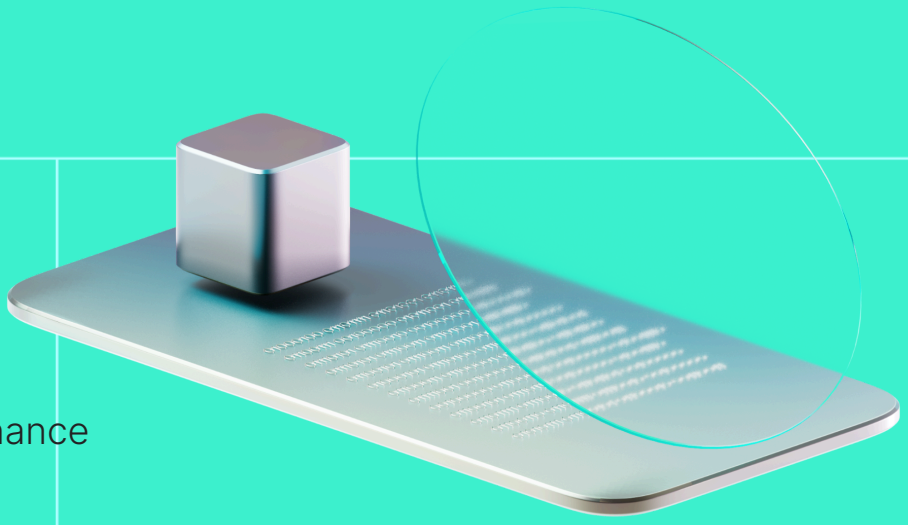
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: Clearpool.finance

Date: 8 Jan, 2024

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

We thank Clearpool.finance for allowing us to conduct a Smart Contract Security Assessment. This document outlines our methodology, limitations, and results of the security assessment.

Clearpool.finance is an efficient Defi credit marketplace (higher lender APR) yet very liquid (for lenders) product that is highly customizable and can work for any borrower type.

Platform: EVM

Language: Solidity

Tags: Lending Market

Timeline: 06.12.2023 - 08.01.2024

Methodology: Link

Last review scope

Repository	https://github.com/clearpool-finance/open-term-pools
Commit	cd0aa788f690bcfa507b9592bfe089ba169ca130

View full scope

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

Audit Summary

10/10	9.5/10	100%	9/10
Security score	Code quality score	Test coverage	Documentation quality score

Total: 9.8/10

The system users should acknowledge all the risks summed up in the risks section of the report.

4	2	1	1
Total Findings	Resolved	Acknowledged	Mitigated

Findings by severity	Findings Number	Resolved	Mitigated	Acknowledged
Critical	0	0	0	0
High	0	1	0	0
Medium	0	0	1	0
Low	0	1	0	1

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for Clearpool.finance
Audited By	Carlo Parisi | SC Lead Auditor at Hacken OÜ Roman Tiutiun | SC Auditor at Hacken OÜ
Approved By	Przemyslaw Swiatowiec | SC Audits Expert at Hacken OÜ
Website	https://clearpool.finance/
Changelog	22.12.2023 – Preliminary Report 08.01.2024 - Secondary Report

Introduction

Hacken OÜ (Consultant) was contracted by Clearpool.finance (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

System Overview

Clearpool.finance is a Decentralized Finance ecosystem first-ever permissionless marketplace for unsecured institutional liquidity with the following contracts permissionless single-borrower pools enable institutions to raise short-term capital while providing DeFi lenders access to risk-adjusted returns based on interest rates derived by market consensus:

- PoolFactory — a smart contract for creating term pools.
- PoolMaster — a flexible lending product that has higher lender APR, potentially high flexibility for lenders, and can be customized and used by any borrower type.
- BondNFT — a contract for minting tokens ERC1155 standard.
- Auction — a contract for allowing participants to bid for the pool's cpTokens (the total debt of the pool).
- UtilsGuard — an abstract contract providing a set of error handling and utility functions.

- RewardAsset — a library providing a set of functions for managing and updating data related to reward assets, including their rates, magnified reward corrections, and withdrawal amounts.
- NFTDescriptor — a library facilitates the generation of URIs for NFTs based on the provided parameters.

Privileged roles

- The privileged roles of the PoolMaster contract:
  - onlyPoolFactory is able to withdrawReward().
  - onlyBorrower is able to repay the full amount, repay current period debt and closePool, increase repayment frequency if pool is active, decrease or increase max capacity, change minimum deposit amount, change APR.
  - onlyAuction is able to process pool debt claims and process auction start.
  - onlyGovernor is able to request a repayment, change the pool protocol fee, change the pool borrower address, set the reward asset and rating, and set the pool status to Default.
- The privileged roles of the PoolFactory contract:
  - onlyOwner is able to update currency allowance in the protocol, set the address of the quadReader contract, update the pool beacon, update treasury address, update bond beacon, change the grace period, change the penalty interest value, set the reward asset and rating, and set auction.

- The privileged roles of the BondNFT contract:
    - onlyMinter is able to mint a new ERC1155 and multiple tokens and assign it to the specified address, burn a specified amount of an ERC1155 token.
- The privileged roles of the Auction contract:
    - checkBidder ensures that the caller of a function is neither the borrower of the specified pool nor blacklisted as a bidder and can bid on a pool.
    - onlyPoolAddr restricts the execution of a function to only those calls where the specified address is recognized as a valid pool by the factory contract. If the address is not a valid pool, the function call will be reverted.
    - onlyActive ensures that the function can only be called when the auction associated with the specified pool is still active. If the auction has ended, the function call will be reverted.

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **9** out of **10**.

- Technical documentation is sufficient,
- Functional requirements are mostly provided, however:

- The documentation provides an incomplete description of the KYC process.

Code quality

The total Code Quality score is **9.5** out of **10**.

- Best practice violations (I03)

Test coverage

Code coverage of the project is **100%** (branch coverage).

Security score

Upon completion of the audit, it was discovered that the code contained **1** high, **1** medium, and **2** low severity issues. All issues, with the exception of the second low severity issue (L02), were resolved. This led to a final security score of **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.8**. The system users should acknowledge all the risks summed up in the risks section of the report.

Risks

- The repository contains dependencies that are out of the audit scope. Contracts, being composable, introduce the possibility of security risks associated with out-of-scope elements, and evaluating the security of these elements is beyond the scope of this audit.

- Contracts can be upgraded after deployment, but these changes must be approached with caution as they can potentially introduce critical vulnerabilities.
- The *changeProtocolFee()* function, responsible for updating the protocol fee, lacks an upper limit constraint. Without an upper boundary, the function permits the setting of an excessively high protocol fee, potentially reaching 100%. This absence of a reasonable upper limit could lead to unintended consequences and undermine the stability of the protocol.
- The *setAuction()* function, designed to update the auction address, introduces a potential vulnerability. If the auction address is used in frontends to display available auctions, and a change occurs while there is an ongoing auction, this situation may introduce inconsistencies in the displayed information.
- Borrowers and lenders have the option to utilize Permissioned Pools, which incorporate on-chain Quadrata KYC verification. It should be noted that the KYC process is required to be used only in Permissioned Pools and borrowers are able to create pools that do not verify the KYC status of users. Furthermore, KYC verifications and ratings are implemented as third-party, off-chain measures. Quadrata protocol, as a 3rd party provider, is out of the scope of this audit.

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

H01. Severe requirements violation - incorrect implementation of KYC for borrower

Impact	Medium
Likelihood	High

The implementation of the system does not adhere to functional requirements. *Clearpool* is described in the documentation as follows:

"Borrowers need to onboard the same way as currently on Clearpool (MLA, KYC + rating)".

The vulnerability lies in the fact that the KYC does not check the address of the borrower for KYC, and it does not explicitly check whether the *borrower* has completed the KYC process. This means that the borrower's KYC status is not verified during the pool creation process.

```
  // LINE 152
if (kycRequired && address(quadrataReader) == address(0)) revert
ActionNotAllowed();
```

Also, a *changeBorrower()* function in *PoolMaster.sol* contains a potential documentation violation related to the KYC requirements for borrowers during the changing of the borrower.

```
function changeBorrower(
    address _newBorrower
  ) external onlyGovernor nonZeroAddress(_newBorrower) nonSameAddress(borrower,
_newBorrower) {
    borrower = _newBorrower;
    emit BorrowerChanged(_newBorrower);
  }
```

Borrowers can create pools without passing the KYC implies the unauthorized pool creation, and *changeBorrower()* is able to add a new borrower without performing or checking KYC introduces a risk of unauthorized individuals opening and borrowing from a pool.

Proof of Concept:

- Deploy contract.
- Invoke *createPool()* with all valid parameters.
- The Pool can be created without KYC.
- Access the Pool and initiate the *changeBorrower()* function exclusively through the *onlyGovernor* role.
- The *onlyGovernor* can change a borrower without proper KYC.

Path: ./contracts/PoolMaster.sol: changeBorrower();
./contracts/PoolFactory.sol: createPool();

Recommendation: To address this vulnerability, the code should be modified to explicitly check and ensure that the borrower has completed the KYC process. This may involve adding additional logic to *createPool()* and *changeBorrower()* functions to validate the borrower's KYC status before allowing the creation of a new pool and change borrower.

Found in: 3c15738

Status: Fixed (Revised commit: cd0aa78)

Remediation: The KYC status of borrowers is verified in *permissioned pools*, where the *kycRequired* pool variable is set to *true*.

Medium

M01. Incomplete eligibility check in _beforeTokenTransfer function

Impact	Medium
Likelihood	Medium

The *_beforeTokenTransfer()* function within the contract checks the eligibility of the recipient *to* using the *onlyEligible* modifier but neglects to verify the eligibility of the sender *from*.

```
  function _beforeTokenTransfer(
   address from,
   address to,
   uint256 amount
) internal override onlyEligible(to) {
```

```
    super._beforeTokenTransfer(from, to, amount);
}
```

Transactions originating from ineligible senders may not undergo the required eligibility checks, leading to potential unauthorized transfers.

Path: ./contracts/PoolMaster.sol : _beforeTokenTransfer()

Recommendation: To rectify this vulnerability, it is advisable to enhance the *_beforeTokenTransfer()* function to incorporate a comprehensive eligibility check for both the sender and recipient addresses. This can be achieved by extending the *onlyEligible* modifier to contain the sender as well. The modified implementation ensures that both parties involved in a token transfer meet the specified eligibility criteria, strengthening the contract against potential unauthorized transactions.

Found in: 3c15738

Status: Mitigated

Remediation: The previous token owner (token sender, *from*) is validated on *supply* function.

Low

L01. Undocumented Functionality

Impact	Low

Likelihood	Low

Thorough documentation is crucial to facilitate a clear understanding of contract purpose and provide guidance to interact with the contract accurately while minimizing the risk of potential vulnerabilities or misuse. To enhance this aspect, it is recommended to establish comprehensive documentation covering all non-standard functionalities.

Inside the *PoolMaster* contract, there are instances of burn(address(this), 'amount') accompanied by the comment */// burn all frozen tokens*. However, the documentation lacks explicit guidance on what is the reason for the presence of tokens, and what prompts their burning.

```
    // burn all freezed tokens
line 417
    _burn(address(this), balanceOf(address(this)));


// burn round freezed tokens
line 435
    _burn(address(this), roundInfo.debtAmount);
```

Also, the absence of documentation detailing the *Overdue* logic in *PoolMaster.sol*. This omission leaves without essential information about the conditions, and actions associated with the *Overdue* logic.

Furthermore, the *changeNoticePeriod()* function lacks detailed explanations in the documentation regarding the constraints introduced by this change.

```
if (_newNoticePeriod > minimumNoticePeriod) revert ActionNotAllowed();
```

Without comprehensive documentation, users might misunderstand or misuse functionalities such as the *_burn* operation, *Overdue* logic and *changeNoticePeriod().*

Path: ./contracts/PoolMaster.sol : changeNoticePeriod(), repayAll(), processDebtClaim()

Recommendation: Each contract function should have a clear and comprehensive description that outlines its purpose, inputs, outputs, and behavior under different conditions. Additionally, any relevant limitations, edge cases, and potential risks should be documented to guide users in making informed decisions while interacting with the contract. This is especially important for contracts with non-standard or complex logic.

Found in: 3c15738

Status: Fixed (Revised commit: f9000d7)

Remediation: NatSpec and in-line comments were added.

L02. Lack of handling for currencies with decimals

Impact	Low
Likelihood	Low

The contract *Utils.sol* contains functions *toWei()* and *fromWei()* designed to convert between amounts, considering the currency's decimal precision. The potential issue with the current implementation is that it does not handle scenarios where the token has more than 18 decimals.

```
function toWei(uint256 amount_, uint256 decimals) internal pure returns
(uint256) {
   return amount_ * 10 ** (18 - decimals);
}

function fromWei(uint256 amount_, uint256 decimals) internal pure returns
(uint256) {
   return amount_ / 10 ** (18 - decimals);
}
```

Using a token with more than 18 decimals in *toWei()* and *fromWe()* functions may result in a reverted transaction due to overflow.

Path: ./contracts/utils/Utils.sol.sol : toWei(), fromWei()

Recommendation: To mitigate the risk of overflow when handling tokens with more than 18 decimals, it is recommended to include a check before whitelisting a token.

Found in: 3c15738

Status: Acknowledged

Informational

I01. Variable name shadowing issues in PoolMaster.sol contract function

This issue manifests in the current implementation, where variable names within a contract's functions overshadow variables of the same name within the contract scope itself. Such a situation can lead to confusion and unintended consequences during code execution.

The *symbol* variable in the *PoolMaster.sol* contract is causing shadowing issues with the *symbol()* function from the *ERC20* standard. This occurs in functions such as *__init()*.

Path: ./contracts/PoolMaster.sol : __init()

Recommendation: To mitigate the shadowing issue and enhance clarity in the code, consider renaming the owner variable in the *PoolMaster.sol* contract to a more specific name that does not conflict with the *symbol()* function inherited from *ERC20Upgradeable* contract.

Found in: 3c15738

Status: Fixed (Revised commit: f9000d7)

Remediation: The *symbol* variable was renamed to *poolSymbol*.

I02. Typos in the code

The provided code segment includes a comment with a typo indicating the creation of an NFT contract. However, the subsequent line of code is initializing a *bondNFT* contract:

```
/// Creating a new `nft contract` with parameters;
   bool success = bondNFT.__init(address(this), '');
```

Path: ./contracts/PoolMaster.sol : __init()

Recommendation: Update the comment to accurately reflect that the line of code initializes a *bondNFT* contract rather than creating a new one.

Found in: 3c15738

Status: Fixed (Revised commit: f9000d7)

Remediation: Typo was fixed.

I03. Best practices violations

There are multiple occasions where the Checks-Effects-Interactions (CEI) pattern is broken in *supply()* and *redeem()* functions in *PoolMaster.sol* and *bid()* and *increaseBid()* in *Auction.sol*.

While each of these functions is explicitly marked as *nonReentrant*, it's important to note that adherence to the CEI pattern is a fundamental best practice in smart contract development aimed at mitigating reentrancy attacks. The observed divergence from this pattern in the specified functions warrants attention and consideration for enhanced security measures.

Path: ./contracts/PoolMaster.sol : supply() redeem();
./contracts/Auction.sol.sol : increaseBid() bid()

Recommendation: Consider reordering the operations in functions to follow the CEI pattern more strictly. Move interactions to the end of the functions, ensuring that state modifications are completed before interacting. This can help reduce

the risk of reentrancy-related vulnerabilities and enhance the overall security of the smart contract.

Found in: 3c15738

Status: Acknowledged (Revised commit: d3a48dd)

Remediation: The CEI pattern was introduced in *supply()*, *redeem()*, and *increaseBid()* functions. The pattern was not introduced in *bid()* function as according to the team: *The requested changes increase the contract size by adding additional variables to store the previous bidder's address and amount, also a duplicate if will be required.*

I04. Missing event indexes

In Solidity, events play a crucial role in facilitating communication between smart contracts and external applications. To optimize event searches and enable more efficient log analysis, developers can leverage indexed parameters within events in *PoolFactory.sol*.

Path: ./contracts/PoolFactory.sol : *

Recommendation: Use indexed events to keep track of a smart contract's activity after it is deployed, which is helpful in reducing overall Gas.

Found in: 3c15738

Status: Fixed (Revised commit: f9000d7)

Remediation: Event indexes were introduced.

I05. Implement lowerlLimit for auction duration in setAuctionDuration function

The *setAuctionDuration* function, responsible for updating the auction duration, should include a lower limit to ensure that the auction duration is reasonable and not set to an extremely short interval, such as 1 second.

Path: ./contracts/Auction.sol : setAuctionDuration()

Recommendation: Introduce a lower limit check within the *setAuctionDuration()* function to prevent setting an unreasonably short auction duration. This limit ensures that the auction duration remains within a practical range.

Status: Fixed (Revised commit: f9000d7)

Remediation: The lower limit check was implemented.

I06. Constructor usage _disableInitializers

In *Auction.sol* contract, recommended *_disableInitializers()* call should be added to the constructor. This precautionary measure ensures that the initializer is locked within the context of the logic contract. Consequently, any potential attacker is prevented from invoking the *initializer()* function in the logic contract's state, thereby preventing any attempts to engage in malicious activities.

Path: ./contracts/Auction.sol : *

Recommendation: Include the *_disableInitializers()* call in the constructor in the *Auction.sol* contract.

Found in: 3c15738

Status: Fixed (Revised commit: f9000d7)

Remediation: The *_disableInitializers()* call was included in the constructor in the *Auction.sol* contract.

I07. Functions that can be declared external

To optimize Gas consumption, it is advisable to designate public functions in the contract as external in the following functions *totalDue(), dueOf(), supply(), setApprovalForAll()*. This practice helps minimize Gas costs.

Path: ./contracts/Auction.sol: totalDue(), dueOf(), supply(), ./contracts/PoolFactory.sol: setApprovalForAll()

Recommendation: Use the external attribute for functions never called from the contracts. The *setApprovalForAll* function was removed.

Status: Fixed (Revised commit: f9000d7)

Remediation: The aforementioned functions were declared as *external*.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Risk Level	High Impact	Medium Impact	Low Impact
High Likelihood	Critical	High	Medium
Medium Likelihood	High	Medium	Low
Low Likelihood	Medium	Low	Low

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

Risk Levels

Critical: Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.

High: High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.

Medium: Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.

Low: Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Impact Levels

High Impact: Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

Medium Impact: Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

Low Impact: Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

Likelihood Levels

High Likelihood: Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

Medium Likelihood: Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

Low Likelihood: Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope details

Repository	https://github.com/clearpool-finance/open-term-pools
Commit	3c15738
Requirements	Google Docs
Technical Requirements	Google Docs

Contracts in Scope

contracts/Auction.sol
contracts/BondNFT.sol
contracts/PoolFactory.sol
contracts/PoolMaster.sol
contracts/interfaces/IAuction.sol
contracts/interfaces/IBondNFT.sol
contracts/interfaces/IMulticall3.sol
contracts/interfaces/IPoolFactory.sol
contracts/interfaces/IPoolMaster.sol
contracts/kyc/QuadrataReader.sol
contracts/libraries/Decimal.sol
contracts/libraries/NFTDescriptor.sol

contracts/libraries/RewardAsset.sol contracts/utils/Utils.sol

Second scope details

Repository	https://github.com/clearpool-finance/open-term-pools
Commit	cd0aa788
Requiremens	Google Docs
Technical Requiremens	Google Docs

Contracts in Scope

contracts/Auction.sol contracts/BondNFT.sol contracts/PoolFactory.sol contracts/PoolMaster.sol contracts/interfaces/IAuction.sol contracts/interfaces/IBondNFT.sol contracts/interfaces/IMulticall3.sol contracts/interfaces/IPoolFactory.sol contracts/interfaces/IPoolMaster.sol contracts/kyc/QuadrataReader.sol contracts/libraries/Decimal.sol contracts/libraries/NFTDescriptor.sol contracts/libraries/RewardAsset.sol contracts/utils/Utils.sol