CERTIK

Security Assessment

Huma Finance - Audit

CertiK Verified on Dec 30th, 2022

CertiK Verified on Dec 30th, 2022

Huma Finance - Audit

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
Lending	Ethereum	Manual Review, Static Analysis
LANGUAGE	**TIMELINE**	**KEY COMPONENTS**
Solidity	Delivered on 12/30/2022	N/A

CODEBASE

https://github.com/00labs/huma-contracts

...View All

COMMITS

base: 1075003788f343a1a30606834fd9ac073b23c23a

update1: 40b1972f4ddde1490faa4b9dce01f007fc1653ae

update2: 15a2c52c7f90eb4528f2fd4976393f897c4196e5

...View All

Vulnerability Summary

14	12	1	0	1	0	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined	Unresolved

0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
1	Major	1 Mitigated	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
1	Medium	1 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
5	Minor	4 Resolved, 1 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
7	Informational	7 Resolved	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | HUMA FINANCE - AUDIT

CODEBASE | HUMA FINANCE - AUDIT

Repository

https://github.com/00labs/huma-contracts

Commit

base: 1075003788f343a1a30606834fd9ac073b23c23a
update1: 40b1972f4ddde1490faa4b9dce01f007fc1653ae
update2: 15a2c52c7f90eb4528f2fd4976393f897c4196e5

AUDIT SCOPE | HUMA FINANCE - AUDIT

22 files audited ● 10 files with Resolved findings ● 12 files without findings

ID	Repo	Commit	File	SHA256 Checksum
● HDH	00labs/huma-contracts	1075003	contracts/HDT/HDT.sol	c52b7e9c4272fd04b47504ff4c7efcdf368a051c009a0cf7d83053b53e95cf41
● BSB	00labs/huma-contracts	1075003	contracts/libraries/BaseStructs.sol	92e194027707b4fadfdecb33b2c2b5fb84ef276496ba04b3780ace8c791f9318
● BCP	00labs/huma-contracts	1075003	contracts/BaseCreditPool.sol	370d72d0928c12a150ebd560d60ed0f06ed19558a84f8e4ff8ae241c7d1d0af8
● BFM	00labs/huma-contracts	1075003	contracts/BaseFeeManager.sol	f7209fda2a5fa4ad5ca702ddd5efb0f0ca61319ab2849370c3986c67588d06af
● BPC	00labs/huma-contracts	1075003	contracts/BasePoolConfig.sol	7fa4e1f6072ee6c56a7214141d0237898f55dfc3d0526d46fb10f087272914b2
● BPB	00labs/huma-contracts	1075003	contracts/BasePool.sol	d35ae7b06f3a87bc98d80a4b3bb39699663ee33a255563c09d191aa677529e17
● BPS	00labs/huma-contracts	1075003	contracts/BasePoolStorage.sol	d1c5e4a30818dd37afa7704c810928da81ba35f94375e4d8cf339507c0caf49c
● EAN	00labs/huma-contracts	1075003	contracts/EvaluationAgentNFT.sol	411c314244029f7fa613d93c254afe53d71553c7a0e616c8eed0ec5cfabe0bb6
● HCB	00labs/huma-contracts	1075003	contracts/HumaConfig.sol	9c8435e5f45c92add24177156027fadfe82db91d4fa4108ec31c017da7f29dbc
● RFP	00labs/huma-contracts	1075003	contracts/ReceivableFactoringPool.sol	f57e3ae9b3ef1436363a10afc927cfe8529ac7c299b0f24ae51cb0f7b025090d
○ IHD	00labs/huma-contracts	1075003	contracts/HDT/interfaces/IHDT.sol	f05f5d17a6a9e2f732ec3cf095a3407ad82ead2f2da69470df621504b178b49b
○ HDS	00labs/huma-contracts	1075003	contracts/HDT/HDTStorage.sol	4c26633335b190ff2ac8e800c0f234d34c88edf2252ad99b1a8aede72c84aa97
○ ICB	00labs/huma-contracts	1075003	contracts/interfaces/ICredit.sol	456b8bc0d905824f6e08dacc0eac2b352b0f4d1535c3c4f447383f9d1c8a7502

ID	Repo	Commit	File	SHA256 Checksum
IFM	00labs/huma-contracts	1075003	contracts/interfaces/IFeeManager.sol	60c8e5e20e347f0aaecc35b8e0d645f3f2dbc79890dcc0f85cc4c7f52892c2a3
ILP	00labs/huma-contracts	1075003	contracts/interfaces/ILiquidityProvider.sol	1e7bf3d6a2219b0f5c7a4d12b8173b616ec1bae139c2522e7c911015fa38bda2
IPB	00labs/huma-contracts	1075003	contracts/interfaces/IPool.sol	9d1494e3e0971881cba2d9a45672a2e8f43e6e4d988fb143e37bc038f163c08f
IRB	00labs/huma-contracts	1075003	contracts/interfaces/IReceivable.sol	8754df2a2fd5c32906a713068e93b6c34f929e2f1c942935ca17a1a49a4d8511
TCB	00labs/huma-contracts	1075003	contracts/openzeppelin/TimelockController.sol	1a5a0b043e8284c20ae68182993c530962f6790381b37cc5391e13e1477e0b38
TUP	00labs/huma-contracts	1075003	contracts/openzeppelin/TransparentUpgradeableProxy.sol	45285378ffc0d04183d271351c7f192ba4a8b717a96839f3ca21b703e0533d99
BCS	00labs/huma-contracts	1075003	contracts/BaseCreditPoolStorage.sol	35efc58464616beb3ea8023d50ce9f0fdb7e105cc87fa91bafa987dcd06eae00
ERR	00labs/huma-contracts	1075003	contracts/Errors.sol	7557212536ba6ff4e6d2de760d91525e869b0a2a51780730472a8a5a40275958
RFS	00labs/huma-contracts	1075003	contracts/ReceivableFactoringPoolStorage.sol	0acf6102f172904b27c3598f5e96a577ed3fe200fd4735d4a94bd30a73dc005e

APPROACH & METHODS | HUMA FINANCE - AUDIT

This report has been prepared for Huma Finance to discover issues and vulnerabilities in the source code of the Huma Finance - Audit project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

FINDINGS | HUMA FINANCE - AUDIT

14	0	1	1	5	7
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Huma Finance - Audit. Through this audit, we have uncovered 14 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
GLOBAL-01	**Centralization Related Risks**	**Centralization / Privilege**	**Major**	**Mitigated**
BPC-01	Potential Lost Protocol Fees	Logical Issue	Medium	Resolved
BPB-01	Sets Approval Of `newPoolConfig` To `0` Instead Of `oldConfig`	Logical Issue	Minor	Resolved
BPC-02	`initialize()` Can Be Called Multiple Times	Logical Issue	Minor	Resolved
CON-01	Missing Checks	Volatile Code	Minor	Resolved
EAN-01	Lack Of Access Control	Logical Issue	Minor	Resolved
RFP-01	Third Party Dependencies	Volatile Code	Minor	Acknowledged
BCP-01	`approveCredit()` Comment Is Unclear	Inconsistency	Informational	Resolved
BPC-03	Variables Not Initialized On Deployment	Volatile Code	Informational	Resolved
CON-02	Typos	Coding Style	Informational	Resolved

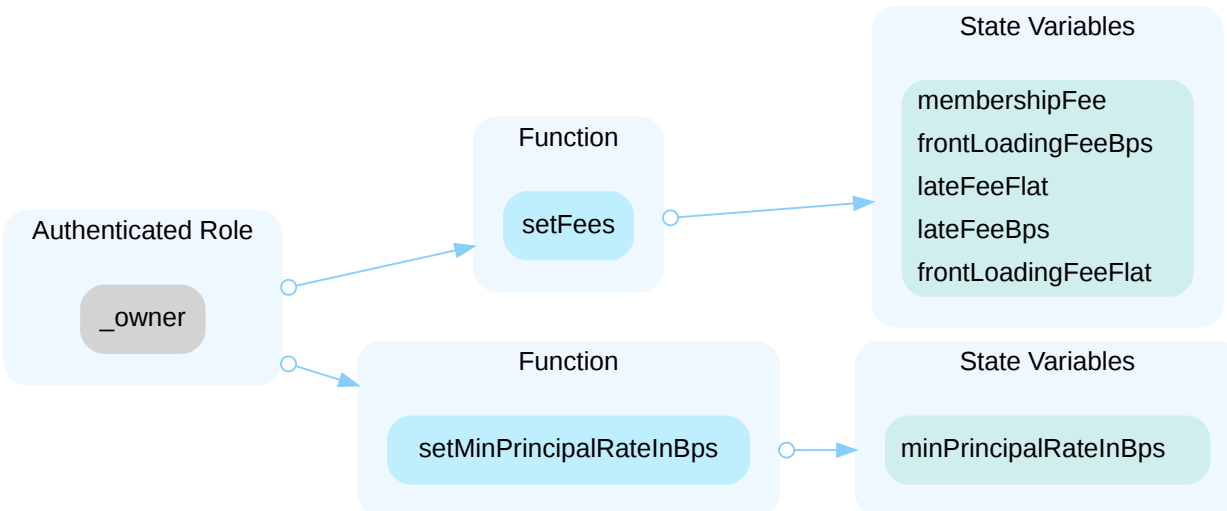
ID	Title	Category	Severity	Status
CON-03	Time Units Can Be Used Directly	Coding Style	Informational	Resolved
CON-04	Possible Overflow	Logical Issue, Volatile Code	Informational	Resolved
COT-01	Typos And Errors In Change To Centralized Roles	Logical Issue	Informational	Resolved
GLOBAL-02	Tokens Can Be Stuck In The Protocol	Logical Issue	Informational	Resolved

GLOBAL-01 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization / Privilege	**Major**		**Mitigated**

Description

In the contract `BaseFeeManager` the role `onlyOwner` has authority over the functions shown in the diagram below.

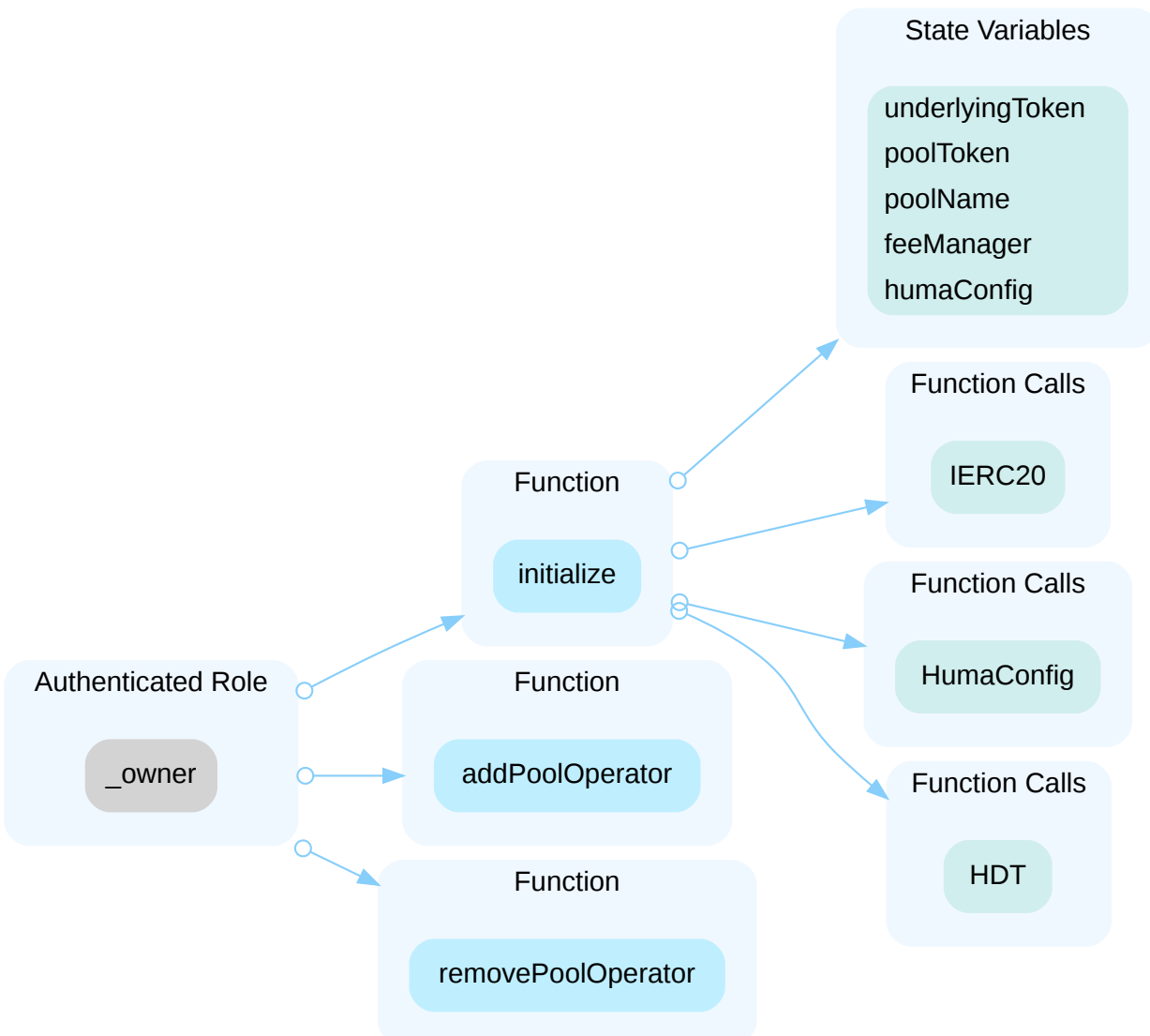


Any compromise to the `onlyOwner` account may allow the hacker to take advantage of this by being able to change the fees to any number they choose. A malicious user could also set `minPrincipalRateInBps` to an arbitrarily high number.

In the contract `BasePoolConfig` the role `_onlyOwnerOrHumaMasterAdmin`, `onlyPoolOwnerOrEA`, `onlyPoolOwnerTreasury`, and `humaConfig.owner()` have authority over those functions listed below and the role `onlyOwner` has authority over those functions shown in the diagram:

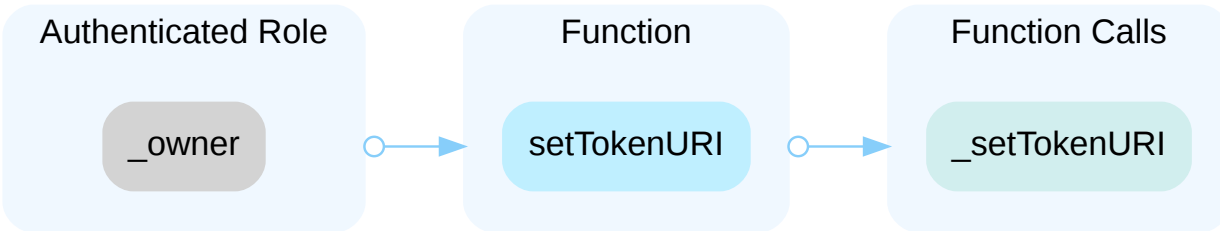
- `setAPR()` - set the default APR for the pool, only callable by `_onlyOwnerOrHumaMasterAdmin`.
- `setCreditApprovalExpiration()` - set the default for credit approval expiration, only callable by `_onlyOwnerOrHumaMasterAdmin`.
- `setEARewardsAndLiquidity()` - set the default for rewardsRate and liquidityRate, only callable by `_onlyOwnerOrHumaMasterAdmin`.
- `setEvaluationAgent()` - add an evaluation agent to be able to approve loans, only callable by `_onlyOwnerOrHumaMasterAdmin`.
- `setFeeManager()` - set the fee manager for the protocol, only callable by `_onlyOwnerOrHumaMasterAdmin`.
- `setHumaConfig()` - set the address for the `HumaConfig` file, only callable by `_onlyOwnerOrHumaMasterAdmin`.
- `setMaxCreditLine()` - set the default max size of each loan and credit line, only callable by `_onlyOwnerOrHumaMasterAdmin`.

- `setPool()` - set the pool address, only callable by `_onlyOwnerOrHumaMasterAdmin`.
- `setPoolLiquidityCap()` - set the upper bound that approved depositors can deposit, only callable by `_onlyOwnerOrHumaMasterAdmin`.
- `setPoolOwnerRewardsAndLiquidity()` - set the default for owner rewards, only callable by `_onlyOwnerOrHumaMasterAdmin`.
- `setPoolPayPeriod()` - set the pay cycle in amount of periods, only callable by `_onlyOwnerOrHumaMasterAdmin`.
- `setPoolName()` - set a string for the pool name, only callable by `_onlyOwnerOrHumaMasterAdmin`.
- `setPoolToken()` - set the address of the pool token, only callable by `_onlyOwnerOrHumaMasterAdmin`.
- `setReceivableRequiredInBps()` - set the rate in basis points for receivable. This can be over 100% because the pool might want to have over-collateralization, only callable by `_onlyOwnerOrHumaMasterAdmin`.
- `setWithdrawalLockoutPeriod()` - set the default amount of time a lender has to make after making a deposit, only callable by `_onlyOwnerOrHumaMasterAdmin`.
- `withdrawEAFee()` - withdraw rewards to EA, only callable by `onlyPoolOwnerOrEA`.
- `withdrawPoolOwnerFee()` - withdraw rewards to pool owner, only callable by `onlyPoolOwnerTreasury`.
- `withdrawProtocolFee()` - withdraw protocol rewards to treasury account, only callable by `humaConfig.owner()`.



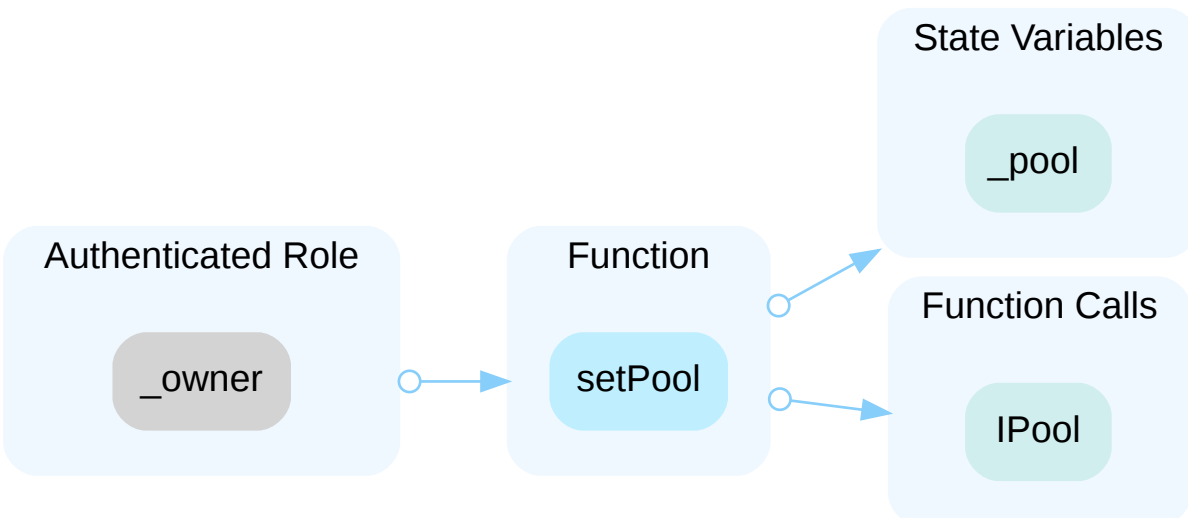
Any compromise to the `onlyOwner`, `_onlyOwnerOrHumaMasterAdmin`, `onlyPoolOwnerOrEA`, `onlyPoolOwnerTreasury`, and `humaConfig.owner()` account(s) may allow the hacker to take advantage of these functions.

In the contract `EvaluationAgentNFT` the role `onlyOwner` has authority over the functions shown in the diagram below.



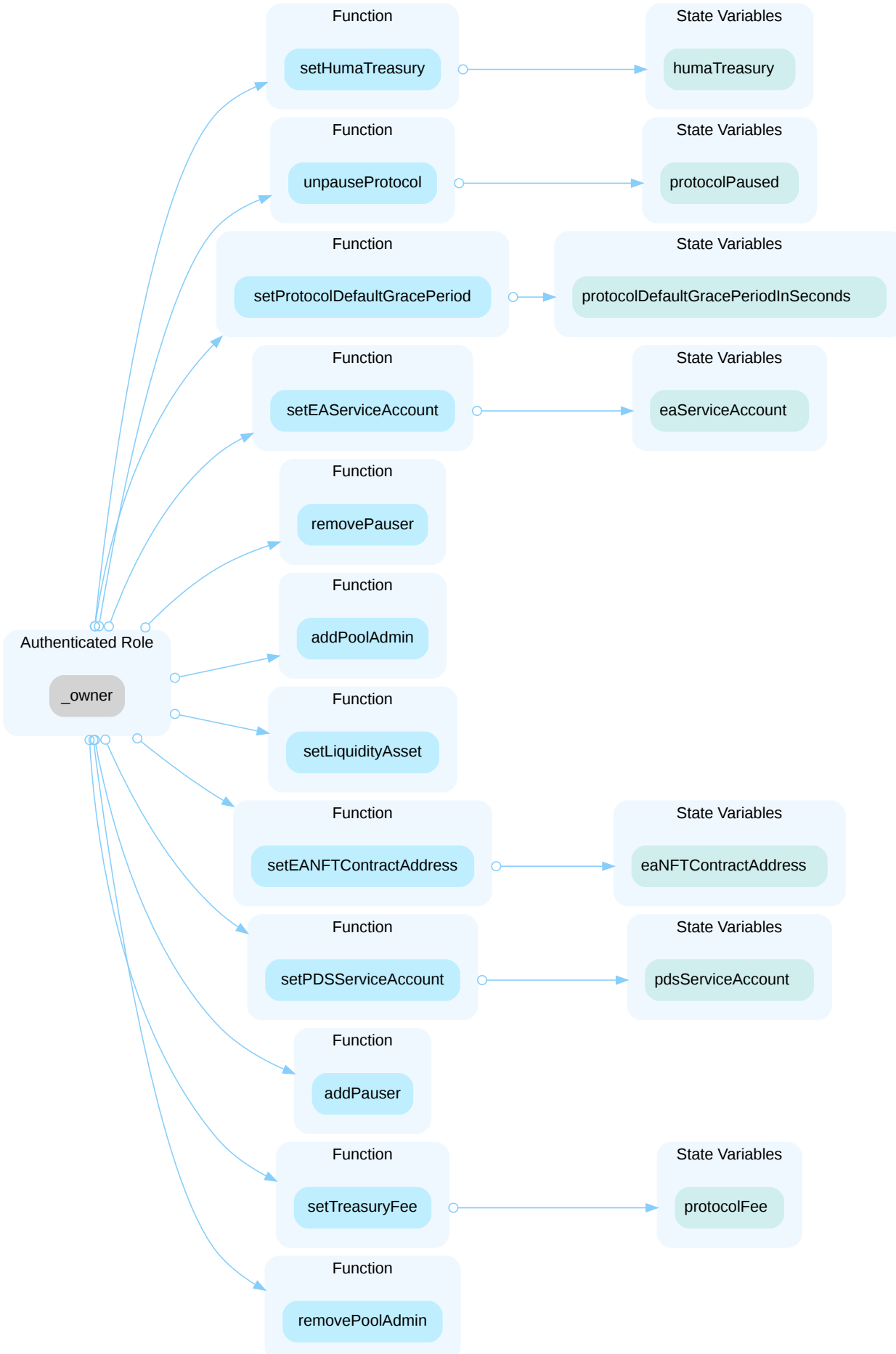
Any compromise to the `onlyOwner` account may allow the hacker to take advantage of this and change the `TokenURI`.

In the contract `HDT` the role `onlyOwner` has authority over the functions shown in the diagram below.



Any compromise to the `onlyOwner` account may allow the hacker to take advantage of this and change the address of the pool.

In the contract `HumaConfig` the role `onlyOwner` has authority over the functions shown in the diagram below.


Function
setHumaTreasury
State Variables
humaTreasury
Function
unpauseProtocol
State Variables
protocolPaused
Function
setProtocolDefaultGracePeriod
State Variables
protocolDefaultGracePeriodInSeconds
Function
setEAServiceAccount
State Variables
eaServiceAccount
Function
removePauser
Function
addPoolAdmin
Authenticated Role
_owner
Function
setLiquidityAsset
Function
setEANFTContractAddress
State Variables
eaNFTContractAddress
Function
setPDSServiceAccount
State Variables
pdsServiceAccount
Function
addPauser
Function
setTreasuryFee
State Variables
protocolFee
Function
removePoolAdmin

Any compromise to the `onlyOwner` account may allow the hacker to take advantage of these and set the treasury fee to 50%. A malicious user could also pause and unpause the protocol for favorable transactions. They can set the treasury and default grace period. However, the default grace period has a minimum value it cannot be set below.

In the contract `BaseCreditPool` the role `onlyEAServiceAccount` has authority over the functions listed below:

- `approveCredit()` - approves a user's credit line.
- `changeCreditLine()` - if a user wants to change their credit line to lower, they may do that at their own will. If the credit line is to be increased, then it has to be done by an `onlyEAServiceAccount` address.
- `extendCreditLineDuration()` - extend the maturity date of the credit line.

Any compromise to the `onlyEAServiceAccount` account may allow the hacker to take advantage of this.

In the contract `BasePool` the role `onlyPoolOwnerTreasuryOrEA`, `_onlyOwnerOrHumaMasterAdmin`, `_onlyPoolOperator`, and `_onlyApprovedLender` have authority over the functions listed below:

- `makeInitialDeposit()` - this allows the pool owner treasury or EA of the pool to make the first deposit before the pool goes live.
- `_deposit()` - an internal function that can only be called if the user is an approved lender for the pool.
- `addApprovedLender()` - once an entity is approved off-chain, they are then added on-chain by this function by `_onlyPoolOperator`.
- `disablePool()` - turns the pool off, only callable by `_onlyPoolOperator`.
- `enablePool()` - turns the pool back on, only callable by `_onlyOwnerOrHumaMasterAdmin`.
- `removeApprovedLender()` - revokes an approved lenders status, only callable by `_onlyPoolOperator`.
- `setPoolConfig()` - takes an address as input, this allows it to point towards a configuration contract, only callable by `_onlyOwnerOrHumaMasterAdmin`.
- `updateCoreData()` - allows the caller to change the underlying token address, pool token address, `humaConfig` address, and `feeManager` address. Only callable by `_onlyOwnerOrHumaMasterAdmin`.

Any compromise to the `onlyPoolOwnerTreasuryOrEA`, `_onlyOwnerOrHumaMasterAdmin`, `_onlyPoolOperator`, and `_onlyApprovedLender` account(s) may allow the hacker to take advantage of this.

In the contract `ReceivableFactoringPool` the role `onlyPDSServiceAccount`, `_poolConfig.onlyPoolOwner()`, and `onlyEAServiceAccount` have authority over the functions listed below:

- `onReceivedPayment()` - only callable by `PDSServiceAccount`, which sends a `paymentId` to make a payment on behalf of a `borrower` when the pool receives a payment from a receivable.
- `changeCreditLine()` - if a user wants to change their credit line to lower, they may do that at their own will. If the credit line is to be increased, then it has to be done by an `onlyEAServiceAccount` address.
- `markPaymentInvalid()` - only callable by `PDSServiceAccount`, which marks a payment as invalid to avoid repeat payments.

- `processPaymentAfterReview()` - only callable by `_poolConfig.onlyPoolOwner()`, which either marks payments that were marked for review as invalid or processes them.
- `approveCredit()` - only callable by `onlyEAServiceAccount`, approves a receivable factoring.

Any compromise to the `onlyPDSServiceAccount`, `_poolConfig.onlyPoolOwner()`, and `onlyEAServiceAccount` account(s) may allow the hacker to mark payments as invalid and mark accounts as paid off.

In the contract `TransparentUpgradeableProxy.sol` the role `admin` has the authority to upgrade the implementation contract. Any compromise to the `admin` account can allow the attacker to upgrade the contract without the community's commitment. If an attacker compromises the account, they can change the implementation of the contract and drain tokens from the contract.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement;
  AND

- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles;
  OR
- Remove the risky functionality.

Noted: Recommend considering the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

`[CertiK]` : The client made the following changes and has launched with multisigs and timelocks. The multisigs all require 2 signatures out of 3 signers and the timelocks have a minimum delay of 24 hours. Information on these can be found in this doc.

`[Huma Finance]` : "Our plan has already been to launch with multisig and timelock, and transition to DAO and timelock. We have published our contract addresses, multisig and signee addresses as suggested. We will continue to do so as we launch more pools."

"We also implemented a few additional changes to make our contract operations safer and more transparent.

- Required all pool owners to be multisig and timelock. For each pool, there are actually two timelocks. poolTimeLock for pool owner administrative functions. poolProxyTimeLock is for contract upgrade. (Commit: 1393a7d56233956784bc4f5c44fc743a0f5af0a6)
- Replaced in-house pause solution with OZ Defender's pause (Commit: 680acd26f54111948de8d56afe96f25c6e748c12)
- Introduced two new roles at pool level: poolOperator (Commit: f50030905bc8d3949d50d601d899b44f73129a8c) who handles lender KYC/KYB check and approves potential lenders, and poolOwnerTreasury (Commit: 3d651691e118232b3e0521fa5ba028cae8a825c2) who handles all the financial interest of a pool owner.
- Added capability to flag abnormal payment transactions for review (Commit: d87c6478acabe1e1ae8a7f8219590a429065bae3) to limit security exposure in cases when critical admin or service accounts are compromised. For further protection, all the transactions triggered after a review are also put behind a timelock so that we still have time to catch the issue even if the review account is also compromised."

BPC-01 | POTENTIAL LOST PROTOCOL FEES

Category	Severity	Location	Status
Logical Issue	Medium	contracts/BasePoolConfig.sol (base): 388~391	Resolved

Description

If `treasuryAddress` is the zero address, then protocol fee will be lost when `withdrawProtocolFee()` is called.

Recommendation

We recommend handling the case when the `treasuryAddress` is the zero address.

Alleviation

`[CertiK]` : The client made the recommended changes in commit 9a582d5d9eadf0f2846bedc8d53aa592be4debc3.

BPB-01 SETS APPROVAL OF `newPoolConfig` TO `0` INSTEAD OF `oldConfig`

Category	Severity	Location	Status
Logical Issue	Minor	contracts/BasePool.sol (base): 233~234	Resolved

Description

`_safeApproveForPoolConfig(0)` is called after `_poolConfig` is changed to the `newPoolConfig`. Thus, it does not update the approval of the `oldConfig` to be 0.

Recommendation

We recommend calling `_safeApproveForPoolConfig(0)` before `_poolConfig` is updated to ensure intended behavior occurs.

Alleviation

`[CertiK]`: The client made the recommended changes in commit 16d9e82e2eda5f175cd5386b962de6c3ed6fcd6c.

BPC-02 | `initialize()` CAN BE CALLED MULTIPLE TIMES

Category	Severity	Location	Status
Logical Issue	Minor	contracts/BasePoolConfig.sol (base): 135	Resolved

Description

The `initialize()` function in `BasePoolConfig.sol` can be called multiple times by the owner.

Recommendation

We recommend adding a modifier to ensure this can only be called once.

Alleviation

`[CertiK]` : The client made the recommended changes in commit d49865726d7c7dd1128770a608a47e4ea834a56f.

CON-01 | MISSING CHECKS

Category	Severity	Location	Status
Volatile Code	Minor	contracts/BaseCreditPool.sol (base): 129; contracts/BasePool.sol (base): 171; contracts/BasePoolConfig.sol (base): 221~222, 269, 307, 313~314, 320, 364, 364	Resolved

Description

In `BasePoolConfig.sol`:

- In `setEARewardsAndLiquidity()`, it is not checked that the input `rewardsRate` and `liquidityRate` are less than or equal to `10000`.
- In `setHumaConfig()`, it is not checked that the input `_humaConfig` is not the zero address.
- In `setPoolLiquidityCap()`, it is not checked that the input `liquidityCap` is greater than zero.
- In `setPoolOwnerRewardsAndLiquidity()`, it is not checked that the input `rewardsRate` and `liquidityRate` are less than or equal to `10000`.
- In `setPoolPayPeriod()`, it is not checked that the input `periodInDays` is non-zero.
- In `setPoolToken()`, it is not checked that the input `_poolToken` is not the zero address.
- In `withdrawEAFee()`, it is not checked that the input `amount` is non-zero.
- In `withdrawPoolOwnerFee()`, it is not checked that the input `amount` is non-zero.

In `BaseCreditPool.sol`:

- In `approveCredit()`, it is not checked that the credit limit is less than or equal to the max credit line. It is possible this could accidentally be set higher than the maximum amount as the inputs of approve credit are never checked against the requested credit.

In `BasePool.sol`:

- In `reverseIncome()`, if the calculated `poolIncome` is larger than the `_totalPoolValue`, this will cause a revert and not allow a user to payoff their debt early. We recommend handling this case as is done in `distributeLosses()`.

Recommendation

We recommend adding the checks mentioned above to prevent unexpected errors.

Alleviation

`[CertiK]` : The client made the recommended changes in commits 6d7472b429960b855c3dacd7b0a3fb73ad55de4e and 5eaa97fecbdf11986ad8ebedc31a37383aa183a2.

EAN-01 | LACK OF ACCESS CONTROL

Category	Severity	Location	Status
Logical Issue	Minor	contracts/EvaluationAgentNFT.sol (base): 18	Resolved

Description

The `mintNFT()` function in `EvaluationAgentNFT.sol` has no access control, allowing anyone to mint themselves an NFT. While it is not possible to get access to privileged functions with simply minting an NFT, it is is still possible for a malicious user to mint themselves an NFT and attempt to pose as an EA in a phishing attempt.

Recommendation

We recommend restricting access to this function or adding a comment warning users that owning the NFT does not necessarily mean someone has authority.

Alleviation

`[CertiK]` : The client resolved the issue by adding a comment warning users in commit adee0f6056a0996e15003a08b6197f3adbecd405.

RFP-01 | THIRD PARTY DEPENDENCIES

Category	Severity	Location	Status
Volatile Code	Minor	contracts/ReceivableFactoringPool.sol (NRoriginal): 281	Acknowledged

Description

The contract is serving as the underlying entity to interact with third party verification and payment processors. The scope of the audit treats 3rd party entities as black boxes and assume their functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of 3rd parties can possibly create severe impacts, such as increasing fees of 3rd parties, migrating to new LP pools, etc.

Recommendation

We understand that the business logic of Huma Finance requires interaction with certain off-chain engines. We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

Alleviation

`[CertiK]` : The client acknowledged the finding and provided the following quote.

`[Huma Finance]` : "Our integration with the third party is very loosely coupled. After they receive and process a payment, they will emit an event. Huma Sentinel will monitor the event and check to make sure the money has been transferred to our account, it calls our contract to bookkeep this payment and process the disbursement accordingly. If the third party is compromised, it can fake the event emission, unless it can also fake a payment to our contract, Huma Sentinel will not be fooled to trigger disbursement."

BCP-01 | `approveCredit()` COMMENT IS UNCLEAR

Category	Severity	Location	Status
Inconsistency	Informational	contracts/BaseCreditPool.sol (base): 121	Resolved

Description

In the comment for `approveCredit()` it says it approves the credit request with the terms on record. However, it uses the inputs not the terms on record for the `creditLimit`, `intervalInDays`, `remainingPeriods`, and `aprInBps`.

Recommendation

We recommend ensuring the comment line matches the intended functionality.

Alleviation

`[CertiK]`: The client made the recommended changes in commit 91f4601b0bd8823105d6fee20c613fa0e3fba4bb.

BPC-03 | VARIABLES NOT INITIALIZED ON DEPLOYMENT

Category	Severity	Location	Status
Volatile Code	Informational	contracts/BasePoolConfig.sol (base): 25, 29, 31, 33, 35, 37, 42, 47, 49	Resolved

Description

When `BasePoolConfig` is deployed, there is minimal initialization for variables in the configuration. Because of this, the deployment script needs to be thoroughly checked to ensure that all variables are initialized to proper values to ensure issues do not occur.

Recommendation

To minimize this accidental error, we recommend adding these variables to the initializer.

Alleviation

`[CertiK]` : The client made the recommended changes in commit f6ce68bb72d494fb39fee8898dbdd914092f7e54.

CON-02 | TYPOS

Category	Severity	Location	Status
Coding Style	Informational	contracts/BaseCreditPool.sol (base): 49, 237, 239, 239, 350, 387, 413, 476, 520, 525, 652, 676, 677, 682; contracts/BaseFeeManager.sol (base): 195, 271; contracts/BasePool.sol (base): 86, 179; contracts/BasePoolConfig.sol (base): 23, 36, 43, 303, 562; contracts/BasePoolStorage.sol (base): 39; contracts/HDT/HDT.sol (base): 81, 111; contracts/HumaConfig.sol (base): 141, 217; contracts/ReceivableFactoringPool.sol (base): 204, 224, 277; contracts/libraries/BaseStructs.sol (base): 9	Resolved

Description

The following typos are found throughout the repo:

In `BasePool`, the following typos were found:

- depisit should be spelled deposit.
- reqirements should be spelled requirements.

In `HumaConfig`, the following typos were found:

- "* @param valid The new validity status a Liquidity Asset in Pools." should be something similar to "* @param valid The new validity status of a Liquidity Asset in Pools."
- sentivity should be spelled sensitivity.

In `HDT`, the following typos were found:

- brun should be spelled burn
- `_owner` is the owner of the token contract, however the amount is the amount that the input `account` can withdraw. We think this should be `account` instead of `_owner`.

In `BasePoolStorage`, the following typos were found:

- whether should be capitalized.

In `BasePoolConfig`, the following typos were found:

- bais should be spelled basis.
- depositers should be spelled depositors.
- M0difier should be spelled Modifier.
- Line 23 and 36 are outdated comments. We recommend updating the comments.

In `BaseStructs`, the following typos were found:

- "credit originated" should just be credit or explain where it originated from.

In `BaseFeeManager`, the following typos were found:

- stotal should be spelled total.
- assersion should be spelled assertion

In `BaseCreditPool`, the following typos were found:

- distribut should be spelled distribute.
- "Positive correction is generated becasue of a drawdown within this period," should have a "." instead of a ",".
- distributed should be changed distribute.
- "be add" should be changed to "will be added".
- themselvers should be spelled themselves.
- becasue should be spelled because.
- calcuate should be spelled calculate.
- actuall should be spelled actual.
- indciating should be spelled indicating.
- initate should be spelled initiate.
- creditRecrodStatic should be corrected to creditRecordStatic.
- deaulted should be corrected to defaulted.
- shwos should be changed to shows.

In `ReceivableFactoringPool`, the following typos were found:

- assset should be spelled as asset.
- dispersement should be spelled disbursement.
- addrescredit should be spelled addresscredit.

Recommendation

We recommend correcting the following typos.

Alleviation

`[CertiK]` : The client made the recommended changes in commit a20a027bdac89f9647ada9a51edde81749c8bc6.

CON-03 | TIME UNITS CAN BE USED DIRECTLY

Category	Severity	Location	Status
Coding Style	Informational	contracts/BaseFeeManager.sol (base): 21, 22; contracts/BasePoolConfig.sol (base): 62, 63; contracts/BasePoolStorage.sol (base): 11	Resolved

Description

Suffixes like seconds, minutes, hours, days and weeks after literal numbers can be used to specify units of time where seconds are the base unit and units are considered naively in the following way:

- 1 == 1 seconds
- 1 minutes == 60 seconds
- 1 hours == 60 minutes
- 1 days == 24 hours
- 1 weeks == 7 days

The gas is almost the same usage on either method, but this increases code readability.

Recommendation

We recommend replacing the value assigned to the variables linked above using the built in time units.

Alleviation

`[CertiK]` : The client made the recommended changes in commit d628a1f56f138da7ad1b2979a00ca00e2942cd25.

CON-04 | POSSIBLE OVERFLOW

Category	Severity	Location	Status
Logical Issue, Volatile Code	Informational	contracts/BaseCreditPool.sol (base): 139~141, 145, 159, 173, 215, 225, 323, 445, 477, 478, 487, 544, 627, 630, 639, 654, 685, 727, 793; contracts/BaseFeeManager.sol (base): 269, 276, 292, 293, 294; contracts/ReceivableFactoringPool.sol (base): 258	Resolved

Description

There is casting from `uint256` to lower integers such as `uint96`, which can overflow and will not return an error. Similarly there is also casting between `uint` and `int`, which can cause unexpected errors.

For example in `BaseCreditPool`: feesAndInterestDue is a `uint96`. If it is greater than the max positive `int96`, then it will end up being a large negative value when it is cast as an `int`. For tokens with 18 decimals, this means it would have to be over 39 billion.

For example in `BaseFeeManager`: If `fees+interest` is greater than or equal to `2**96`, then it will overflow. For tokens with 18 decimals, this means that it would have to be over 79 billion.

For example in `ReceivableFactoringPool.sol`: If `receivableAmount` is greater than or equal to `2**88` it will overflow. For tokens with 18 decimals, this means it would have to be over 309 million.

Recommendation

These scenarios are unlikely considering the amount of credit a user must be given, but we recommend ensuring that it is not possible to be given a credit line large enough to cause these overflows.

Alleviation

`[CertiK]`: The client made the recommended changes in commits eeab9821f6328180da1d6dadd763cee04bd30364 and 999b6a1253e54cf9a8c78ec540967bf0741c39a5.

COT-01 TYPOS AND ERRORS IN CHANGE TO CENTRALIZED ROLES

Category	Severity	Location	Status
Logical Issue	Informational	contracts/BaseCreditPool.sol (update1): 109~110, 583, 584, 587, 746; contracts/BasePool.sol (update1): 187~189, 201; contracts/BasePoolConfig.sol (update1): 137, 182, 183, 188, 189, 190, 192, 194, 612, 617, 618, 619, 621, 623	Resolved

Description

The following typos and errors were found in the code:

- In `BaseCreditPool.sol`, in the comment for the function `_makePayment()`, “manul” is used instead of “manual”, “Outier” is used instead of “Outlier”, “actuall” is used instead of “actual”. Also there is a missing comment for the third return variable `isReviewRequired`.
- In `BaseCreditPool.sol`, in the comment for the event `PaymentMade`, it still reads “it is possible for someone to make payment on behalf of the borrower.” However, the logic of `makePayment()` has changed so payments can only be made by the borrower or come from a receivable. In addition, the `PaymentMade` event emitted in `_makePayment()` has `msg.sender` for the `by` address, as the tokens are always payed by the borrower in this case either this should be changed to `borrower` or the comment for `by` address changed.
- In `BasePoolConfig.sol`, "_opeartor" is used instead of "_operator" in multiple places.
- In `BasePool.sol`, the comment for the function `addApprovedLender()` mentions only the pool owner can call this function, when now only a `poolOperator` can call the function. In addition, only a `poolOperator` can disable a pool, we recommend having the comments in `BasePoolConfig.sol` reflect that operators can also disable pools.

Recommendation

We recommend fixing the typos and errors.

Alleviation

`[CertiK]`: The client made all the recommended changes in commits 24985288a085509c5f222be6fea85100f7ea47c4, 388ee57bed092afda21cdd8bd8ed0d56ae770cb7, and 9dd26c935854444df0bc2a052718c3225d0bdee3.

GLOBAL-02 | TOKENS CAN BE STUCK IN THE PROTOCOL

Category	Severity	Location	Status
Logical Issue	Informational		Resolved

Description

If tokens are accidentally sent to the protocol directly, they will be stuck forever.

Recommendation

To prevent against loss of user funds, we recommend adding a `recoverTokens()` function that allows any tokens *not* supported by the pool to be withdrawn or to add a comment warning users that tokens sent directly to the contract cannot be recovered.

Alleviation

`[CertiK]` : The client resolved the issue by adding a comment warning that any tokens sent directly to it will be lost in commit f9853061e01a407a9b4be88d5c3887d9715bfbfd.

APPENDIX | HUMA FINANCE - AUDIT

Finding Categories

Categories	Description
Centralization / Privilege	Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.
Logical Issue	Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.
Coding Style	Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.
Inconsistency	Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE

FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | Securing the Web3 World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.