HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Clearpool.finance
Date: April 27, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Clearpool.finance
Approved By	Noah Jelich | Lead SC Auditor at Hacken OU
Type	Lending Platform
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://clearpool.finance/
Changelog	12.04.2023 - Initial Review 27.04.2023 - Second Review

Table of contents

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.

Scope

The scope of the project includes review and security analysis of the following smart contracts from the provided repository:

Initial review scope

Repository	https://github.com/clearpool-finance/prime-protocol
Commit	62f5e3e8b694e6c8dd33ef9ba21412d099f41aa9
Whitepaper	
Functional Requirements	NatSpec
Technical Requirements	NatSpec
Contracts	File: ./protocol/contracts/Pool/IERC20Lite.sol SHA3: 41d00b31d349e5c803eb96bf76b193d648e01e8e654e924b89fc58093f1b08b2 File: ./protocol/contracts/Pool/IPool.sol SHA3: ff04607e2221e280aae74001ed4d425aa01a9696f5fa0ff4d36b3c8ad2535540 File: ./protocol/contracts/Pool/IPoolFactory.sol SHA3: d171d5e33fc3e9de4ea45809a0650254973e2d9db0cc077b675ff6d255365a50 File: ./protocol/contracts/Pool/Pool.sol SHA3: 0a3628256824a144a0cf462823c49a26236b595439c31d4f5ddae4b9bb97a7cd File: ./protocol/contracts/Pool/PoolFactory.sol SHA3: e1250da03d0e6e6be686006a40896a509ddbeb6780a07614cbb0e9a8d2a54015 File: ./protocol/contracts/PrimeMembership/Asset.sol SHA3: 0381d0c8edf124ea27d713c881451168d594f87a0be10e5f5f3dc11372d74329 File: ./protocol/contracts/PrimeMembership/IPrime.sol SHA3: 1b7c74af459ff4a8049fcfedcbb1d371263e9379978fa47e313192fd56da0ca2 File: ./protocol/contracts/PrimeMembership/Prime.sol SHA3: d7e393dec1e4766f0c92548bad4453adf86f70faa8e0b11f164ed8e812efe654 File: ./protocol/contracts/utils/AddressCoder.sol SHA3: d6dac49d167e22934f85fad19ff2f4724c5b872342dd13074d6595e9e5337872 File: ./protocol/contracts/utils/Counters.sol SHA3: d0208474e51fb67555ce15cb83753c133f7d8afd87c8521ea96040dd42385aae File: ./protocol/contracts/utils/NZAGuard.sol

	SHA3: f9292ae80c8f8ec4e2e90a68c4d1976d2517a8e244bcfe183813da1ded191b22

Second review scope

Repository	https://github.com/clearpool-finance/prime-protocol
Commit	bfdc7947258e3d6200b33e4ed1d80c9d2cd77b05
Whitepaper	
Functional Requirements	NatSpec
Technical Requirements	NatSpec
Contracts	File: Pool/IPool.sol SHA3: d39adf1603dc4fd75412af034560299317511ccf3be8305a5736d6edca9ed4a3 File: Pool/IPoolFactory.sol SHA3: d171d5e33fc3e9de4ea45809a0650254973e2d9db0cc077b675ff6d255365a50 File: Pool/Pool.sol SHA3: 485a33b74abb416f6693ad4409a12cb2f3c8ad69dd73db42cb952123d17e88e5 File: Pool/PoolFactory.sol SHA3: a09a2a5c5dc98bea26cbd14e8eaa15f6a8301f267c528ad22bad79a39dc03e9a File: PrimeMembership/Asset.sol SHA3: 0381d0c8edf124ea27d713c881451168d594f87a0be10e5f5f3dc11372d74329 File: PrimeMembership/IPrime.sol SHA3: 3e9d3cd6d60f9520124cf599884c4ea2f22adf10d81c438f941ef111356d3272 File: PrimeMembership/Prime.sol SHA3: 1bfe6317fcfddea35608368423461700568c9a68a124818523ac4b3723b5edd1 File: utils/AddressCoder.sol SHA3: c7e99dde24338f80238a07b22664c430877b53543a680c024a795daff379ff9d File: utils/NZAGuard.sol SHA3: f9292ae80c8f8ec4e2e90a68c4d1976d2517a8e244bcfe183813da1ded191b22

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation by external or internal actors.
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 by external or internal actors.
Medium	Medium vulnerabilities are usually limited to state manipulations but cannot lead to asset loss. Major deviations from best practices are also in this category.
Low	Low vulnerabilities are related to outdated and unused code or minor Gas optimization. These issues won't have a significant impact on code execution but affect code quality

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 **10** out of **10**.

- Technical description is provided.
    - Project overview is detailed.
    - Use cases are described and detailed.
    - NatSpec is sufficient.
    - Run instructions are not provided.
- Functional requirements:
    - Overall system requirements are provided.

Code quality

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

- The development environment is configured.
- Code follows the Solidity style guide.

Test coverage

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

- Deployment and basic user interactions are covered with tests.
- Negative cases coverage is present.
- Interactions with several users are tested.

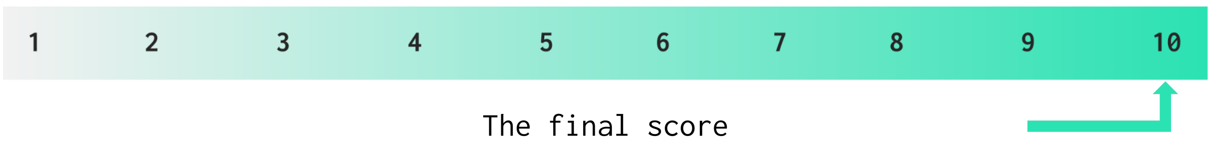
Security score

As a result of the audit, the code contains no issues. The security score is **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: **10**.The system users should acknowledge all the risks summed up in the risks section of the report.



The final score

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
12 April 2023	5	2	4	1
27 April 2023	0	0	0	0

Risks

- Contracts are upgradeable and are subject to future modification.
- Any prime member, regardless of their risk score, is allowed to create pools.
- Borrowers that fail to pay principal and/or interest by the agreed due date will be provided 72 hours to settle all outstanding payments with the lenders. If a lender changes the loan classification from “Overdue” to “Default”, all loans in the pool will be classified as such. Upon classification of a loan to “Default”, interest will stop accruing, and the legal process begins.
  Once the default process has been triggered, the pool can be removed from the protocol and the lenders, as legal creditors, may pursue the borrower according to the terms of the lending agreement.
  The above procedures apply to loans with both bullet repayments and monthly repayments.
- The **security of the funds in the system depends on the out-of-scope legal process**.

System Overview

Clearpool Prime is an EVM-compatible protocol that matches peer-to-peer orders for its whitelisted participants. The participants, suppliers and borrowers of an asset, interact directly with the protocol's smart contracts, earning (and paying) fixed interest rates without the requirement for collateral, custody or intermediary. All activities within the protocol are transparent and publicly inspectable.
Clearpool Prime enables vetted and verified institutions to borrow USDC (or other ERC-20 assets) from one or more pools of lenders, selected by the borrower, for a fixed time period and interest rate.
The files in the scope:

- **Prime.sol** - a contract for control of the Clearpool Prime membership database.
- **IPrime.sol** - interface of the Prime membership contract.
- **PoolFactory.sol** - a contract responsible for creating new pools.
- **IPoolFactory.sol** - interface of the PoolFactory contract.
- **Pool.sol** - is the settlement venue for borrowing in the Clearpool Prime protocol.
- **IPool.sol** - interface of the Pool contract.
- **Asset.sol** - a library which provides a simple utility for managing a collection of unique asset addresses.
- **AddressCoder.sol** - a library which contains utility functions for encoding and decoding arrays of addresses into bytes and vice versa.
- **NZAGuard.sol** - contains modifiers to check inputs for a non-zero address, non-zero value, non-same address, non-same value, and none-more-than-one.

Privileged roles

- Prime:
    - owner: can whitelist, blacklist members, update member risk score, can change the spread rate, the origination fee rate, the rolling increment fee rate, penalty rate per year and update the treasury address.
    - onlyPrime: The role of onlyPrime in the Prime contract is to grant the authority to engage in token lending, request and cancel callbacks, and accept rollover requests from Borrowers.
- PoolFactory:
    - owner: can mark the pools as defaulted, can change the prime contract address and the pool beacon address.
    - prime member: can create new pools.
- Pool:
    - borrower: can whitelist, blacklist lenders, change pool from private to public and vice versa, can repay lenders with the

principal and interest, can request the roll, close pools and mark pools as defaulted.

- prime lender: can lend funds to the pool and accept the roll, can create or cancel the callback.

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Passed
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	Passed

Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Not Relevant
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155 EIP-712	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of Unused Variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP Standards Violation	EIP	EIP standards should not be violated.	Not Relevant
Assets Integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed
User Balances Manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed

Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Passed
Token Supply Manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer.	Not Relevant
Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed
Style Guide Violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, which may be changed in the future.	Passed

Findings

■■■■ Critical

C01. Non-Finalized Code

In the Clearpool Prime membership contract, there is a function called whitelistMemberTestnet. This function allows any user to add any address as a whitelisted prime member. As a result, unauthorized users can potentially add addresses to the prime member list without proper validation or permission, compromising the integrity of the membership system and potentially allowing unauthorized access to exclusive prime member features.

The production code should not contain any functions or variables that are used solely in the test environment. This will allow malicious parties to manipulate the code or users to trigger them accidentally.

Path: ./protocol/contracts/PrimeMembership/Prime.sol : whitelistMemberTestnet(), isMember()

Recommendation: remove the testing functionality from the production code.

Found in: 62f5e3e

Status: Fixed (Revised commit: bfdc794)

■■■ High

H01. Undocumented Behavior; Variable is not limited

In the current implementation, the Prime contract owner can update the penalty rate at any time for all pools without any limitations. This unrestricted control over penalty rate adjustments could lead to unexpectedly high penalties for pools, potentially causing issues for borrowers and undermining trust in the system.

Path: ./protocol/contracts/PrimeMembership/Prime.sol : updatePenaltyRatePerYear()

Recommendation: add a penaltyRate variable to the pool contract and set the penalty rate during pool creation. Limit max penaltyRate in the Prime contract.

Found in: 62f5e3e

Status: Fixed (Revised commit: bfdc794)

H02. Data Consistency

The current implementation of the blacklisting functionality in the lending pool smart contract has inconsistencies.

Specifically, when a lender is blacklisted in a public pool, they can still lend assets due to the absence of a check for blacklisted addresses in public pools.

This allows blacklisted lenders to bypass the intended access control mechanism.

Path: ./protocol/contracts/Pool/Pool.sol : blacklistLenders()

Recommendation: allow blacklisting of lenders only if the pool is private or add the possibility to blacklist specified lenders in public pools.

Found in: 62f5e3e

Status: Fixed (Revised commit: bfdc794)

H03. Requirements Violation

The documentation states that the interest calculation in the lending pool smart contract will be performed on a per-block basis.

However, the current implementation calculates interest using timestamps.

This discrepancy between the intended behavior and the actual implementation could lead to unexpected outcomes.

Path: ./protocol/contracts/Pool/Pool.sol

Recommendation: update the documentation to accurately reflect the current implementation. If the requirement to calculate interest on a block basis is essential, modify the smart contract implementation to use block numbers instead of timestamps.

Found in: 62f5e3e

Status: Fixed (Revised commit: bfdc794)

H04. Non-Finalized Code

The code contains a comment stating "TODO: implement check for riskScore" but this check has not been implemented. The code should not contain TODO comments. Otherwise, it means that the code is not finalized and additional changes will be introduced in the future.

Path: ./protocol/contracts/PrimeMembership/Prime.sol : isMember()

Recommendation: remove the TODO comments.

Found in: 62f5e3e

Status: Fixed (Revised commit: bfdc794)

■■ Medium

M01. Uninitialized implementation

In the current Prime.sol and PoolFactory.sol contracts implementation, the base implementation contract is left uninitialized.

This could potentially lead to unintended usage or vulnerabilities if someone manages to interact with the uninitialized implementation contract directly.

To prevent this, it is recommended to lock the contract and disable its initializer functions upon deployment.

In the __Prime_init function the _unchained function of OwnableUpgradeable.sol is used directly. This is not recommended because it can cause initialization issues when upgrading or when inheriting from multiple upgradeable contracts.

If you call _unchained functions directly, you risk bypassing any additional code that may be added in future versions, which could lead to compatibility issues and unexpected behavior.

Path: ./protocol/contracts/PrimeMembership/Prime.sol

./protocol/contracts/Pool/PoolFactory.sol

Recommendation: invoke the _disableInitializers function in the constructor of the implementation contract. This will automatically lock the contract and disable its initializer functions when it is deployed, ensuring that the contract cannot be misused or compromised due to uninitialized state. Call the non-unchained initialization functions in the main initialization function.

Found in: 62f5e3e

Status: Fixed (Revised commit: bfdc794)

M02. Unchecked Transfer or Approve

The functions _lend, _repayTo and _repayInterestTo do not use the SafeERC20 library for checking the result of ERC20 token transfers. Tokens may not follow the ERC20 standard and return a false in case of transfer failure or not return any value at all.

Path: ./protocol/contracts/Pool/Pool.sol : _lend(), _repayTo(), _repayInterestTo()

Recommendation: use the SafeERC20 library to interact with tokens safely.

Found in: 62f5e3e

Status: Fixed (Revised commit: bfdc794)

■ Low

L01. Floating Pragma

Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Path: ./protocol/contracts/utils/Counters.sol

Recommendation: consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

Found in: 62f5e3e

Status: Fixed (Revised commit: bfdc794)

L02. Redundant Code Block

Solidity 0.8.0 and later versions have ABI coder v2 enabled by default, providing efficient and safe ABI encoding and decoding functionality.

There is no need to specify it explicitly.

Path: ./protocol/contracts/utils/AddressCoder.sol

Recommendation: remove redundant code parts.

Found in: 62f5e3e

Status: Fixed (Revised commit: bfdc794)

L03. NatSpec comments contradiction

The NatSpec comments of the function __Prime_init declare that function as internal.

The NatSpec comments of the function changeSpreadRate declare that the parameter spreadRate_ as the new fee collector address.

The NatSpec comments of the library Counters state that it provides counters that can only be incremented, decremented or reset.

This can lead to the wrong assumptions about the code's purpose.

Path: ./protocol/contracts/PrimeMembership/Prime.sol

./protocol/contracts/utils/Counters.sol

Recommendation: fix the mismatch.

Found in: 62f5e3e

Status: Fixed (Revised commit: bfdc794)

L04. Use of Redundant Modifier

The internal function _repayInterest uses modifiers onlyBorrower nonDefaulted. This function can be called only internally by the external function repayInterest.

The use of unnecessary modifiers will increase Gas consumption of the code. Thus they should be removed from the code.

Path: ./protocol/contracts/Pool/Pool.sol

Recommendation: remove the redundant modifiers.

Found in: 62f5e3e

Status: Fixed (Revised commit: bfdc794)

L05. Redundant Import

The functionality of OwnableUpgradeable.sol is unused in the Pool contract.

Unused imports should be removed from the contracts. Unused imports are allowed in Solidity and do not pose a direct security issue. However, it is best practice to avoid them as they can decrease readability.

Path: ./protocol/contracts/Pool/Pool.sol

Recommendation: remove the redundant import.

Found in: 62f5e3e

Status: Fixed (Revised commit: bfdc794)

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.