SOLID GROUP

AUDIT REPORT

DATE MARCH 24TH

FOR

KICKPAD

Solid Group Auditing Service
Telegram: @solid_1
Twitter: https://twitter.com/solid_group_1

Disclaimer

SolidGroup reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team.
SolidGroup Audits do not provide any warranty or guarantee **regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors. SolidGroup Audits should not be used in any way to make decisions around investment or involvement with any particular project**. **These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.**
SolidGroup Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Blockchain technology and cryptographic assets present a high level of ongoing risk. **SolidGroup’s position is that each company and individual are responsible for their own due diligence and continuous security.** SolidGroup in no way claims any guarantee of security or functionality of the technology we agree to analyze.

Overview

Contract Address: Not Deployed
Network: BSC (Binance Smart Chain)
Website: thekickpad.com
Telegram Group: https://t.me/kick_pad

Description

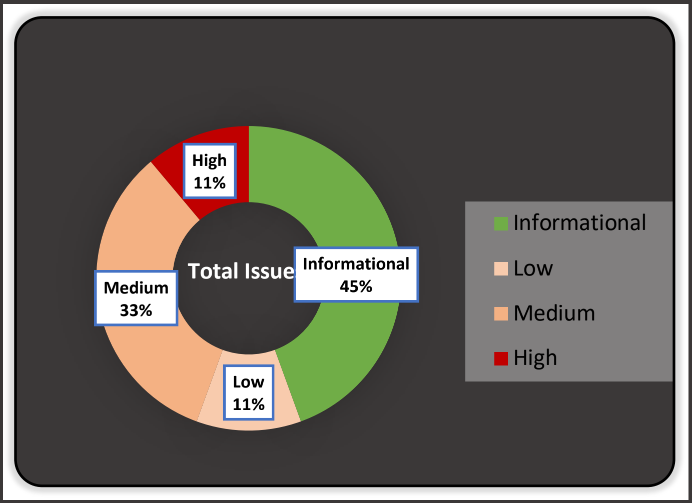
IDO platform that helps bring quality DeFi Projects to the public. Either stake or hold KickPAD tokens to get access to launchpad allocations.

Files In Scope

Contract Name	MD5
RBAC.sol	a12ac37cb977ebf9d2873a8e89dbdb3e
Factory.sol	31174420a90bdc2e1e29f8b484dc8854
StakingPool.sol	d99aca9d6bd0c4bb1479da54ae3a8ccd
TokenPool.sol	d99aca9d6bd0c4bb1479da54ae3a8ccd

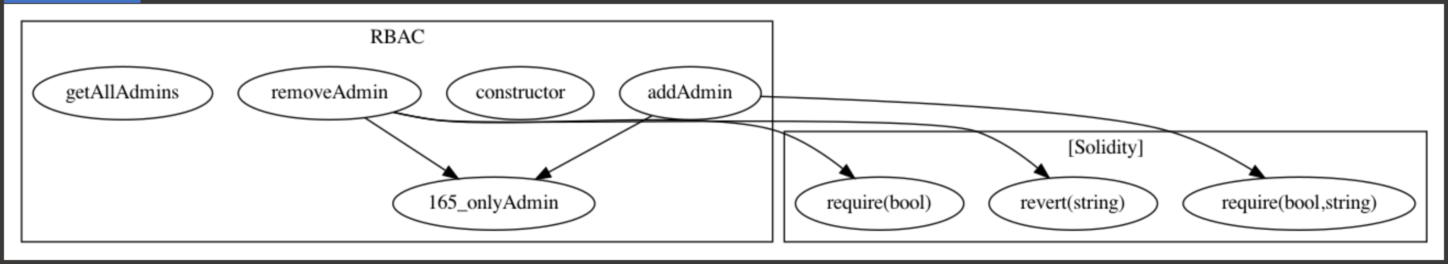
Vulnerability Summary

● Informational severity Issues	8
● Low severity issues	2
● Medium severity issues	6
● High severity issues	2

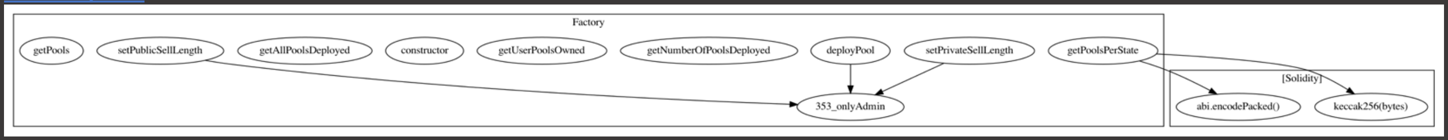


UML

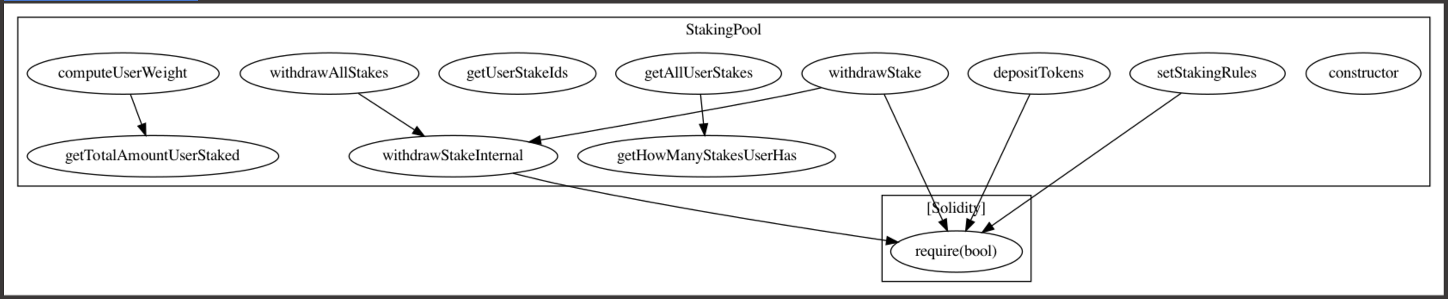
RBAC.sol



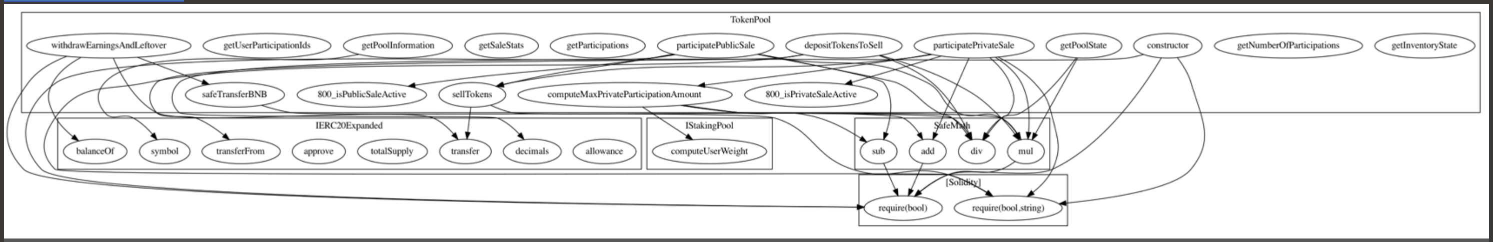
Factory.sol



StakingPool.sol



TokenPool.sol



Findings

RBAC.sol

Issue #1:

Type	Severity	Location
Gas Optimization	● Informational	RBAC.sol

Description:
The public addAdmin, removeAdmin, and getAllAdmins functions should be declared as external.

Recommendation:
These functions are only called from outside of the contract, consider using the external attribute instead of public to save gas.

Issue #2:

Type	Severity	Location
Gas Optimization	● Informational	RBAC.sol

Description:
In the removeAdmin function the loop iterates all admins and the number of iterations can grow beyond the block gas limit (unbounded array). However, this scenario is extremely unlikely and therefore the severity is only informational.

Recommendation:
Limit the number of possible admins, or iterate over a specific range

Summary

● Informational severity Issues	2
● Low severity issues	0
● Medium severity issues	0
● High severity issues	0

Factory.sol

Issue #1:

Type	Severity	Location
Gas Optimization	● Informational	Factory.sol

Description

The public deployPool , getNumberOfPoolsDeployed, getAllPoolsDeployed, getPoolsPerState , getPools, getUserPoolsOwned, setPublicSellLength and setPrivateSellLength functions should be declared as external.

Recommendation:

These functions are only called from outside of the contract. Consider using external attribute instead of public.

Issue #2:

Type	Severity	Location
Logical issues	● Low	Factory.sol

Description:

The function “deployToken” is using privateSaleLength and publicSaleLength variables which may be uninitialized. This can cause the presale to never start (salePublicStartTime == salePublicEndTime).

Recommendation:

Use require to ensure variables are initialized.

Issue #3:

Type	Severity	Location
Unbounded gas	● Medium	Factory.sol

Description:

The “getPoolsPerState” function iterates over all pools with a certain state (unbounded iteration). Although This function is declared as view (and therefore does not incur gas **cost**), such functions may be called by other contracts.

Recommendation:

Iterating over unbounded arrays is an anti-pattern and should be avoided. Our recommendation is to iterate on-demand (the client will ask for the item instead of pushing all the array to the client; An example of this pattern can be found in the Uniswap’s Factory contract) or iterate over a specific range.

Summary

● Informational severity Issues	1
● Low severity issues	1
● Medium severity issues	1
● High severity issues	0

TokenPool.sol

Issue #1:

Type	Severity	Location
Volatile Code	🔴 High	TokenPool.sol

Description:

In the “depositTokenToSell”, “sellTokens” and “withdrawEarningAndLeftover” functions, tokenSold variable is a generic ERC20 token. These functions call transfer/transferFrom on tokenSold, and ignores the return value. The ERC20 standard does not enforce a transfer function to revert on failure, which means the sellTokens() function may sometimes accept the user’s funds while **silently failing** to send him the tokens.

Recommendation:

When calling to transfer/transferFrom functions on tokenSold remember that tokenSold is a generic ERC20 token. Always ensure that transfer/transferFrom succeeds.

Issue #2:

Type	Severity	Location
Magic Number	🟢 Informational	TokenPool.sol L142, L317, L349

Description:

Magic numbers are considered an anti-pattern, and should be avoided.

Recommendation:

Consider replacing all significant magic numbers with named constants.

Issue #3:

Type	Severity	Location
Best Practice	🟢 Informational	TokenPool.sol L167

Description:

Missing events for critical operations. Events are highly important for the integration with certain DApps in the future.

Recommendation:

Consider using events for critical operations (For example: when core state of the contract is changed, such as privateSellTokenSold, totalTokensSold, totalBNBRaised, participation etc).

Issue #4:

Type	Severity	Location
Logic Error	● Medium	TokenPool.sol L230

Description:

The presale is finished only when time is reached, However consider a scenario where there are no more tokens to buy

Recommendation:

Consider marking the sale as over when the leftover variable is zero as well.

Issue #5:

Type	Severity	Location
Gas Optimization	● Informational	TokenPool.sol

Description:

The public depositTokensToSell, participatePrivateSale, participatePublicSale, withdrawEarningsAndLeftover, getUserParticipationIds, getNumberOfParticipations, getParticipations getSaleStats, getInventoryState, getPoolState and getPoolInformation functions should be declared as external.

Recommendation:

These functions are only called outside the contract consider using external attribute instead of public.

Issue #6:

Type	Severity	Location
Logical Error	● Low	TokenPool.sol L173-L190

Description:

In the “participatePublicSale” and “participatePrivateSale” functions, if the amountOfTokensSold – totalTokensSold > amountOfTokensBuying the transaction is reverted. However, there is a scenario which amountOfTokensSold – totalTokensSold == amountOfTokensBuying, which is perfectly fine

Recommendation:

Consider changing the require condition to: require(amountOfTokensSold – totalTokensSold >= amountOfTokensBuying)

Issue #7:

Type	Severity	Location
Logical Error	● High	TokenPool.sol

Description:

The contract doesn’t handle the case where a user participates multiple times in a private/public sale. In this case, max participation amount is irrelevant and can be overcome by participating multiple times.

Recommendation:

Track the participation history for each user.

Issue #8:

Type	Severity	Location
Best Practice	● Informational	TokenPool.sol L167

Description:

The error message is not indicative to the user. For example, in the common case where the user has no stakes he receives the error "overflow", instead a more sensible error message such as "user does not have enough stakes"

Recommendation:

Consider changing the error messages to be more indicative to non-technical user.

Summary

● Informational severity Issues	4
● Low severity issues	1
● Medium severity issues	1
● High severity issues	2

StakingPool.sol

Issue #1:

Type	Severity	Location
Logical Issue	● Medium	StakingPool.sol

Description:

The "depositTokens" function can be run before setStakingRule is initialized (setStakingRules was called).

Recommendation:

Consider adding a require to make sure StakingPool is initialized.

Issue #2:

Type	Severity	Location
Volatile Code	● Medium	StakingPool.sol

Description:

There are multiple places in the contract where there is an iteration over an unbounded array. In solidity due to block gas limit, iterating over unbounded array is anti-pattern and should be avoided. The problem is that you can't ensure that iterating over the array wouldn't reach block gas limit.

Recommendation:

There are various ways to handle this unbounded array scenario, for example iterate over a specific range, or instead of pushing the array to the client the client can request one item at a time.

Issue #3:

Type	Severity	Location
Volatile Code	● Medium	StakingPool.sol

Description:

In the userToHisStakeIds mapping, the user's stake is not removed when withdrawn. This array can grow indefinitely the more the user interacts with the platform.

Recommendation:

Our recommendation is instead of marking the stake as withdrawn (L147), override the element to be deleted with the last element of the array and pop the last element to reduce the size of the array.

Issue #4:

Type	Severity	Location
Best Practice	● Informational	StakingPool.sol

Description:

Lack of events in the contract.

Recommendation:

Our recommendation is to add events in critical parts of the staking contract, such us when a user withdraw it's stake (in withdrawStake function) or when a user depositTokens for stake (in depositTokens function). Events are great for integrating with DApps in the future. We recommend considering emitting events when state is changed.

Issue #5:

Type	Severity	Location
Arithmetic Error	● Medium	StakingPool.sol L223-L244

Description:

In the "computeUserWeight" function, the totalUserStaked variable can be zero when the user withdraws all of his stake. This can lead to division by zero error.

Recommendation:

Ensure totalUserStaked is not zero before division.

Summary

● Informational severity Issues	1
● Low severity issues	0
● Medium severity issues	4
● High severity issues	0