



BUIDL, Contract Code Review and Security Analysis Report

Customer: BUIDL
Prepared on: 14th March 2024
Platform: Base Chain
Language: Solidity

rdauditors.com

Table of Contents

Disclaimer	2
Document	3
Introduction	5
Project Scope	6
Executive Summary	7
Code Quality	7
Documentation	9
Use of Dependencies	9
AS-IS Overview	10
UML Diagram - PoolFactory	20
Inheritance Diagram - PoolFactory	21
UML Diagram - CorePool	23
Inheritance Diagram - CorePool	24
UML Diagram - PoolBase	26
Inheritance Diagram - PoolBase	27
Slither Results Log	28
Audit Findings	44
Conclusion	46
Note For Contract Users	46
Our Methodology	48
Disclaimers	50

Disclaimer

This document may contain confidential information about its systems and intellectual property of the customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the customer or it can be disclosed publicly after all vulnerabilities are fixed - upon the decision of the customer.

Document

Name	Smart Contract Code Review and Security Analysis Report of BUIDL
Platform	Base Chain / Solidity
File 1	PoolFactory.sol
File 1 Initial Code Link	5545ae4f5ecf1535e34f169b87eec4ea
MD5 hash	61B4BF6CB3E3033DE0CD5154C95081C3
SHA256 hash	9af88918ab4bf1edd0f83ea6344d467bf79620f3b8d5bf79951c976da0b5710c
File 2	CorePool.sol
File 2 Initial Code Link	d1f7f19e25d7cc08fcdd411933265162
MD5 hash	46FAEE64482C283221FF90314FBDEA14
SHA256 hash	9e516da5c0522881b22e3d561b777df7f84f6251fa936b52ff75697b5cbbad27
File 3	PoolBase.sol

File 3 Initial Code Link	e65359e2b3efb191090ec8575cc34519
MD5 hash	564D3794D503A0A3A6E0001C7FEED78
SHA256 hash	794a6cf1cb160c828754e43f401d849b844654eb0c9729434381df3ed75f923d
File 4	Interfaces.sol
File 4 Initial Code Link	8f17db91c95e3a13b211fbb86a6e9a84
MD5 hash	981DACB4C0F4CE76F9DE788A27CD3196
SHA256 hash	d2831bec9818fc6dfb328d2475d710d45fa8f90b67ded4e04ad414c0ff2ca3f5
Date	14/03/2024

Introduction

RD Auditors (Consultant) were contracted by BUIDL (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report represents the findings of the security assessment of the customer's smart contract and its code review conducted between 4th - 14th March 2024.

This contract consists of one interface and three smart contracts files.

Project Scope



The scope of the project is a smart contract. We have scanned this smart contract for commonly known and more specific vulnerabilities, below are those considered (the full list includes but is not limited to):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Byte array vulnerabilities
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level

Executive Summary






According to the assessment, the customer's solidity smart contract is now **Secured**.

You are Here

 **Insecure**  **Poorly Secured**  **Secure**  **Well-Secured**

Automated checks are with smartDec, Mythril, Slither and remix IDE. All issues were performed by our team, which included the analysis of code functionality, the manual audit found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the audit overview section. The general overview is presented in the AS-IS section and all issues found are located in the audit overview section.

We found the following;

Total Issues	0
 Critical	0
 High	0
 Medium	0
 Low	1
 Very Low	1

Code Quality

The libraries within this smart contract are part of a logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned to a specific address and its properties/methods can be reused many times by other contracts.

The BUIDL team has not provided any scenario and unit test scripts, which would help to determine the integrity of the code in an automated way.

Overall, the code is almost commented. Commenting can provide rich documentation for functions, return variables and more. Use of the Ethereum Natural Language Specification Format (NatSpec) for commenting is recommended.

Documentation

We were given the BUIDL code as a gist.github link:

PoolFactory.sol : [5545ae4f5ecf1535e34f169b87eec4ea](#)

CorePool.sol : [d1f7f19e25d7cc08fcdd411933265162](#)

PoolBase.sol : [e65359e2b3efb191090ec8575cc34519](#)

Interfaces.sol : [8f17db91c95e3a13b211fbb86a6e9a84](#)

The hash of that file is mentioned in the table. As mentioned above, it's commented on smart contract code, so anyone can quickly understand the programming flow as well as complex code logic.

Comments are very helpful in understanding the overall architecture of the protocol. It also provides a clear overview of the system components, including helpful details, like the lifetime of the background script.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure. Those were based on well known industry standard open source projects and even core code blocks that are written well and systematically.

AS-IS Overview

PoolFactory.sol

File And Function Level Report

Contract: PoolFactory
Import: IERC20, Ownable, IPool, IPoolFactory, CorePool
Inherit: Ownable
Observation: Passed
Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	getPoolAddress	external	Passed	All Passed	No Issue	Passed
2	getPoolData	write	Passed	All Passed	No Issue	Passed
3	shouldUpdateRatio	read	Passed	All Passed	No Issue	Passed
4	createPool	external	onlyOwner	All Passed	No Issue	Passed
5	registerPool	write	onlyOwner	All Passed	No Issue	Passed
6	removePool	write	Critical operation lacks event log, Wrong comment	All Passed	Refer Audit Findings	Passed
7	updateBUIDLPerBlock	external	Passed	All Passed	No Issue	Passed
8	transferYieldToken	external	Passed	All Passed	No Issue	Passed
9	changePoolWeight	external	onlyOwner	All Passed	No Issue	Passed

10	setBurnFee	external	onlyOwner	All Passed	No Issue	Passed
11	setRewardLoc kLimit	external	onlyOwner	All Passed	No Issue	Passed
12	setAddresses	external	onlyOwner	All Passed	No Issue	Passed
13	setGeneralInf o	external	onlyOwner	All Passed	No Issue	Passed
14	addPoolAddre ss	external	onlyOwner	All Passed	No Issue	Passed
15	removePoolA ddress	external	onlyOwner	All Passed	No Issue	Passed
16	emergencyWi thdraw	external	onlyOwner	All Passed	No Issue	Passed
17	onlyOwner	modifier	Passed	All Passed	No Issue	Passed
18	owner	read	Passed	All Passed	No Issue	Passed
19	_checkOwner	internal	Passed	All Passed	No Issue	Passed
20	renounceOwn ership	write	onlyOwner	All Passed	No Issue	Passed
21	transferOwner ship	write	onlyOwner	All Passed	No Issue	Passed
22	_transferOwn ership	internal	Passed	All Passed	No Issue	Passed

PoolBase.sol

File And Function Level Report

Contract: PoolBase
Import: SafeERC20, ReentrancyGuard, IPool, ICorePool,
IPoolFactory, IStarterInfo
Inherit: IPool, ReentrancyGuard
Observation: Passed
Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	onlyStarterDe vOrFactory	modifier	Passed	All Passed	No Issue	Passed
2	constructor	write	Passed	All Passed	No Issue	Passed
3	initConfig	internal	Passed	All Passed	No Issue	Passed
4	pendingYield Rewards	external	Passed	All Passed	No Issue	Passed
5	balanceOf	external	Passed	All Passed	No Issue	Passed
6	getDeposit	external	Passed	All Passed	No Issue	Passed
7	getDepositsLe ngth	external	Passed	All Passed	No Issue	Passed
8	stake	external	Passed	All Passed	No Issue	Passed
9	stakeFor	external	Passed	All Passed	No Issue	Passed
10	unstake	external	Passed	All Passed	No Issue	Passed

11	updateStakeLock	external	Passed	All Passed	No Issue	Passed
12	sync	external	Passed	All Passed	No Issue	Passed
13	processRewards	external	Passed	All Passed	No Issue	Passed
14	setWeight	external	Passed	All Passed	No Issue	Passed
15	_pendingYieldRewards	internal	Passed	All Passed	No Issue	Passed
16	_stake	internal	Passed	All Passed	No Issue	Passed
17	_unstake	internal	Passed	All Passed	No Issue	Passed
18	_sync	internal	Passed	All Passed	No Issue	Passed
19	_processRewards	internal	Passed	All Passed	No Issue	Passed
20	_updateStakeLock	internal	Passed	All Passed	No Issue	Passed
21	weightToReward	read	Passed	All Passed	No Issue	Passed
22	rewardToWeight	read	Passed	All Passed	No Issue	Passed
23	transferPoolToken	internal	Passed	All Passed	No Issue	Passed
24	transferPoolTokenFrom	internal	Passed	All Passed	No Issue	Passed
25	getHistoryLength	external	Passed	All Passed	No Issue	Passed
26	getTokenBurnFee	read	Passed	All Passed	No Issue	Passed

27	setStakingConfig	external	onlyStarterDevOrFactory	All Passed	No Issue	Passed
28	isLongStaker	external	Passed	All Passed	No Issue	Passed
29	updateLastInvestTimestamp	external	Passed	All Passed	No Issue	Passed
30	clearHistory	external	Passed	All Passed	No Issue	Passed
31	setConfiguration	external	Passed	All Passed	No Issue	Passed
32	setInitialSettings	external	Passed	All Passed	No Issue	Passed
33	getRewardLockPeriod	read	Passed	All Passed	No Issue	Passed
34	addPresaleAddress	external	onlyStarterDevOrFactory	All Passed	No Issue	Passed
35	nonReentrant	modifier	Passed	All Passed	No Issue	Passed
36	_nonReentrantBefore	write	Passed	All Passed	No Issue	Passed
37	_nonReentrantAfter	write	Passed	All Passed	No Issue	Passed
38	_reentrancyGuardEntered	internal	Passed	All Passed	No Issue	Passed

CorePool.sol

File And Function Level Report

Contract: CorePool
Import: Initializable, PoolBase
Inherit: PoolBase, Initializable
Observation: Passed
Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	constructor	write	Passed	All Passed	No Issue	Passed
2	initialize	write	initializer	All Passed	No Issue	Passed
3	pendingVault Rewards	read	Passed	All Passed	No Issue	Passed
4	setVault	external	Passed	All Passed	No Issue	Passed
5	receiveVault Rewards	external	Passed	All Passed	No Issue	Passed
6	processRewards	external	Passed	All Passed	No Issue	Passed
7	stakeAsPool	external	Passed	All Passed	No Issue	Passed
8	_stake	internal	Passed	All Passed	No Issue	Passed
9	_unstake	internal	Passed	All Passed	No Issue	Passed
10	_processRewards	internal	Passed	All Passed	No Issue	Passed
11	_processVault Rewards	write	Passed	All Passed	No Issue	Passed
12	transferBuidl	internal	Passed	All Passed	No Issue	Passed

13	transferBuidlFrom	internal	nonReentrant	All Passed	No Issue	Passed
14	initializer	modifier	Passed	All Passed	No Issue	Passed
15	reinitializer	modifier	Passed	All Passed	No Issue	Passed
16	onlyInitializing	modifier	Passed	All Passed	No Issue	Passed
17	_checkInitializing	internal	Passed	All Passed	No Issue	Passed
18	_disableInitializers	internal	Passed	All Passed	No Issue	Passed
19	_getInitializedVersion	internal	Passed	All Passed	No Issue	Passed
20	_isInitializing	internal	Passed	All Passed	No Issue	Passed
21	_getInitializableStorage	write	Passed	All Passed	No Issue	Passed
22	onlyStarterDevOrFactory	modifier	Passed	All Passed	No Issue	Passed
23	constructor	write	Passed	All Passed	No Issue	Passed
24	initConfig	internal	Passed	All Passed	No Issue	Passed
25	pendingYieldRewards	external	Passed	All Passed	No Issue	Passed
26	balanceOf	external	Passed	All Passed	No Issue	Passed
27	getDeposit	external	Passed	All Passed	No Issue	Passed
28	getDepositsLength	external	Passed	All Passed	No Issue	Passed
29	stake	external	Passed	All Passed	No Issue	Passed
30	stakeFor	external	Passed	All Passed	No Issue	Passed
31	unstake	external	Passed	All Passed	No Issue	Passed

32	updateStakeLock	external	Passed	All Passed	No Issue	Passed
33	sync	external	Passed	All Passed	No Issue	Passed
34	processRewards	external	Passed	All Passed	No Issue	Passed
35	setWeight	external	Passed	All Passed	No Issue	Passed
36	_pendingYieldRewards	internal	Passed	All Passed	No Issue	Passed
37	_stake	internal	Passed	All Passed	No Issue	Passed
38	_unstake	internal	Passed	All Passed	No Issue	Passed
39	_sync	internal	Passed	All Passed	No Issue	Passed
40	_processRewards	internal	Passed	All Passed	No Issue	Passed
41	_updateStakeLock	internal	Passed	All Passed	No Issue	Passed
42	weightToReward	read	Passed	All Passed	No Issue	Passed
43	rewardToWeight	read	Passed	All Passed	No Issue	Passed
44	transferPoolToken	internal	Passed	All Passed	No Issue	Passed
45	transferPoolTokenFrom	internal	Passed	All Passed	No Issue	Passed
46	getHistoryLength	external	Passed	All Passed	No Issue	Passed
47	getTokenBurnFee	read	Passed	All Passed	No Issue	Passed

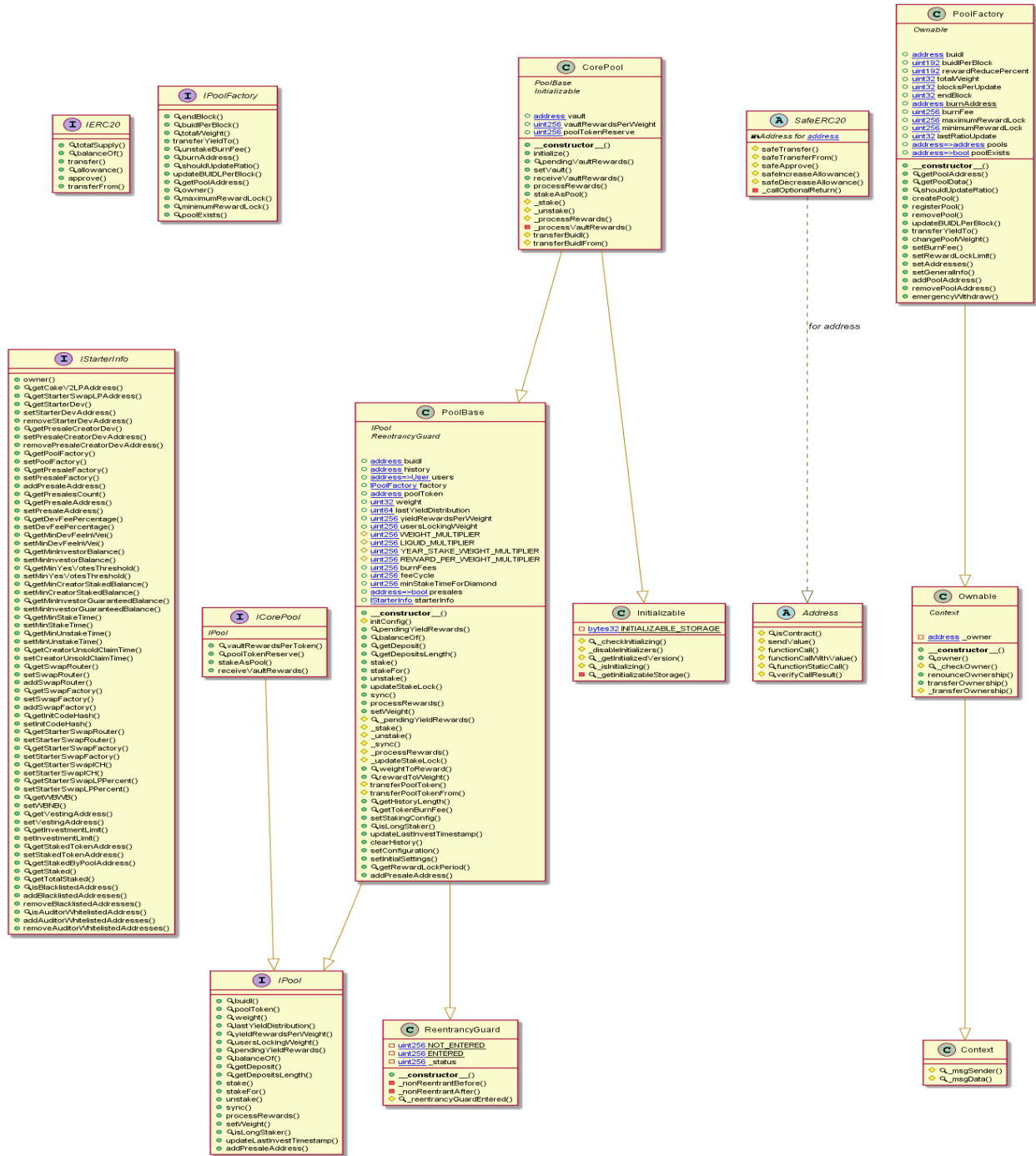
48	setStakingConfig	external	onlyStarterDevOrFactory	All Passed	No Issue	Passed
49	isLongStaker	external	Passed	All Passed	No Issue	Passed
50	updateLastInvestTimestamp	external	Passed	All Passed	No Issue	Passed
51	clearHistory	external	Passed	All Passed	No Issue	Passed
52	setConfiguration	external	Passed	All Passed	No Issue	Passed
53	setInitialSettings	external	Passed	All Passed	No Issue	Passed
54	getRewardLockPeriod	read	Passed	All Passed	No Issue	Passed
55	addPresaleAddress	external	onlyStarterDevOrFactory	All Passed	No Issue	Passed

Code Flow Diagram - PoolFactory

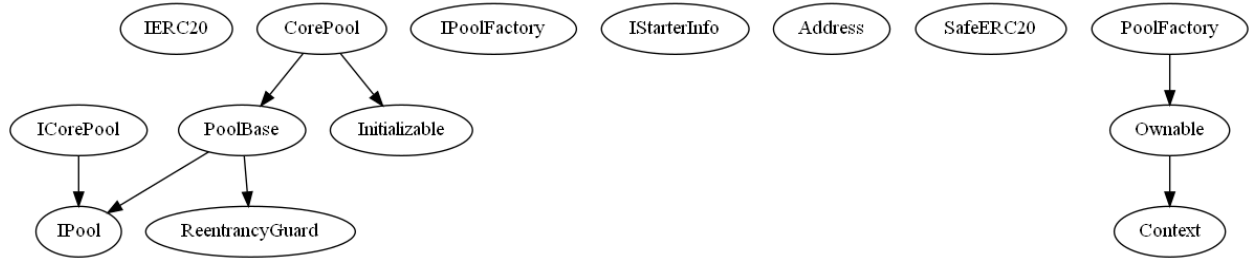
Please follow the below link to view the image:

https://drive.google.com/file/d/17rlktgejXVaiP5l22MNYVVbKfRU7Ocx1/view?usp=drive_link

UML Diagram - PoolFactory



Inheritance Diagram - PoolFactory

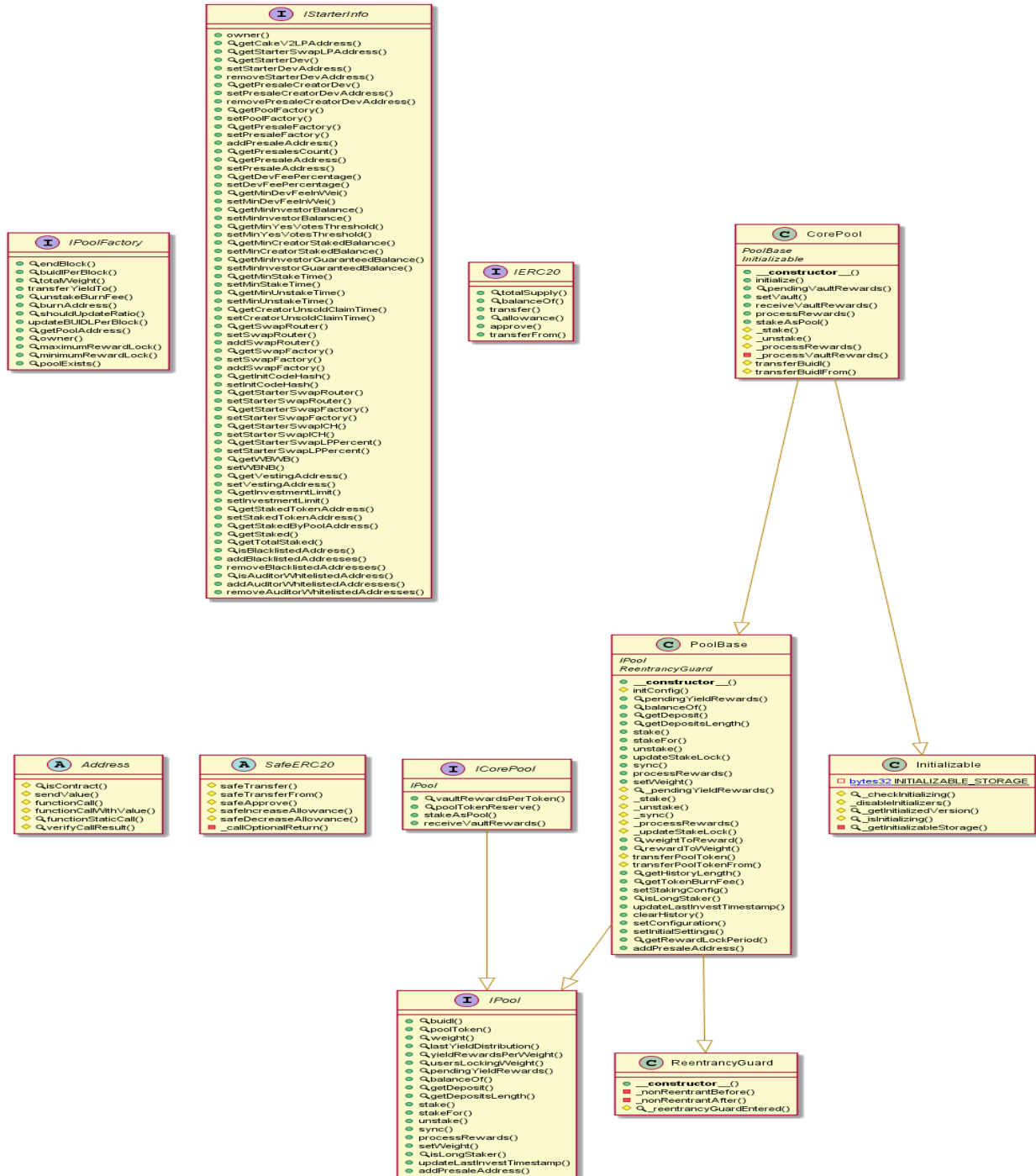


Code Flow Diagram - CorePool

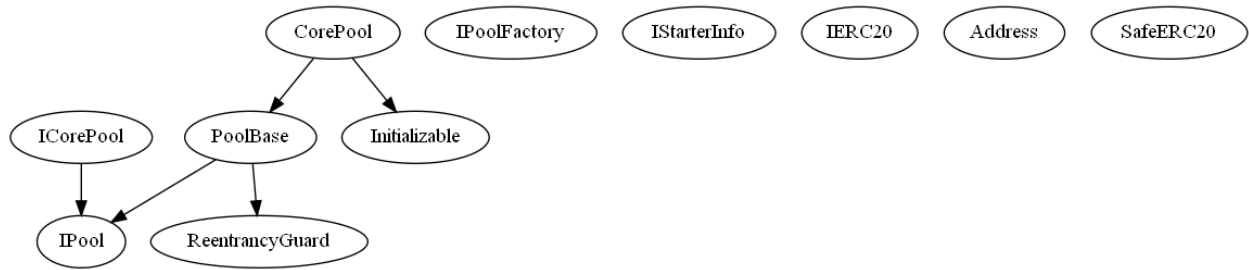
Please follow the below link to view the image:

https://drive.google.com/file/d/1swysRZhPV5QnaGCXMLjTF4EkOYZfZTOT/view?usp=drive_link

UML Diagram - CorePool



Inheritance Diagram - CorePool

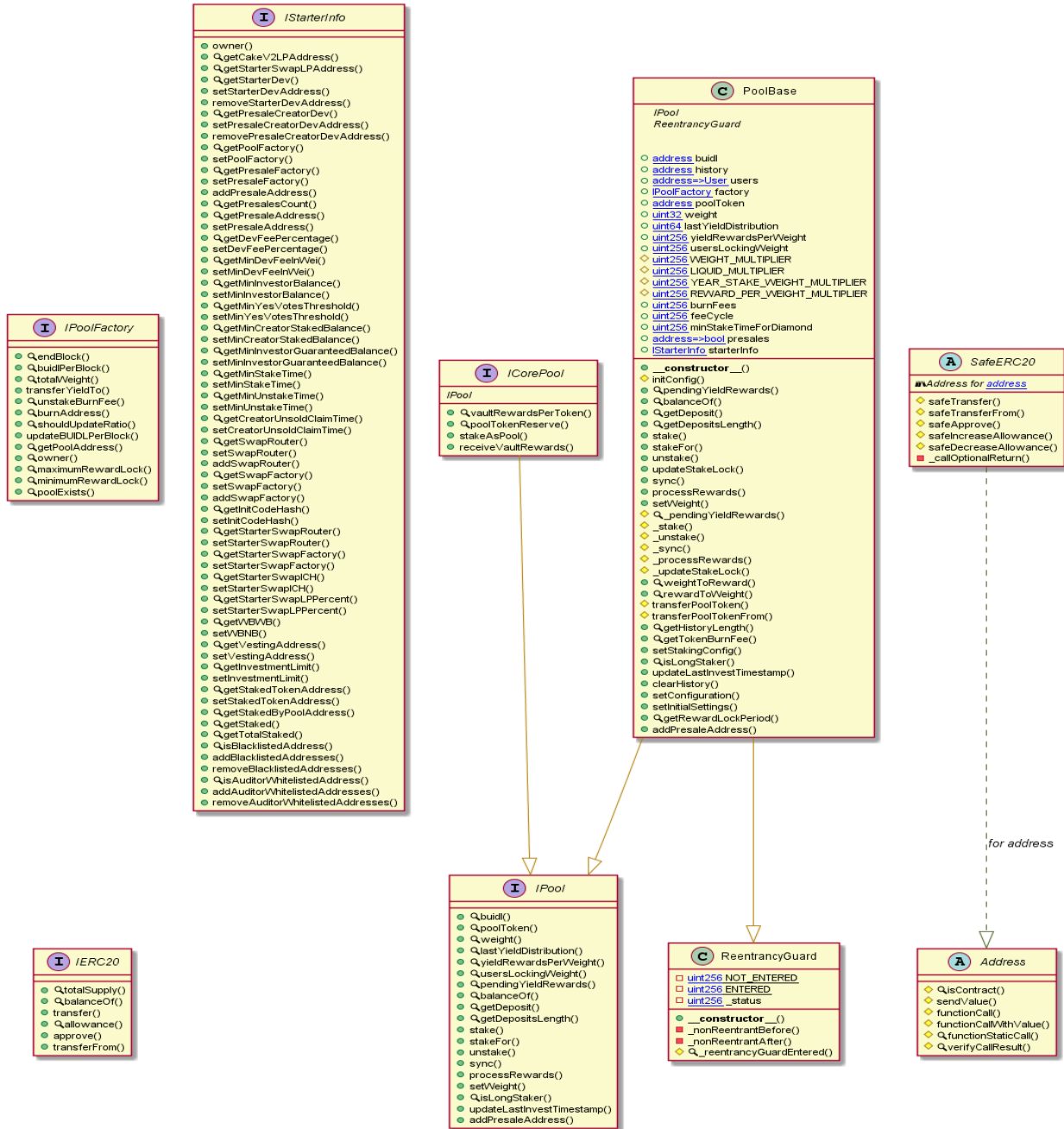


Code Flow Diagram - PoolBase

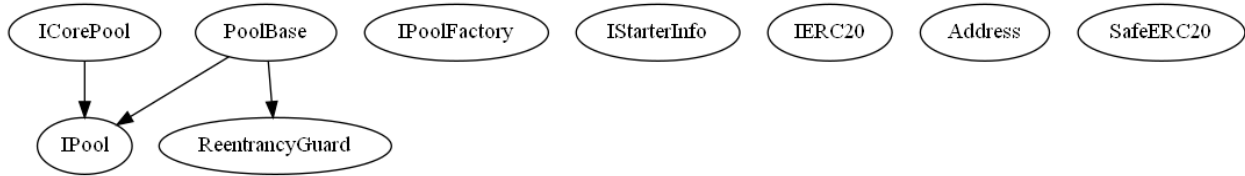
Please follow the below link to view the image:

https://drive.google.com/file/d/1RzRzwTQ45ldBMVJkJGBQwgSHrvbXAPvG/view?usp=drive_link

UML Diagram - PoolBase



Inheritance Diagram - PoolBase



Slither Results Log

PoolFactory.sol

```
INFO:Detectors:
PoolFactory.setBurnFee(uint256) (PoolFactory.sol#2584-2586) should emit an event for:
  - burnFee = _burnFee (PoolFactory.sol#2585)
PoolFactory.setGeneralInfo(address,uint192,uint192,uint32,uint32,uint32) (PoolFactory.sol#2601-2616) should emit an event for:
  - buildPerBlock = _buildPerBlock (PoolFactory.sol#2611)
  - rewardReducePercent = _rewardReducePercent (PoolFactory.sol#2612)
  - blocksPerUpdate = _blocksPerUpdate (PoolFactory.sol#2613)
  - lastRatioUpdate = _initBlock (PoolFactory.sol#2614)
  - endBlock = _endBlock (PoolFactory.sol#2615)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
INFO:Detectors:
PoolBase.setInitialSettings(address,address)._poolToken (PoolFactory.sol#1601) lacks a zero-check on :
  - poolToken = _poolToken (PoolFactory.sol#1604)
PoolFactory.setAddresses(address)._build (PoolFactory.sol#2597) lacks a zero-check on :
  - build = _build (PoolFactory.sol#2598)
PoolFactory.setGeneralInfo(address,uint192,uint192,uint32,uint32,uint32)._build (PoolFactory.sol#2602) lacks a zero-check on :
  - build = _build (PoolFactory.sol#2610)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in PoolBase._processRewards(address,bool) (PoolFactory.sol#1334-1397):
  External calls:
  - _sync() (PoolFactory.sol#1341)
    - factory.updateBUIDLPerBlock() (PoolFactory.sol#1292)
  State variables written after the call(s):
  - user.deposits.push(newDeposit) (PoolFactory.sol#1368)

r.totalWeight,getRewardLockPeriod(_staker)) (PoolFactory.sol#1379-1384)
  External calls sending eth:
  - _stake(_staker,_amount,_lockUntil,false,0) (PoolFactory.sol#965)
    - (success,returndata) = target.call{value: value}(data) (PoolFactory.sol#417)
  State variables written after the call(s):
  - history.push(_staker) (PoolFactory.sol#966)
Reentrancy in PoolBase.unstake(uint256,uint256) (PoolFactory.sol#977-981):
  External calls:
  - _unstake(msg.sender,_depositId,_amount) (PoolFactory.sol#979)
    - SafeERC20.safeTransfer(IERC20(poolToken),_to,_value) (PoolFactory.sol#1502)
    - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (PoolFactory.sol#536)
    - factory.updateBUIDLPerBlock() (PoolFactory.sol#1292)
    - (success,returndata) = target.call{value: value}(data) (PoolFactory.sol#417)
    - ICorePool(buildPool).stakeAsPool(_staker,pendingYield,(user.liquidWeight * 100) / user.totalWeight,getRewardLockPeriod(_staker)) (PoolFactory.sol#1379-1384)
    - factory.transferYieldTo(msg.sender,_amount,liquidRewardAmount) (PoolFactory.sol#1264)
  External calls sending eth:
  - _unstake(msg.sender,_depositId,_amount) (PoolFactory.sol#979)
    - (success,returndata) = target.call{value: value}(data) (PoolFactory.sol#417)
  State variables written after the call(s):
  - history.push(msg.sender) (PoolFactory.sol#980)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
```

```
INFO:Detectors:
Reentrancy in PoolBase._processRewards(address,bool) (PoolFactory.sol#1334-1397):
  External calls:
    - _sync() (PoolFactory.sol#1341)
      - factory.updateBUIDLPerBlock() (PoolFactory.sol#1292)
    - ICorePool(buidlPool).stakeAsPool(_staker,pendingYield,(user.liquidWeight * 100) / user.totalWeight,getRewardLockPeriod(_staker)) (PoolFactory.sol#1379-1384)
  Event emitted after the call(s):
    - YieldClaimed(msg.sender,_staker,pendingYield) (PoolFactory.sol#1396)
Reentrancy in CorePool._processRewards(address,bool) (PoolFactory.sol#2083-2095):
  External calls:
    - _processVaultRewards(_staker) (PoolFactory.sol#2088)
      - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (PoolFactory.sol#536)
      - SafeERC20.safeTransfer(IERC20(buidl),_to,_value) (PoolFactory.sol#2136)
      - (success,returndata) = target.call{value: value}(data) (PoolFactory.sol#417)
    - pendingYield = super._processRewards(_staker,_withUpdate) (PoolFactory.sol#2089)
      - factory.updateBUIDLPerBlock() (PoolFactory.sol#1292)
      - ICorePool(buidlPool).stakeAsPool(_staker,pendingYield,(user.liquidWeight * 100) / user.totalWeight,getRewardLockPeriod(_staker)) (PoolFactory.sol#1379-1384)
  External calls sending eth:
    - _processVaultRewards(_staker) (PoolFactory.sol#2088)
      - (success,returndata) = target.call{value: value}(data) (PoolFactory.sol#417)
  Event emitted after the call(s):
    - Synchronized(msg.sender,yieldRewardsPerWeight,lastYieldDistribution) (PoolFactory.sol#1324)
      - pendingYield = super._processRewards(_staker,_withUpdate) (PoolFactory.sol#2089)
    - YieldClaimed(msg.sender,_staker,pendingYield) (PoolFactory.sol#1396)
      - pendingYield = super._processRewards(_staker,_withUpdate) (PoolFactory.sol#2089)
```

```
INFO:Detectors:
Reentrancy in PoolBase._processRewards(address,bool) (PoolFactory.sol#1334-1397):
  External calls:
    - _sync() (PoolFactory.sol#1341)
      - factory.updateBUIDLPerBlock() (PoolFactory.sol#1292)
    - ICorePool(buidlPool).stakeAsPool(_staker,pendingYield,(user.liquidWeight * 100) / user.totalWeight,getRewardLockPeriod(_staker)) (PoolFactory.sol#1379-1384)
  Event emitted after the call(s):
    - YieldClaimed(msg.sender,_staker,pendingYield) (PoolFactory.sol#1396)
Reentrancy in CorePool._processRewards(address,bool) (PoolFactory.sol#2083-2095):
  External calls:
    - _processVaultRewards(_staker) (PoolFactory.sol#2088)
      - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (PoolFactory.sol#536)
      - SafeERC20.safeTransfer(IERC20(buidl),_to,_value) (PoolFactory.sol#2136)
      - (success,returndata) = target.call{value: value}(data) (PoolFactory.sol#417)
    - pendingYield = super._processRewards(_staker,_withUpdate) (PoolFactory.sol#2089)
      - factory.updateBUIDLPerBlock() (PoolFactory.sol#1292)
      - ICorePool(buidlPool).stakeAsPool(_staker,pendingYield,(user.liquidWeight * 100) / user.totalWeight,getRewardLockPeriod(_staker)) (PoolFactory.sol#1379-1384)
  External calls sending eth:
    - _processVaultRewards(_staker) (PoolFactory.sol#2088)
      - (success,returndata) = target.call{value: value}(data) (PoolFactory.sol#417)
  Event emitted after the call(s):
    - Synchronized(msg.sender,yieldRewardsPerWeight,lastYieldDistribution) (PoolFactory.sol#1324)
      - pendingYield = super._processRewards(_staker,_withUpdate) (PoolFactory.sol#2089)
    - YieldClaimed(msg.sender,_staker,pendingYield) (PoolFactory.sol#1396)
      - pendingYield = super._processRewards(_staker,_withUpdate) (PoolFactory.sol#2089)
```

```
PoolBase.getTokenBurnFee(address) (PoolFactory.sol#1525-1536) uses timestamp for comparisons
  Dangerous comparisons:
  - (block.timestamp < user.lastUnstakedTimestamp + feeCycle[i]) || block.timestamp < user.lastIn
  vestTimestamp + feeCycle[i] (PoolFactory.sol#1529-1530)
PoolBase.isLongStaker(address) (PoolFactory.sol#1551-1557) uses timestamp for comparisons
  Dangerous comparisons:
  - user.tokenAmount > 0 && user.firstStakedTimestamp > 0 && user.firstStakedTimestamp + minStake
  TimeForDiamond < block.timestamp (PoolFactory.sol#1553-1556)
CorePool.receiveVaultRewards(uint256) (PoolFactory.sol#1941-1959) uses timestamp for comparisons
  Dangerous comparisons:
  - require(bool,string)(usersLockingWeight != 0,-4) (PoolFactory.sol#1947)
CorePool._unstake(address,uint256,uint256) (PoolFactory.sol#2057-2075) uses timestamp for comparisons
  Dangerous comparisons:
  - require(bool,string)(stakeDeposit.lockedFrom == 0 || block.timestamp > stakeDeposit.lockedUnt
  il,-6) (PoolFactory.sol#2064-2068)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.verifyCallResult(bool,bytes,string) (PoolFactory.sol#439-457) uses assembly
  - INLINE ASM (PoolFactory.sol#449-452)
Initializable._getInitializableStorage() (PoolFactory.sol#1804-1808) uses assembly
  - INLINE ASM (PoolFactory.sol#1805-1807)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Address.functionCall(address,bytes) (PoolFactory.sol#388-390) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (PoolFactory.sol#400-406) is never used and should
  be removed
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
PoolBase.YEAR_STAKE_WEIGHT_MULTIPLIER (PoolFactory.sol#671) is set pre-construction with a non-constant
  function or state variable:
  - 2 * WEIGHT_MULTIPLIER
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#function-initializing-state
INFO:Detectors:
Pragma version0.8.13 (PoolFactory.sol#2) allows old versions
solc-0.8.13 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (PoolFactory.sol#381-386):
  - (success) = recipient.call{value: amount}{} (PoolFactory.sol#384)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (PoolFactory.sol#408-419)
  :
  - (success, returndata) = target.call{value: value}(data) (PoolFactory.sol#417)
Low level call in Address.functionStaticCall(address,bytes,string) (PoolFactory.sol#425-434):
  - (success, returndata) = target.staticcall(data) (PoolFactory.sol#432)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter PoolBase.initConfig(address,IPoolFactory,address,uint64,uint32)._buidl (PoolFactory.sol#828)
  is not in mixedCase
Parameter PoolBase.initConfig(address,IPoolFactory,address,uint64,uint32)._factory (PoolFactory.sol#829
  ) is not in mixedCase
Parameter PoolBase.initConfig(address,IPoolFactory,address,uint64,uint32)._poolToken (PoolFactory.sol#8
  30) is not in mixedCase
Parameter PoolBase.initConfig(address,IPoolFactory,address,uint64,uint32)._initBlock (PoolFactory.sol#8
  31) is not in mixedCase
```

```

Parameter PoolFactory.setGeneralInfo(address,uint192,uint192,uint32,uint32,uint32)._buidlPerBlock (PoolFactory.sol#2603) is not in mixedCase
Parameter PoolFactory.setGeneralInfo(address,uint192,uint192,uint32,uint32,uint32)._rewardReducePercent (PoolFactory.sol#2604) is not in mixedCase
Parameter PoolFactory.setGeneralInfo(address,uint192,uint192,uint32,uint32,uint32)._blocksPerUpdate (PoolFactory.sol#2605) is not in mixedCase
Parameter PoolFactory.setGeneralInfo(address,uint192,uint192,uint32,uint32,uint32)._initBlock (PoolFactory.sol#2606) is not in mixedCase
Parameter PoolFactory.setGeneralInfo(address,uint192,uint192,uint32,uint32,uint32)._endBlock (PoolFactory.sol#2607) is not in mixedCase
Parameter PoolFactory.addPoolAddress(address)._pool (PoolFactory.sol#2618) is not in mixedCase
Parameter PoolFactory.removePoolAddress(address)._pool (PoolFactory.sol#2622) is not in mixedCase
Parameter PoolFactory.emergencyWithdraw(address)._to (PoolFactory.sol#2626) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (PoolFactory.sol#2159)" inContext (PoolFactory.sol#2153-2162)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
Variable PoolBase.LIQUID_MULTIPLIER (PoolFactory.sol#665) is too similar to PoolBase.setConfiguration(uint256,uint256,uint256,uint256)._liquidMultiplier (PoolFactory.sol#1579)
Variable PoolBase.WEIGHT_MULTIPLIER (PoolFactory.sol#661) is too similar to PoolBase.setConfiguration(uint256,uint256,uint256,uint256)._weightMultiplier (PoolFactory.sol#1578)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar
INFO:Detectors:
Loop condition i < feeCycle.length (PoolFactory.sol#1527) should use cached array length instead of referencing `length` member of the storage array.
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#cache-array-length
INFO:Slither:PoolFactory.sol analyzed (14 contracts with 93 detectors), 155 result(s) found

```

CorePool.sol

```

INFO:Detectors:
PoolBase.setInitialSettings(address,address)._poolToken (CorePool.sol#1601) lacks a zero-check on :
- poolToken = _poolToken (CorePool.sol#1604)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in PoolBase._processRewards(address,bool) (CorePool.sol#1334-1397):
  External calls:
  - _sync() (CorePool.sol#1341)
    - factory.updateBUIDLPerBlock() (CorePool.sol#1292)
  State variables written after the call(s):
  - user.deposits.push(newDeposit) (CorePool.sol#1368)
  - user.tokenAmount += pendingYield (CorePool.sol#1371)
  - user.totalWeight += depositWeight (CorePool.sol#1372)
Reentrancy in PoolBase._sync() (CorePool.sol#1289-1325):
  External calls:
  - factory.updateBUIDLPerBlock() (CorePool.sol#1292)
  State variables written after the call(s):
  - lastYieldDistribution = uint64(block.number) (CorePool.sol#1306)
  - lastYieldDistribution = uint64(currentBlock) (CorePool.sol#1321)
  - yieldRewardsPerWeight += rewardToWeight(buidlReward,usersLockingWeight) (CorePool.sol#1320)
Reentrancy in CorePool.receiveVaultRewards(uint256) (CorePool.sol#1940-1958):
  External calls:
  - transferBuidlFrom(msg.sender,address(this),_rewardsAmount) (CorePool.sol#1948)
  - SafeERC20.safeTransferFrom(IERC20(buidl),_from,_to,_value) (CorePool.sol#2149)
  - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (CorePool.sol#536)
  - (success,returndata) = target.call{value: value}(data) (CorePool.sol#417)

```



```
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (CorePool.sol#503-510) is never used and should
be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
PoolBase.YEAR_STAKE_WEIGHT_MULTIPLIER (CorePool.sol#671) is set pre-construction with a non-constant fu
nction or state variable:
- 2 * WEIGHT_MULTIPLIER
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#function-initializing-state
INFO:Detectors:
Pragma version0.8.13 (CorePool.sol#3) allows old versions
solc-0.8.13 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (CorePool.sol#381-386):
- (success) = recipient.call{value: amount}{} (CorePool.sol#384)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (CorePool.sol#408-419):
- (success,returndata) = target.call{value: value}(data) (CorePool.sol#417)
Low level call in Address.functionStaticCall(address,bytes,string) (CorePool.sol#425-434):
- (success,returndata) = target.staticcall(data) (CorePool.sol#432)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter PoolBase.initConfig(address,IPoolFactory,address,uint64,uint32)._buidl (CorePool.sol#828) is
not in mixedCase
Parameter PoolBase.initConfig(address,IPoolFactory,address,uint64,uint32)._factory (CorePool.sol#829) i
s not in mixedCase
Parameter PoolBase.initConfig(address,IPoolFactory,address,uint64,uint32)._poolToken (CorePool.sol#830)
is not in mixedCase

Parameter CorePool.stakeAsPool(address,uint256,uint256,uint256)._amount (CorePool.sol#1989) is not in m
ixedCase
Parameter CorePool.stakeAsPool(address,uint256,uint256,uint256)._liquidPercent (CorePool.sol#1990) is n
ot in mixedCase
Parameter CorePool.stakeAsPool(address,uint256,uint256,uint256)._rewardLockPeriod (CorePool.sol#1991) i
s not in mixedCase
Parameter CorePool.transferBuidl(address,uint256)._to (CorePool.sol#2133) is not in mixedCase
Parameter CorePool.transferBuidl(address,uint256)._value (CorePool.sol#2133) is not in mixedCase
Parameter CorePool.transferBuidlFrom(address,address,uint256)._from (CorePool.sol#2144) is not in mixed
Case
Parameter CorePool.transferBuidlFrom(address,address,uint256)._to (CorePool.sol#2145) is not in mixedCa
se
Parameter CorePool.transferBuidlFrom(address,address,uint256)._value (CorePool.sol#2146) is not in mixe
dCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming
-conventions
INFO:Detectors:
Variable PoolBase.LIQUID_MULTIPLIER (CorePool.sol#665) is too similar to PoolBase.setConfiguration(uint
256,uint256,uint256,uint256)._liquidMultiplier (CorePool.sol#1579)
Variable PoolBase.WEIGHT_MULTIPLIER (CorePool.sol#661) is too similar to PoolBase.setConfiguration(uint
256,uint256,uint256,uint256)._weightMultiplier (CorePool.sol#1578)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar
INFO:Detectors:
Loop condition i < feeCycle.length (CorePool.sol#1527) should use cached array length instead of refere
ncing `length` member of the storage array.
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#cache-array-length
INFO:Slither:CorePool.sol analyzed (11 contracts with 93 detectors), 128 result(s) found
```

PoolBase.sol

```
INFO:Detectors:
PoolBase.setStakingConfig(uint256,uint256,uint256,uint256,address) (PoolBase.sol#1537-1548) should emit
an event for:
    - minStakeTimeForDiamond = _minStakeTime (PoolBase.sol#1546)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
INFO:Detectors:
PoolBase.setInitialSettings(address,address)._poolToken (PoolBase.sol#1600) lacks a zero-check on :
    - poolToken = _poolToken (PoolBase.sol#1603)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in PoolBase._processRewards(address,bool) (PoolBase.sol#1333-1396):
  External calls:
    - _sync() (PoolBase.sol#1340)
      - factory.updateBUIDLPerBlock() (PoolBase.sol#1291)
  State variables written after the call(s):
    - user.deposits.push(newDeposit) (PoolBase.sol#1367)
    - user.tokenAmount += pendingYield (PoolBase.sol#1370)
    - user.totalWeight += depositWeight (PoolBase.sol#1371)
Reentrancy in PoolBase._sync() (PoolBase.sol#1288-1324):
  External calls:
    - factory.updateBUIDLPerBlock() (PoolBase.sol#1291)
  State variables written after the call(s):
    - lastYieldDistribution = uint64(block.number) (PoolBase.sol#1305)
    - lastYieldDistribution = uint64(currentBlock) (PoolBase.sol#1320)
    - yieldRewardsPerWeight += rewardToWeight(buidlReward,usersLockingWeight) (PoolBase.sol#1319)
Reentrancy in PoolBase.stake(uint256,uint64) (PoolBase.sol#941-945):
  External calls:
    - stake(msg.sender, amount, lockUntil,false,0) (PoolBase.sol#943)
      - factory.updateBUIDLPerBlock() (PoolBase.sol#1291)
    - _processRewards(_staker,false) (PoolBase.sol#1212)
      - factory.updateBUIDLPerBlock() (PoolBase.sol#1291)
      - ICorePool(buidlPool).stakeAsPool(_staker,pendingYield,(user.liquidWeight * 100) / use
r.totalWeight,getRewardLockPeriod(_staker)) (PoolBase.sol#1378-1383)
  Event emitted after the call(s):
    - Synchronized(msg.sender,yieldRewardsPerWeight,lastYieldDistribution) (PoolBase.sol#1323)
      - _processRewards(_staker,false) (PoolBase.sol#1212)
    - YieldClaimed(msg.sender,_staker,pendingYield) (PoolBase.sol#1395)
      - _processRewards(_staker,false) (PoolBase.sol#1212)
Reentrancy in PoolBase.unstake(address,uint256,uint256) (PoolBase.sol#1188-1280):
  External calls:
    - _sync() (PoolBase.sol#1210)
      - factory.updateBUIDLPerBlock() (PoolBase.sol#1291)
    - _processRewards(_staker,false) (PoolBase.sol#1212)
      - factory.updateBUIDLPerBlock() (PoolBase.sol#1291)
      - ICorePool(buidlPool).stakeAsPool(_staker,pendingYield,(user.liquidWeight * 100) / use
r.totalWeight,getRewardLockPeriod(_staker)) (PoolBase.sol#1378-1383)
    - factory.transferYieldTo(msg.sender,_amount,liquidRewardAmount) (PoolBase.sol#1263)
    - transferPoolToken(address(0x000000000000000000000000000000000000000000000000000000000000dEaD),burnAmount) (PoolBase.s
ol#1268-1271)
      - SafeERC20.safeTransfer(IERC20(poolToken),_to,_value) (PoolBase.sol#1501)
    - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (Pool
Base.sol#536)
      - (success,returndata) = target.call{value: value}(data) (PoolBase.sol#417)
    - transferPoolToken(msg.sender,_amount - burnAmount) (PoolBase.sol#1274)
      - SafeERC20.safeTransfer(IERC20(poolToken),_to,_value) (PoolBase.sol#1501)
    - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (Pool
Base.sol#536)
```

```

- _updateStakeLock(msg.sender,depositId,lockedUntil) (PoolBase.sol#998)
- Synchronized(msg.sender,yieldRewardsPerWeight,lastYieldDistribution) (PoolBase.sol#1323)
- _processRewards(msg.sender,false) (PoolBase.sol#996)
- YieldClaimed(msg.sender,_staker,pendingYield) (PoolBase.sol#1395)
- _processRewards(msg.sender,false) (PoolBase.sol#996)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
PoolBase.pendingYieldRewards(address) (PoolBase.sol#846-882) uses timestamp for comparisons
Dangerous comparisons:
- block.number > lastYieldDistribution && usersLockingWeight != 0 (PoolBase.sol#857)
PoolBase._stake(address,uint256,uint64,bool,uint256) (PoolBase.sol#1083-1179) uses timestamp for comparisons
Dangerous comparisons:
- require(bool,string)(_lockUntil == 0 || (_lockUntil > block.timestamp && _lockUntil - block.timestamp <= 31536000),-10) (PoolBase.sol#1092-1097)
- assert(bool)(stakeWeight > 0) (PoolBase.sol#1147)
PoolBase._unstake(address,uint256,uint256) (PoolBase.sol#1188-1280) uses timestamp for comparisons
Dangerous comparisons:
- user.tokenAmount == 0 (PoolBase.sol#1245)
PoolBase._sync() (PoolBase.sol#1288-1324) uses timestamp for comparisons
Dangerous comparisons:
- usersLockingWeight == 0 (PoolBase.sol#1304)
PoolBase._processRewards(address,bool) (PoolBase.sol#1333-1396) uses timestamp for comparisons
Dangerous comparisons:
- pendingYield == 0 (PoolBase.sol#1347)
PoolBase._updateStakeLock(address,uint256,uint64) (PoolBase.sol#1405-1452) uses timestamp for comparisons
Dangerous comparisons:
- require(bool,string)(_lockedUntil > block.timestamp,-13) (PoolBase.sol#1411)
- require(bool,string)(_lockedUntil > stakeDeposit.lockedUntil,-14) (PoolBase.sol#1419)

```

```

PoolBase.initConfig(address,IPoolFactory,address,uint64,uint32) (PoolBase.sol#826-838) is never used and should be removed
ReentrancyGuard._reentrancyGuardEntered() (PoolBase.sol#598-600) is never used and should be removed
SafeERC20.safeApprove(IERC20,address,uint256) (PoolBase.sol#488-501) is never used and should be removed
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (PoolBase.sol#512-523) is never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (PoolBase.sol#503-510) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
PoolBase.YEAR_STAKE_WEIGHT_MULTIPLIER (PoolBase.sol#670) is set pre-construction with a non-constant function or state variable:
- 2 * WEIGHT_MULTIPLIER
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#function-initializing-state
INFO:Detectors:
Pragma version0.8.13 (PoolBase.sol#3) allows old versions
solc-0.8.13 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (PoolBase.sol#381-386):
- (success) = recipient.call{value: amount}() (PoolBase.sol#384)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (PoolBase.sol#408-419):
- (success,returndata) = target.call{value: value}(data) (PoolBase.sol#417)
Low level call in Address.functionStaticCall(address,bytes,string) (PoolBase.sol#425-434):
- (success,returndata) = target.staticcall(data) (PoolBase.sol#432)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

```

```
INFO:Detectors:
Parameter PoolBase.initConfig(address,IPoolFactory,address,uint64,uint32)._buidl (PoolBase.sol#827) is
not in mixedCase
Parameter PoolBase.initConfig(address,IPoolFactory,address,uint64,uint32)._factory (PoolBase.sol#828) i
s not in mixedCase
Parameter PoolBase.initConfig(address,IPoolFactory,address,uint64,uint32)._poolToken (PoolBase.sol#829)
is not in mixedCase
Parameter PoolBase.initConfig(address,IPoolFactory,address,uint64,uint32)._initBlock (PoolBase.sol#830)
is not in mixedCase
Parameter PoolBase.initConfig(address,IPoolFactory,address,uint64,uint32)._weight (PoolBase.sol#831) is
not in mixedCase
Parameter PoolBase.pendingYieldRewards(address)._staker (PoolBase.sol#846) is not in mixedCase
Parameter PoolBase.balanceOf(address)._user (PoolBase.sol#890) is not in mixedCase
Parameter PoolBase.getDeposit(address,uint256)._user (PoolBase.sol#904) is not in mixedCase
Parameter PoolBase.getDeposit(address,uint256)._depositId (PoolBase.sol#904) is not in mixedCase
Parameter PoolBase.getDepositsLength(address)._user (PoolBase.sol#922) is not in mixedCase
Parameter PoolBase.stake(uint256,uint64)._amount (PoolBase.sol#941) is not in mixedCase
Parameter PoolBase.stake(uint256,uint64)._lockUntil (PoolBase.sol#941) is not in mixedCase
Parameter PoolBase.stakeFor(address,uint256,uint64)._staker (PoolBase.sol#958) is not in mixedCase
Parameter PoolBase.stakeFor(address,uint256,uint64)._amount (PoolBase.sol#959) is not in mixedCase
Parameter PoolBase.stakeFor(address,uint256,uint64)._lockUntil (PoolBase.sol#960) is not in mixedCase
Parameter PoolBase.unstake(uint256,uint256)._depositId (PoolBase.sol#976) is not in mixedCase
Parameter PoolBase.unstake(uint256,uint256)._amount (PoolBase.sol#976) is not in mixedCase
Parameter PoolBase.setWeight(uint32)._weight (PoolBase.sol#1041) is not in mixedCase
Parameter PoolBase.weightToReward(uint256,uint256).weight (PoolBase.sol#1462) is not in mixedCase
Parameter PoolBase.transferPoolToken(address,uint256)._to (PoolBase.sol#1496) is not in mixedCase
Parameter PoolBase.transferPoolToken(address,uint256)._value (PoolBase.sol#1496) is not in mixedCase
Parameter PoolBase.transferPoolTokenFrom(address,address,uint256)._from (PoolBase.sol#1510) is not in m
ixedCase
```

```
Parameter PoolBase.setConfiguration(uint256,uint256,uint256,uint256)._yearStakeWeightMultiplier (PoolBa
se.sol#1576) is not in mixedCase
Parameter PoolBase.setConfiguration(uint256,uint256,uint256,uint256)._weightMultiplier (PoolBase.sol#15
77) is not in mixedCase
Parameter PoolBase.setConfiguration(uint256,uint256,uint256,uint256)._liquidMultiplier (PoolBase.sol#15
78) is not in mixedCase
Parameter PoolBase.setInitialSettings(address,address)._factory (PoolBase.sol#1600) is not in mixedCase
Parameter PoolBase.setInitialSettings(address,address)._poolToken (PoolBase.sol#1600) is not in mixedCa
se
Parameter PoolBase.getRewardLockPeriod(address)._staker (PoolBase.sol#1607) is not in mixedCase
Parameter PoolBase.addPresaleAddress(address)._presale (PoolBase.sol#1636) is not in mixedCase
Variable PoolBase.WEIGHT_MULTIPLIER (PoolBase.sol#660) is not in mixedCase
Variable PoolBase.LIQUID_MULTIPLIER (PoolBase.sol#664) is not in mixedCase
Variable PoolBase.YEAR_STAKE_WEIGHT_MULTIPLIER (PoolBase.sol#670) is not in mixedCase
Variable PoolBase.REWARD_PER_WEIGHT_MULTIPLIER (PoolBase.sol#675) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

```
INFO:Detectors:
Variable PoolBase.LIQUID_MULTIPLIER (PoolBase.sol#664) is too similar to PoolBase.setConfiguration(uint
256,uint256,uint256,uint256)._liquidMultiplier (PoolBase.sol#1578)
Variable PoolBase.WEIGHT_MULTIPLIER (PoolBase.sol#660) is too similar to PoolBase.setConfiguration(uint
256,uint256,uint256,uint256)._weightMultiplier (PoolBase.sol#1577)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar
```

```
INFO:Detectors:
Loop condition i < feeCycle.length (PoolBase.sol#1526) should use cached array length instead of refere
ncing `length` member of the storage array.
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#cache-array-length
```

```
INFO:Slither:PoolBase.sol analyzed (9 contracts with 93 detectors). 97 result(s) found
```

Solidity Static Analysis

PoolFactory.sol

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in PoolFactory.registerPool(address): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 249:2:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in PoolFactory.changePoolWeight(address,uint32): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 340:2:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

[more](#)

Pos: 962:59:

Gas costs:

Gas requirement of function `CorePool.updateStakeLock` is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 400:2:

Constant/View/Pure functions:

`PoolBase.getRewardLockPeriod(address)` : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 1014:2:

Similar variable names:

`PoolFactory.setRewardLockLimit(uint256,uint256)` : Variables have very similar names "maximumRewardLock" and "minimumRewardLock". Note: Modifiers are currently not considered by this static analysis.

Pos: 364:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

[more](#)

Pos: 324:4:

Delete from dynamic array:

Using "delete" on an array leaves a gap. The length of the array remains the same. If you want to remove the empty position you need to shift items manually and update the "length" property.

[more](#)

Pos: 638:6:

CorePool.sol**Check-effects-interaction:**

Potential violation of Checks-Effects-Interaction pattern in CorePool.stakeAsPool(address,uint256,uint256,uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 185:2:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

[more](#)

Pos: 502:23:

Gas costs:

Gas requirement of function `CorePool.addPresaleAddress` is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 1043:2:

Delete dynamic array:

The "delete" operation when applied to a dynamically sized array in Solidity generates code to delete each of the elements contained. If the array is large, this operation can surpass the block gas limit and raise an OOG exception. Also nested dynamically sized objects can produce the same results.

[more](#)

Pos: 978:4:

Constant/View/Pure functions:

`ICorePool.receiveVaultRewards(uint256)` : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 117:2:

Similar variable names:

`CorePool.pendingVaultRewards(address)` : Variables have very similar names "user" and "users". Note: Modifiers are currently not considered by this static analysis.

Pos: 107:21:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

[more](#)

Pos: 554:6:

Data truncated:

Division of integer values yields an integer value again. That means e.g. $10 / 100 = 0$ instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.

Pos: 1032:6:

PoolBase.sol**Check-effects-interaction:**

Potential violation of Checks-Effects-Interaction pattern in PoolBase._unstake(address,uint256,uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 595:2:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to

a certain degree, to change the outcome of a transaction in the mined block.

[more](#)

Pos: 502:23:

Delete dynamic array:

The "delete" operation when applied to a dynamically sized array in Solidity generates code to delete each of the elements contained. If the array is large, this operation can surpass the block gas limit and raise an OOG exception. Also nested dynamically sized objects can produce the same results.

[more](#)

Pos: 978:4:

Constant/View/Pure functions:

PoolBase.getRewardLockPeriod(address) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 1014:2:

Similar variable names:

PoolBase.pendingYieldRewards(address) : Variables have very similar names "user" and "users". Note: Modifiers are currently not considered by this static analysis.

Pos: 286:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

[more](#)

Pos: 1008:4:

Delete from dynamic array:

Using "delete" on an array leaves a gap. The length of the array remains the same. If you want to remove the empty position you need to shift items manually and update the "length" property.

[more](#)

Pos: 638:6:

Data truncated:

Division of integer values yields an integer value again. That means e.g. $10 / 100 = 0$ instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.

Pos: 1032:6:

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to lost tokens etc.
High	High level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g. public access to crucial functions.
Medium	Medium level vulnerabilities are important to fix; however, they cannot lead to lost tokens.
Low	Low level vulnerabilities are most related to outdated, unused etc. These code snippets cannot have a significant impact on execution.
Lowest Code Style/ Best Practice	Lowest level vulnerabilities, code style violations and information statements cannot affect smart contract execution and can be ignored.

Audit Findings

Critical:

No critical severity vulnerabilities were found.

High:

No high severity vulnerabilities were found.

Medium:

No medium severity vulnerabilities were found.

Low:

1) Critical operation lacks event log: **PoolFactory.sol**

```
/**
 * @dev Registers an already deployed pool instance within the factory
 *
 * @dev Can be executed by the pool factory owner only
 *
 * @param poolAddr address of the already deployed pool instance
 */
function removePool(address poolAddr) public onlyOwner {
    // read pool information from the pool smart contract
    // via the pool interface (IPool)
    address poolToken = IPool(poolAddr).poolToken();
    uint32 weight = IPool(poolAddr).weight();

    delete pools[poolToken];
    delete poolExists[poolAddr];
    // update total pool weight of the factory
    totalWeight -= weight;
}
```

Missing event log for: removePool

Resolution: Please write an event log for listed events.

Very Low:

1) Wrong comment:

PoolFactory.sol

```
266     /**  
267     * @dev Registers an already deployed pool instance within the factory  
268     *  
269     * @dev Can be executed by the pool factory owner only  
270     *  
271     * @param poolAddr address of the already deployed pool instance  
272     */  
273     function removePool(address poolAddr) public onlyOwner {  
274         // read pool information from the pool smart contract  
275         // via the pool interface (IPool)  
276         address poolToken = IPool(poolAddr).poolToken();  
277         uint32 weight = IPool(poolAddr).weight();  
278     }
```

Function `removePool` delete the deployed pool from the factory but comments are written as registers the pool within the factory which confuses the readers / code reviewers.

Resolution: Please update the correct comments for `removePool` function as per its functionality.

Conclusion

We were given a contract code in the form of a `gist.github` link. And have used all possible tests based on the given object. We had observed 1 low issue and 1 very low issue in the smart contracts. but those are not critical. So, it's good to go for the production.

We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

The security state of the reviewed contract is "**Secured**".

Note For Contract Users

BUIDL token holders can stake their tokens on Starter to gain access to the platform and earn rewards. Staking higher amounts of BUIDL gives holders increased access and even greater rewards.

Project Details Overview:

- SoftCap: Minimum amount for successful sale.
- HardCap: Maximum amount to trigger sale end.
- Min Per Wallet: Minimum and maximum contributions to non-IDO sales.
- Maximum Investment: Maximum allocation for IDO sale.
- Presale Rate: Cost for token purchase.
- DEX Listing Rate: Token price set when liquidity is allocated.
- Liquidity Allocation: Percentage of raised funds used for DEX liquidity listing.
- Liquidity Lock Duration: Number of days allocated liquidity will be locked.
- Open Time: Start time of sale.
- Close Time: End time of sale unless Hard Cap is reached.
- Listing Time: Time when DEX liquidity will be added and locked.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyse the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinised by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

RD Auditors Disclaimer

The smart contracts given for audit have been analysed in accordance with the best industry practices at the date of this report, in relation to: 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).

Because the total number of test cases are unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. 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 security of smart contracts.

Technical Disclaimer

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



Email: info@rdauditors.com

Website: www.rdauditors.com

