy is an essential step in maintaining the security of a smart-contract. All the features of the smart contract , such as add/remove roles and upgrade contracts are given by Access Control. For instance, Ownership is the most common form of Access Control. In other words, the owner of a contract (the account that deployed it by default) can do some administrative tasks on it. Additional authorization levels are needed to implement the least privilege principle, also known as least-authority, which ensures only authorized processes, users, or programs can access the necessary resources or information. The ownership role is useful in a simple system, but more complex projects require the use of more roles by using Role-based access control.

There could be multiple roles such as manager, admin in contracts which use a proxy contract.In LendingData.sol contract, owner is the only one privileged role. Owner can transfer the contract ownership, call the following functions. In conclusion, owner role can do too many actions in LendingData smart contract. If the private key of the owner account is compromised and multi-signature was not implemented, the attacker can perform many actions such as transferring ownership or change NFT addresses without following the principle of least privilege.

FINDINGS & TECH DETAILS

Functions:

Listing 1

```
function setDiscounts
function setGlobalVariables
function addGeyserAddress
function addNftTokenId
function transferOwnership
function promissoryExchange
```

Risk Level:

Likelihood - 4
Impact - 4

Recommendation:

A white-listing should be applied for the access policies on the smart contracts. Access Control policies should be determined over each role and the access control policies should be prevented from having only one authority.

Remediation Plan:

Solved: Stater team will use a multi-signature wallet for the deployment to the mainnet.

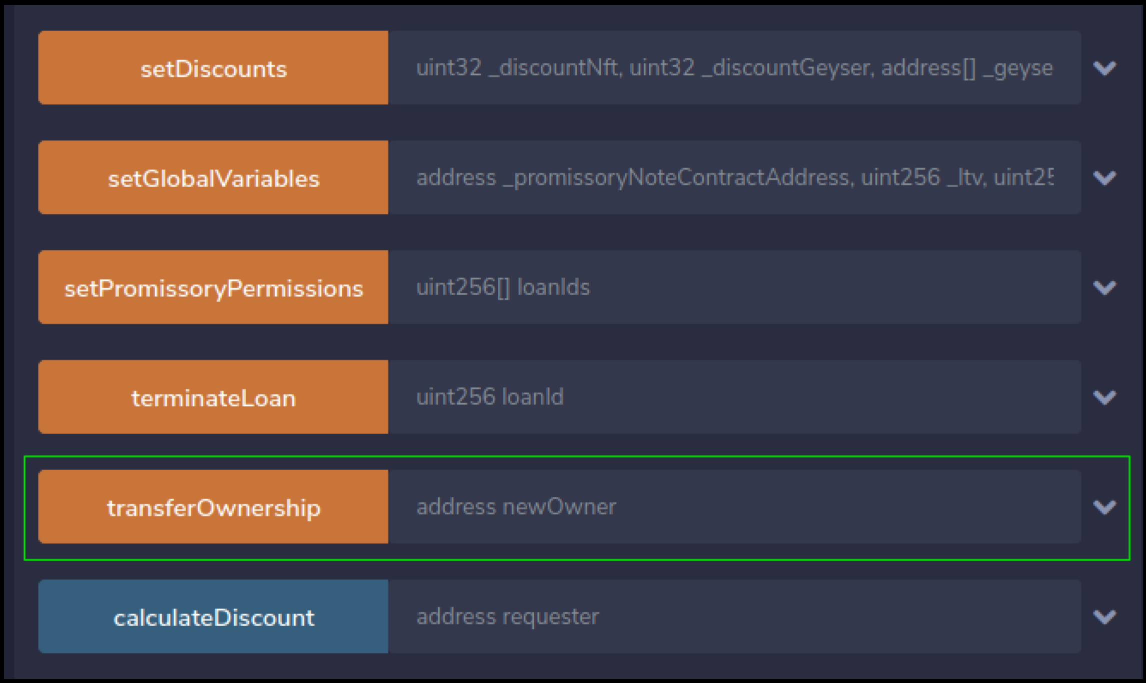
3.3 (HAL-03) LACK OF ADDRESS CONTROL ON THE FUNCTIONS - LOW

Description:

Address validation in contract LendingData.sol is missing. Lack of address validation has been found in the multiple functions. On the following functions, user supplied address values are assigned state variables directly.

Code Location:

LendingData.sol



LendingData.sol Line #~447-452

```
439     /**
440      * @notice Setter function for the discounts
441      * @param _discountNft Discount value for the Stater NFT holders
442      * @param _discountGeyser Discount value for the Stater liquidity mining participants
443      * @param _geyserAddressArray List of the Stater Geyser contracts
444      * @param _staterNftTokenIdArray Array of stater nft token IDs.
445      * @param _nftAddress List of the Stater NFT collections
446      */
447     function setDiscounts(uint32 _discountNft, uint32 _discountGeyser, address[] calldata _geyserAddressArray,
448         discountNft = _discountNft;
449         discountGeyser = _discountGeyser;
450         geyserAddressArray = _geyserAddressArray;
451         staterNftTokenIdArray = _staterNftTokenIdArray;
452         nftAddress = _nftAddress;
453     }
454
455     /**
```

LendingData.sol Line #~465-472

```
455     /**
456      * @notice Setter function
457      * @param _promissoryNoteContractAddress The address of the Stater promissory Note contract
458      * @param _ltv Value of Loan to value
459      * @param _installmentFrequency Value of installment frequency
460      * @param _installmentTimeScale The timescale of all loans.
461      * @param _interestRate Value of interest rate
462      * @param _interestRateToStater Value of interest rate to stater
463      * @param _lenderFee Value of the lender fee
464      */
465     function setGlobalVariables(address _promissoryNoteContractAddress, uint256 _ltv, uint256 _installmentFrequency, Time
466         ltv = _ltv;
467         installmentFrequency = _installmentFrequency;
468         installmentTimeScale = _installmentTimeScale;
469         interestRate = _interestRate;
470         interestRateToStater = _interestRateToStater;
471         lenderFee = _lenderFee;
472         promissoryNoteContractAddress = _promissoryNoteContractAddress;
473     }
474
```

LendingData.sol Line #~479-481

```
475     /**
476      * @notice Adds a new geyser address to the list
477      * @param geyserAddress The new geyser address
478      */
479     function addGeyserAddress(address geyserAddress) external onlyOwner {
480         geyserAddressArray.push(geyserAddress);
481     }
482
```

Risk Level:

Likelihood - 2

Impact - 2

Recommendation:

Although administrative restrictions are imposed to this function it is better to add proper address validation when assigning a value to a variable from user supplied inputs. As a better solution, a white-listing/black-listing should be applied on the related functions.

Listing 2

```
modifier validAddress(address addr) {
   require(addr != 0, "Value can not be null");
   require(addr != address(0), "Address cannot be 0x0");
   require(addr != address(this), "Address cannot be contract address");
   _;
 }
```

Remediation Plan:

Pending: Stater team will fix it in a future release.

3.4 (HAL-04) USAGE OF BLOCK-TIMESTAMP - LOW

Description:

During a manual review, we noticed the use of block.timestamp. The contract developers should be aware that this does not mean current time. Miners can influence the value of block.timestamp to perform Maximal Extractable Value (MEV) attacks. The use of block.timestamp creates a risk that time manipulation can be performed to manipulate price oracles. Miners can modify the timestamp by up to 900 seconds.

Code Location:

LendingData.sol Line #195-199

```
179  /**
180   * @notice The lender will approve the loan
181   * @param loanId The id of the loan
182   */
183  function approveLoan(uint256 loanId) external payable {
184    require(loans[loanId].lender == address(0), "Someone else payed for this loan before you");
185    require(loans[loanId].paidAmount == 0, "This loan is currently not ready for lenders");
186    require(loans[loanId].status == Status.LISTED, "This loan is not currently ready for lenders, check later");
187
188    uint256 discount = calculateDiscount(msg.sender);
189
190    // We check if currency is ETH
191    if ( loans[loanId].currency == address(0) )
192      require(msg.value >= loans[loanId].loanAmount.add(loans[loanId].loanAmount.div(lenderFee).div(discount)),"Not enou
193
194    // Borrower assigned , status is 1 , first installment ( payment ) completed
195    loans[loanId].lender = msg.sender;
196    loans[loanId].loanEnd = block.timestamp.add(loans[loanId].nrOfInstallments.mul(generateInstallmentFrequency()));
197    loans[loanId].status = Status.APPROVED;
198    loans[loanId].loanStart = block.timestamp;
199
200    // We send the tokens here
201    _transferTokens(msg.sender,loans[loanId].borrower,loans[loanId].currency,loans[loanId].loanAmount,loans[loanId].loan
202
```

LendingData.sol Line #220

```
// Borrower cancels a loan
function cancelLoan(uint256 loanId) external {
  require(loans[loanId].lender == address(0), "The loan has a lender , it cannot be cancelled");
  require(loans[loanId].borrower == msg.sender, "You're not the borrower of this loan");
  require(loans[loanId].status != Status.CANCELLED, "This loan is already cancelled");
  require(loans[loanId].status == Status.LISTED, "This loan is no longer cancellable");

  // We set its validity date as block.timestamp
  loans[loanId].loanEnd = block.timestamp;
  loans[loanId].status = Status.CANCELLED;

  // We send the items back to him
  _transferItems(
    address(this),
    loans[loanId].borrower,
    loans[loanId].nftAddressArray,
    loans[loanId].nftTokenIdArray,
    loans[loanId].nftTokenTypeArray
  );

  emit LoanCancelled(
    loanId,
    block.timestamp,
    Status.CANCELLED
  );
}

```

LendingData.sol Line #246

```
240    * @notice Borrower pays installments for the loan
241    * @param loanId The id of the loan
242    */
243   function payLoan(uint256 loanId) external payable {
244     require(loans[loanId].borrower == msg.sender, "You're not the borrower of this loan");
245     require(loans[loanId].status == Status.APPROVED, "This loan is no longer in the approval phase, check its status")
246     require(loans[loanId].loanEnd >= block.timestamp, "Loan validity expired");
247     require((msg.value > 0 && loans[loanId].currency == address(0) ) || ( loans[loanId].currency != address(0) && msg.
248 
249     uint256 paidByBorrower = msg.value > 0 ? msg.value : loans[loanId].installmentAmount;
250     uint256 amountPaidAsInstallmentToLender = paidByBorrower; // >> amount of installment that goes to lender
251     uint256 interestPerInstallement = paidByBorrower.mul(interestRate).div(100); // entire interest for installment
252     uint256 discount = calculateDiscount(msg.sender);
253     uint256 interestToStaterPerInstallement = interestPerInstallement.mul(interestRateToStater).div(100);
254 
255     if ( discount != 1 ){
256         if ( loans[loanId].currency == address(0) ){
257             require(msg.sender.send(interestToStaterPerInstallement.div(discount)), "Discount returnation failed");
258             amountPaidAsInstallmentToLender = amountPaidAsInstallmentToLender.sub(interestToStaterPerInstallement.div(
259         }
260         interestToStaterPerInstallement = interestToStaterPerInstallement.sub(interestToStaterPerInstallement.div(disc
261     }
262     amountPaidAsInstallmentToLender = amountPaidAsInstallmentToLender.sub(interestToStaterPerInstallement);
263 
264     loans[loanId].paidAmount = loans[loanId].paidAmount.add(paidByBorrower);
265     loans[loanId].nrOfPayments = loans[loanId].nrOfPayments.add(paidByBorrower.div(loans[loanId].installmentAmount));
266 
267     if (loans[loanId].paidAmount >= loans[loanId].amountDue)
268       loans[loanId].status = Status.LIQUIDATED;
269 
```

LendingData.sol Line #291-308

```
function terminateLoan(uint256 loanId) external {
  require(msg.sender == loans[loanId].borrower || msg.sender == loans[loanId].lender, "You can't access this loan");
  require((block.timestamp >= loans[loanId].loanEnd || loans[loanId].paidAmount >= loans[loanId].amountDue) || lackOfP
  require(loans[loanId].status == Status.LIQUIDATED || loans[loanId].status == Status.APPROVED, "Incorrect state of lo
  require(loans[loanId].status != Status.WITHDRAWN, "Loan NFTs already withdrawn");

  if ( lackOfPayment(loanId) ) {
    loans[loanId].status = Status.WITHDRAWN;
    loans[loanId].loanEnd = block.timestamp;
    // We send the items back to lender
    _transferItems(
      address(this),
      loans[loanId].lender,
      loans[loanId].nftAddressArray,
      loans[loanId].nftTokenIdArray,
      loans[loanId].nftTokenTypeArray
    );
  } else {
    if ( block.timestamp >= loans[loanId].loanEnd && loans[loanId].paidAmount < loans[loanId].amountDue ) {
      loans[loanId].status = Status.WITHDRAWN;
      // We send the items back to lender
      _transferItems(
        address(this),
        loans[loanId].lender,
        loans[loanId].nftAddressArray,
        loans[loanId].nftTokenIdArray,
        loans[loanId].nftTokenTypeArray
      );
    } else if ( loans[loanId].paidAmount >= loans[loanId].amountDue ){
      loans[loanId].status = Status.WITHDRAWN;
      // We send the items back to borrower
      _transferItems(
        address(this),
        loans[loanId].borrower,
        loans[loanId].nftAddressArray,
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Use block.number instead of block.timestamp or now to reduce the risk of MEV attacks. Check if the timescale of the project occurs across years, days and months rather than seconds. If possible, it is recommended to use Oracles.

Remediation Plan:

Solved: Stater team assumes that the use of block.timestamp is safe because their timescales are higher than 900 seconds.

3.5 (HAL-05) IGNORE RETURN VALUES - LOW

Description:

The return value of an external call is not stored in a local or state variable. In contract LendingData.sol, there are few instances where external methods are being called and return value(bool) are being ignored.

Affected Functions:

Listing 3

```
function createLoan
function approveLoan
function cancelLoan
function payLoan
function terminateLoan
function promissoryExchange
function setPromissoryPermissions
function setDiscounts
function setGlobalVariables
function addGeyserAddress
function addNftTokenId
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Add a return value check to avoid unexpected crash of the contract. Return value check will help in handling the exceptions better way.

Remediation Plan:

Risk Accepted: Stater team considers that if they add returns value they could exceed the code length limit.

3.6 (HAL-06) POSSIBLE RE-ENTRANCY - INFORMATIONAL

Description:

Calling external contracts is dangerous if some functions and variables are called after the external call. An attacker could use a malicious contract to perform a recursive call before calling function and take over the control flow. transfer function is executed without increasing the loanId value before. Thus, an attacker could perform a recursive call to execute malicious code.

Code Location:

LendingData.sol Line #~166-176

```
140
141         // Compute loan to value ratio for current loan application
142         require(_percent(loanAmount, assetsValue) <= ltv, "LTV exceeds maximum limit allowed");
143
144         // Computing the defaulting limit
145         if ( nrOfInstallments <= 3 )
146             loans[loanID].defaultingLimit = 1;
147         else if ( nrOfInstallments <= 5 )
148             loans[loanID].defaultingLimit = 2;
149         else if ( nrOfInstallments >= 6 )
150             loans[loanID].defaultingLimit = 3;
151
152         // Set loan fields
153         loans[loanID].nftTokenIdArray = nftTokenIdArray;
154         loans[loanID].loanAmount = loanAmount;
155         loans[loanID].assetsValue = assetsValue;
156         loans[loanID].amountDue = loanAmount.mul(interestRate.add(100)).div(100); // interest rate >> 20%
157         loans[loanID].nrOfInstallments = nrOfInstallments;
158         loans[loanID].installmentAmount = loans[loanID].amountDue.mod(nrOfInstallments) > 0 ? loans[loanID].amountDue.div(nrOfInstallments).add(1)
159         loans[loanID].status = Status.LISTED;
160         loans[loanID].nftAddressArray = nftAddressArray;
161         loans[loanID].borrower = msg.sender;
162         loans[loanID].currency = currency;
163         loans[loanID].nftTokenTypeArray = nftTokenTypeArray;
164
165         // Transfer the items from lender to stater contract
166         _transferItems(
167             msg.sender,
168             address(this),
169             nftAddressArray,
170             nftTokenIdArray,
171             nftTokenTypeArray
172         );
173
174         // Fire event
175         emit NewLoan(loanID, msg.sender, block.timestamp, currency, Status.LISTED, creationId);
176         ++loanID;
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

As possible, external calls should be at the end of the function in order to avoiding an attacker take over the control flow. In that case, increase loanId before call transfer function.

Remediation Plan:

Risk Accepted: Stater team considers appropriate the structure of the function but they will fix in a future release.

3.7 (HAL-07) FOR LOOP OVER DYNAMIC ARRAY - INFORMATIONAL

Description:

When smart contracts are deployed or functions inside them are called, the execution of these actions always requires a certain amount of gas, based on how much computation is needed to complete them. The Ethereum network specifies a block gas limit and the sum of all transactions included in a block cannot exceed the threshold.

Programming patterns that are harmless in centralized applications can lead to Denial of Service conditions in smart contracts when the cost of executing a function exceeds the block gas limit. Modifying an array of unknown size, that increases in size over time, can lead to such a Denial of Service condition.

A situation in which the block gas limit can be an issue is in sending funds to an array of addresses. Even without any malicious intent, this can easily go wrong. Just by having too large an array of users to pay can max out the gas limit and prevent the transaction from ever succeeding.

Code Location:

LendingData.sol Line #~342-360

```
336
337   /**
338    * @notice Used by the Promissory Note contract to change the ownership of the loan when the Promissory Note NFT is sold
339    * @param loanIds The ids of the loans that will be transferred to the new owner
340    * @param newOwner The address of the new owner
341    */
342   function promissoryExchange(uint256[] calldata loanIds, address payable newOwner) external {
343       require(msg.sender == promissoryNoteContractAddress, "You're not whitelisted to access this method");
344       for (uint256 i = 0; i < loanIds.length; ++i) {
345         require(loans[loanIds[i]].lender != address(0), "One of the loans is not approved yet");
346         require(promissoryPermissions[loanIds[i]] == msg.sender, "You're not allowed to perform this operation on loan");
347         loans[loanIds[i]].lender = newOwner;
348       }
349   }
350
351   /**
352    * @notice Used by the Promissory Note contract to approve a list of loans to be used as a Promissory Note NFT
353    * @param loanIds The ids of the loans that will be approved
354    */
355   function setPromissoryPermissions(uint256[] calldata loanIds) external {
356       for (uint256 i = 0; i < loanIds.length; ++i) {
357           require(loans[loanIds[i]].lender == msg.sender, "You're not the lender of this loan");
358           promissoryPermissions[loanIds[i]] = promissoryNoteContractAddress;
359       }
360   }
361
362   /**
```

LendingData.sol Line #~367-372

```
363      * @notice Liquidity mining participants or Stater NFT holders will be able to get some discount
364      * @param requester The address of the requester
365      */
366     function calculateDiscount(address requester) public view returns(uint256){
367       for (uint i = 0; i < staterNftTokenIdArray.length; ++i)
368           if ( IERC1155(nftAddress).balanceOf(requester,staterNftTokenIdArray[i]) > 0 )
369               return uint256(100).div(discountNft);
370         for (uint256 i = 0; i < geyserAddressArray.length; ++i)
371           if ( Geyser(geyserAddressArray[i]).totalStakedFor(requester) > 0 )
372               return uint256(100).div(discountGeyser);
373         return 1;
374     }
375
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Actions that require looping across the entire data structure should be avoided. If you absolutely must loop over an array of unknown size, then you should plan for it to potentially take multiple blocks, and therefore require multiple transactions.

Remediation Plan:

Risk Accepted: Stater team considers appropriate the use of loops.

3.8 (HAL-08) PRAGMA VERSION - INFORMATIONAL

Description:

LendingData.sol contract uses one of the latest pragma version (0.7.4) which was released back in October 19, 2020. The latest pragma version is (0.8.3) was released in April 2021. Many pragma versions have been released, going from version 0.6.x to the recently released version 0.8.x. in just 6 months.

Code Location:

LendingData.sol Line #2

```
1   // SPDX-License-Identifier: MIT
2   pragma solidity 0.7.4;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

In the Solitidy Github repository, there is a json file listing the bugs reported for each compiler version. No bugs have been found in > 0.7.3 versions and very few in 0.7.0 -- 0.7.3. The latest stable version is pragma 0.6.12. Furthermore, pragma 0.6.12 is widely used by Solidity developers and has been extensively tested in many security audits. We recommend using at minimum the latest stable version.

Reference: https://github.com/ethereum/solidity/blob/develop/docs/bugs_by_version.json

Remediation Plan:

Risk Accepted: Stater team considers appropriate the use of pragma 0.7.4 for the deployment to the mainnet.

3.9 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. 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 codebase.

Results:

LendingData.sol

```
INFO:Detectors:
LendingData.payLoan(uint256) (contracts/LendingData.sol#243-282) performs a multiplication on the result of a division:
	-interestPerInstallement = paidByBorrower.mul(interestRate).div(100) (contracts/LendingData.sol#251)
	-interestToStaterPerInstallement = interestPerInstallement.mul(interestRateToStater).div(100) (contracts/LendingData.sol#253)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
Reentrancy in LendingData.createLoan(uint256,uint256,address,uint256,address[],uint256[],string,LendingData.TokenType[]) (contracts/LendingData.sol#126-177):
	External calls:
	- _transferItems(msg.sender,address(this),nftAddressArray,nftTokenIdArray,nftTokenTypeArray) (contracts/LendingData.sol#166-172)
		- IERC721(nftAddressArray[i]).safeTransferFrom(from,to,nftTokenIdArray[i]) (contracts/LendingData.sol#519-523)
		- IERC1155(nftAddressArray[i]).safeTransferFrom(from,to,nftTokenIdArray[i],1,0x00) (contracts/LendingData.sol#525-531)
	State variables written after the call(s):
	- ++ loanID (contracts/LendingData.sol#176)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
LendingData.constructor(address,address,address[],uint256[])._nftAddress (contracts/LendingData.sol#107) lacks a zero-check on :
		- nftAddress = _nftAddress (contracts/LendingData.sol#108)
LendingData.constructor(address,address,address[],uint256[])._promissoryNoteContractAddress (contracts/LendingData.sol#107) lacks a zero-check on :
		- promissoryNoteContractAddress = _promissoryNoteContractAddress (contracts/LendingData.sol#111)
LendingData.setDiscounts(uint32,uint32,address[],uint256[],address)._nftAddress (contracts/LendingData.sol#447) lacks a zero-check on :
		- nftAddress = _nftAddress (contracts/LendingData.sol#452)
LendingData.setGlobalVariables(address,uint256,uint256,LendingData.TimeScale,uint256,uint256,uint32)._promissoryNoteContractAddress (contracts/LendingData.sol#465) lacks a zero-check on :
		- promissoryNoteContractAddress = _promissoryNoteContractAddress (contracts/LendingData.sol#472)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
LendingData.calculateDiscount(address) (contracts/LendingData.sol#366-374) has external calls inside a loop: IERC1155(nftAddress).balanceOf(requester,staterNftTokenIdArray[i]) > 0 (contracts/
LendingData.sol#368)
LendingData.calculateDiscount(address) (contracts/LendingData.sol#366-374) has external calls inside a loop: Geyser(geyserAddressArray[i_scope_0]).totalStakedFor(requester) > 0 (contracts/Len
dingData.sol#371)
LendingData._transferItems(address,address,address[],uint256[],LendingData.TokenType[]) (contracts/LendingData.sol#508-532) has external calls inside a loop: IERC721(nftAddressArray[i]).safeT
ransferFrom(from,to,nftTokenIdArray[i]) (contracts/LendingData.sol#519-523)
LendingData._transferItems(address,address,address[],uint256[],LendingData.TokenType[]) (contracts/LendingData.sol#508-532) has external calls inside a loop: IERC1155(nftAddressArray[i]).safe
TransferFrom(from,to,nftTokenIdArray[i],1,0x00) (contracts/LendingData.sol#525-531)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
Reentrancy in LendingData.approveLoan(uint256) (contracts/LendingData.sol#183-210):
	External calls:
	- _transferTokens(msg.sender,loans[loanId].borrower,loans[loanId].currency,loans[loanId].loanAmount,loans[loanId].loanAmount.div(lenderFee).div(discount)) (contracts/LendingData.sol#2
01)
		- require(bool,string)(IERC20(currency).transferFrom(from,to,qty1),Transfer of tokens to receiver failed) (contracts/LendingData.sol#550-554)
		- require(bool,string)(IERC20(currency).transferFrom(from,owner(),qty2),Transfer of tokens to Stater failed) (contracts/LendingData.sol#555-559)
	External calls sending eth:
	- _transferTokens(msg.sender,loans[loanId].borrower,loans[loanId].currency,loans[loanId].loanAmount,loans[loanId].loanAmount.div(lenderFee).div(discount)) (contracts/LendingData.sol#2
01)
		- require(bool,string)(to.send(qty1),Transfer of ETH to receiver failed) (contracts/LendingData.sol#561)
		- require(bool,string)(address(owner()).send(qty2),Transfer of ETH to Stater failed) (contracts/LendingData.sol#562)
	Event emitted after the call(s):
	- LoanApproved(loanId,msg.sender,block.timestamp,loans[loanId].loanEnd,Status.APPROVED) (contracts/LendingData.sol#203-209)
Reentrancy in LendingData.cancelLoan(uint256) (contracts/LendingData.sol#213-237):
	External calls:
	- _transferItems(address(this),loans[loanId].borrower,loans[loanId].nftAddressArray,loans[loanId].nftTokenIdArray,loans[loanId].nftTokenTypeArray) (contracts/LendingData.sol#224-230)
		- IERC721(nftAddressArray[i]).safeTransferFrom(from,to,nftTokenIdArray[i]) (contracts/LendingData.sol#519-523)
		- IERC1155(nftAddressArray[i]).safeTransferFrom(from,to,nftTokenIdArray[i],1,0x00) (contracts/LendingData.sol#525-531)
	Event emitted after the call(s):
	- LoanCancelled(loanId,block.timestamp,Status.CANCELLED) (contracts/LendingData.sol#232-236)
Reentrancy in LendingData.createLoan(uint256,uint256,address,uint256,address[],uint256[],string,LendingData.TokenType[]) (contracts/LendingData.sol#126-177):
	External calls:
	- _transferItems(msg.sender,address(this),nftAddressArray,nftTokenIdArray,nftTokenTypeArray) (contracts/LendingData.sol#166-172)
		- IERC721(nftAddressArray[i]).safeTransferFrom(from,to,nftTokenIdArray[i]) (contracts/LendingData.sol#519-523)
		- IERC1155(nftAddressArray[i]).safeTransferFrom(from,to,nftTokenIdArray[i],1,0x00) (contracts/LendingData.sol#525-531)
	Event emitted after the call(s):
	- NewLoan(loanID,msg.sender,block.timestamp,currency,Status.LISTED,creationId) (contracts/LendingData.sol#175)
Reentrancy in LendingData.payLoan(uint256) (contracts/LendingData.sol#243-282):
	External calls:
	- _transferTokens(msg.sender,loans[loanId].lender,loans[loanId].currency,amountPaidAsInstallmentToLender,interestToStaterPerInstallement) (contracts/LendingData.sol#271)
		- require(bool,string)(IERC20(currency).transferFrom(from,to,qty1),Transfer of tokens to receiver failed) (contracts/LendingData.sol#550-554)
		- require(bool,string)(IERC20(currency).transferFrom(from,owner(),qty2),Transfer of tokens to Stater failed) (contracts/LendingData.sol#555-559)
	External calls sending eth:
	- require(bool,string)(msg.sender.send(interestToStaterPerInstallement.div(discount)),Discount returnation failed) (contracts/LendingData.sol#257)
	- _transferTokens(msg.sender,loans[loanId].lender,loans[loanId].currency,amountPaidAsInstallmentToLender,interestToStaterPerInstallement) (contracts/LendingData.sol#271)
		- require(bool,string)(to.send(qty1),Transfer of ETH to receiver failed) (contracts/LendingData.sol#561)
		- require(bool,string)(address(owner()).send(qty2),Transfer of ETH to Stater failed) (contracts/LendingData.sol#562)
	Event emitted after the call(s):
	- LoanPayment(loanId,block.timestamp,msg.value,amountPaidAsInstallmentToLender,interestPerInstallement,interestToStaterPerInstallement,loans[loanId].status) (contracts/LendingData.sol
#273-281)
```

FINDINGS & TECH DETAILS

```
Reentrancy in LendingData.terminateLoan(uint256) (contracts/LendingData.sol#289-335):
        External calls:
        - _transferItems(address(this),loans[loanId].lender,loans[loanId].nftAddressArray,loans[loanId].nftTokenIdArray,loans[loanId].nftTokenTypeArray) (contracts/LendingData.sol#299-305)
                - IERC721(nftAddressArray[i]).safeTransferFrom(from,to,nftTokenIdArray[i]) (contracts/LendingData.sol#519-523)
                - IERC1155(nftAddressArray[i]).safeTransferFrom(from,to,nftTokenIdArray[i],1,0x00) (contracts/LendingData.sol#525-531)
        - _transferItems(address(this),loans[loanId].lender,loans[loanId].nftAddressArray,loans[loanId].nftTokenIdArray,loans[loanId].nftTokenTypeArray) (contracts/LendingData.sol#310-316)
                - IERC721(nftAddressArray[i]).safeTransferFrom(from,to,nftTokenIdArray[i]) (contracts/LendingData.sol#519-523)
                - IERC1155(nftAddressArray[i]).safeTransferFrom(from,to,nftTokenIdArray[i],1,0x00) (contracts/LendingData.sol#525-531)
        - _transferItems(address(this),loans[loanId].borrower,loans[loanId].nftAddressArray,loans[loanId].nftTokenIdArray,loans[loanId].nftTokenTypeArray) (contracts/LendingData.sol#320-326)
                - IERC721(nftAddressArray[i]).safeTransferFrom(from,to,nftTokenIdArray[i]) (contracts/LendingData.sol#519-523)
                - IERC1155(nftAddressArray[i]).safeTransferFrom(from,to,nftTokenIdArray[i],1,0x00) (contracts/LendingData.sol#525-531)
        Event emitted after the call(s):
        - ItemsWithdrawn(loanId,msg.sender,loans[loanId].status) (contracts/LendingData.sol#330-334)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
LendingData.payLoan(uint256) (contracts/LendingData.sol#243-282) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(loans[loanId].loanEnd >= block.timestamp,Loan validity expired) (contracts/LendingData.sol#246)
LendingData.terminateLoan(uint256) (contracts/LendingData.sol#289-335) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)((block.timestamp >= loans[loanId].loanEnd || loans[loanId].paidAmount >= loans[loanId].amountDue) || lackOfPayment(loanId),Not possible to finish this loan yet)
 (contracts/LendingData.sol#291)
        - block.timestamp >= loans[loanId].loanEnd && loans[loanId].paidAmount < loans[loanId].amountDue (contracts/LendingData.sol#307)
LendingData.lackOfPayment(uint256) (contracts/LendingData.sol#424-426) uses timestamp for comparisons
        Dangerous comparisons:
        - loans[loanId].status == Status.APPROVED && loans[loanId].loanStart.add(loans[loanId].nrOfPayments.mul(generateInstallmentFrequency())) <= block.timestamp.sub(loans[loanId].defaultin
gLimit.mul(generateInstallmentFrequency())) (contracts/LendingData.sol#425)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Different versions of Solidity is used in :
        - Version used: ['0.7.4', '>=0.6.0<0.8.0', '>=0.6.2<0.8.0']
        - 0.7.4 (contracts/LendingData.sol#2)
        - 0.7.4 (node_modules/multi-token-standard/contracts/interfaces/IERC1155.sol#1)
        - >=0.6.0<0.8.0 (node_modules/openzeppelin-solidity/contracts/access/Ownable.sol#3)
        - >=0.6.0<0.8.0 (node_modules/openzeppelin-solidity/contracts/introspection/ERC165.sol#3)
        - >=0.6.0<0.8.0 (node_modules/openzeppelin-solidity/contracts/introspection/IERC165.sol#3)
        - >=0.6.0<0.8.0 (node_modules/openzeppelin-solidity/contracts/math/SafeMath.sol#3)
        - >=0.6.0<0.8.0 (node_modules/openzeppelin-solidity/contracts/token/ERC1155/ERC1155Holder.sol#3)
        - >=0.6.0<0.8.0 (node_modules/openzeppelin-solidity/contracts/token/ERC1155/ERC1155Receiver.sol#3)
        - >=0.6.0<0.8.0 (node_modules/openzeppelin-solidity/contracts/token/ERC1155/IERC1155Receiver.sol#3)
        - >=0.6.0<0.8.0 (node_modules/openzeppelin-solidity/contracts/token/ERC20/IERC20.sol#3)
        - >=0.6.0<0.8.0 (node_modules/openzeppelin-solidity/contracts/token/ERC721/ERC721Holder.sol#3)
        - >=0.6.2<0.8.0 (node_modules/openzeppelin-solidity/contracts/token/ERC721/IERC721.sol#3)
        - >=0.6.0<0.8.0 (node_modules/openzeppelin-solidity/contracts/token/ERC721/IERC721Receiver.sol#3)
        - >=0.6.0<0.8.0 (node_modules/openzeppelin-solidity/contracts/utils/Context.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
Redundant expression "this (node_modules/openzeppelin-solidity/contracts/utils/Context.sol#21)" inContext (node_modules/openzeppelin-solidity/contracts/utils/Context.sol#15-24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
Reentrancy in LendingData.approveLoan(uint256) (contracts/LendingData.sol#183-210):
        External calls:
        - _transferTokens(msg.sender,loans[loanId].borrower,loans[loanId].currency,loans[loanId].loanAmount,loans[loanId].loanAmount.div(lenderFee).div(discount)) (contracts/LendingData.sol#2
01)
                - require(bool,string)(to.send(qty1),Transfer of ETH to receiver failed) (contracts/LendingData.sol#561)
                - require(bool,string)(address(owner()).send(qty2),Transfer of ETH to Stater failed) (contracts/LendingData.sol#562)
        Event emitted after the call(s):
        - LoanApproved(loanId,msg.sender,block.timestamp,loans[loanId].loanEnd,Status.APPROVED) (contracts/LendingData.sol#203-209)
Reentrancy in LendingData.payLoan(uint256) (contracts/LendingData.sol#243-282):
        External calls:
        - require(bool,string)(msg.sender.send(interestToStaterPerInstallement.div(discount)),Discount returnation failed) (contracts/LendingData.sol#257)
        State variables written after the call(s):
        - loans[loanId].paidAmount = loans[loanId].paidAmount.add(paidByBorrower) (contracts/LendingData.sol#264)
        - loans[loanId].nrOfPayments = loans[loanId].nrOfPayments.add(paidByBorrower.div(loans[loanId].installmentAmount)) (contracts/LendingData.sol#265)
        - loans[loanId].status = Status.LIQUIDATED (contracts/LendingData.sol#268)
Reentrancy in LendingData.payLoan(uint256) (contracts/LendingData.sol#243-282):
        External calls:
        - require(bool,string)(msg.sender.send(interestToStaterPerInstallement.div(discount)),Discount returnation failed) (contracts/LendingData.sol#257)
        - _transferTokens(msg.sender,loans[loanId].lender,loans[loanId].currency,amountPaidAsInstallmentToLender,interestToStaterPerInstallement) (contracts/LendingData.sol#271)
                - require(bool,string)(to.send(qty1),Transfer of ETH to receiver failed) (contracts/LendingData.sol#561)
                - require(bool,string)(address(owner()).send(qty2),Transfer of ETH to Stater failed) (contracts/LendingData.sol#562)
        Event emitted after the call(s):
        - LoanPayment(loanId,block.timestamp,msg.value,amountPaidAsInstallmentToLender,interestPerInstallement,interestToStaterPerInstallement,loans[loanId].status) (contracts/LendingData.sol
#273-281)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (node_modules/openzeppelin-solidity/contracts/access/Ownable.sol#54-57)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (node_modules/openzeppelin-solidity/contracts/access/Ownable.sol#63-67)
supportsInterface(bytes4) should be declared external:
        - ERC165.supportsInterface(bytes4) (node_modules/openzeppelin-solidity/contracts/introspection/ERC165.sol#35-37)
onERC1155Received(address,address,uint256,uint256,bytes) should be declared external:
        - ERC1155Holder.onERC1155Received(address,address,uint256,uint256,bytes) (node_modules/openzeppelin-solidity/contracts/token/ERC1155/ERC1155Holder.sol#11-13)
onERC1155BatchReceived(address,address,uint256[],uint256[],bytes) should be declared external:
        - ERC1155Holder.onERC1155BatchReceived(address,address,uint256[],uint256[],bytes) (node_modules/openzeppelin-solidity/contracts/token/ERC1155/ERC1155Holder.sol#15-17)
onERC721Received(address,address,uint256,bytes) should be declared external:
        - ERC721Holder.onERC721Received(address,address,uint256,bytes) (node_modules/openzeppelin-solidity/contracts/token/ERC721/ERC721Holder.sol#20-22)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

According to the test results, most of the findings found by slither were considered as false positives. Relevant findings were reviewed by the auditors.

THANK YOU FOR CHOOSING

HALBORN