Guard

Smart contracts security assessment

Final report

Tariff: Standard

KeplerDAO

December 2021

0xguard.com

hello@0xguard.com

Contents

Introduction

Kepler is a DeFi investment protocol based on the KEEPER (KPR) token backed by the Kepler treasury. Each KPR is backed by assets in the Kepler treasury which consists of a diversified pool of KPR-stable coin LP tokens, stable coins such as USDC, DAI etc., and tokens from other protocols where the treasury invests.

Name	KeplerDAO
Audit date	2021-12-14 - 2021-12-14
Language	Solidity
Platform	Ethereum

Contracts checked

Name	Address
sKeplerERC20	
Staking	
EthBondDepository	
VLPBondDepository	
KeplerERC20	
StakingDistributor	
Treasury	
BondDepository	
wTROVE	
StandardBondingCalculator	
VaultOwned	

Procedure

We perform our audit according to the following procedure:

Automated analysis

- Scanning the project's smart contracts with several publicly available automated Solidity analysis tools
- Manual verification (reject or confirm) all the issues found by the tools

Manual audit

- Manually analyse smart contracts for security vulnerabilities
- Smart contracts' logic check

Known vulnerabilities checked

Title	Check result
Unencrypted Private Data On-Chain	passed
Code With No Effects	passed
Message call with hardcoded gas amount	passed
Typographical Error	passed
DoS With Block Gas Limit	passed
Presence of unused variables	passed
Incorrect Inheritance Order	passed
Requirement Violation	passed
Weak Sources of Randomness from Chain Attributes	passed
Shadowing State Variables	passed

Incorrect Constructor Name	passed
Block values as a proxy for time	passed
Authorization through tx.origin	passed
DoS with Failed Call	passed
Delegatecall to Untrusted Callee	passed
Use of Deprecated Solidity Functions	passed
Assert Violation	passed
State Variable Default Visibility	passed
Reentrancy	passed
Unprotected SELFDESTRUCT Instruction	passed
Unprotected Ether Withdrawal	passed
Unchecked Call Return Value	passed
Floating Pragma	passed
Outdated Compiler Version	passed
Integer Overflow and Underflow	passed
Function Default Visibility	passed

Classification of issue severity

High severity High severity issues can cause a significant or full loss of funds, change of contract ownership, major interference with contract logic. Such issues require immediate attention.

Medium severity Medium severity issues do not pose an immediate risk, but can be detrimental to the client's reputation if exploited. Medium severity issues may lead to a contract failure and can be fixed by modifying the contract state or redeployment. Such issues require attention.

Low severity Low severity issues do not cause significant destruction to the contract's functionality. Such issues are recommended to be taken into consideration.

Issues

High severity issues

1. The owner can call initializeBondTerms() multiple times - FIXED (EthBondDepository)

initializeBondTerms() function can be called repeatedly on the following occasions:

a. if adjustment.add = false and adjustment.rate = terms.controlVariable were set;

b. _controlVariable was equal to zero during initializeBondTerms() call.

Multiple terms setting can rewrite terms' properties and result in contract's math crash or great fee increase.

Update: Now to call initializeBondTerms() function terms.vestingTerms should be zero-initialized.

2. Excessive owner rights - FIXED (EthBondDepository)

The owner has access to the setStaking() function. They can substitute staking contract address with a malicious one, which will receive all next stakes' funds.

Recommendation: Transfer ownership to the Timelock contract (with a minimum delay of 48 hours). Although it doesn't fully eliminate the problem, at least users will be able to monitor pending changes. At the current state, users have no choice but to trust the owners of the audited contracts.

Update: setStaking() function was eradicated from the contract.

3. Typical issues - FIXED (VLPBondDepository)

Issues 1-2 are inherent.

4. Excessive owner rights - ACKNOWLEDGED (KeplerERC20)

The token's owner is allowed to change the vault variable and mint any amount of tokens.

Recommendation: same as for the issue 2

5. Excessive owner rights - ACKNOWLEDGED (Treasury)

The owner can set true value in isReserveToken or isLiquidityToken mapping for custom token and mint any amount of KEEPER tokens. Then he is able to exchange KEEPER tokens to any other token in the treasury with 1:1 rate using one of the strategies:

a.

1) mark himself as a ReserveSpender;

2) call withdraw() function.

b.

1) put all KEEPER tokens on staking;

2) receive sKEEPER tokens via claim() function;

3) mark himself as a Debtor;

4) call incurDebt() function.

Recommendation: same as for the issue 2

6. Typical issues - FIXED (BondDepository)

Issues 2-3 are inherent.

Medium severity issues

1. No zero check - FIXED (sKeplerERC20)

INDEX is used in the wTROVE contract for token amount calculation which a user will get while wrapping sKEEPER to wTROVE (L:28) and unwrapping wTROVE to sKEEPER (L:41). If INDEX is set to zero a user will not get wTROVE while wrapping or sKEEPER while unwrapping.

Update: requirement _INDEX != 0 was added.

2. No parameter check - ACKNOWLEDGED (Staking)

Accidental distributor variable setting to an invalid address will cause error throwing in rebase() and stake() functions. Thus, the main staking logic will be corrupted.

3. sKEEPER tokens are not fully backed with KEEPER - ACKNOWLEDGED (Staking)

There is no guarantee that the contract holds enough KEEPER tokens to make transfers in unstake() and forfeit() functions.

4. Exceeding gas limit - ACKNOWLEDGED (Staking)

rebase() function contains an external call to distribute() function in StakingDistributor, which iterates info array. When this array reaches a certain length, iteration over it will exceed the transaction's gas limit.

5. Absence of fault tolerance - ACKNOWLEDGED (Staking)

rebase() function contains the external call to distribute() function in StakingDistributor, which in turn address the treasury contract to mint rewards. If passed to mint _amount parameter in mintRewards() is greater than excessReserves(), the whole transactions will be reverted with "Insufficient reserves" message.

6. Users may lose funds - FIXED (EthBondDepository)

A user may lose their funds if they send to the deposit() function less or more ETH than _amount

argument. If the ETH is less than _amount the user loses all sent ETH, otherwise they just lose the difference between the sent ETH and the amount. In both situations, the user's lost ETH is sent to the DAO contract.

Update: Now _amount parameter should be strictly equal to sent ETH or the transaction will be reverted.

7. Wrong totalDebt calculation - PARTIALLY FIXED (EthBondDepository)

Every terms.vestingTerm variable change arouses totalDebt miscalculation in decayDebt() function. If the vesting period is shortened, calculated decay will be bigger than the real value, otherwise, it will be underestimated. This leads to debtRatio() returns wrong values, that cause bondPrice() inaccuracy.

Recommendation: Change the variable smoothly and as rarely as possible.

Update: Within the update was added the requirement on the currentDebt() to be equal to zero for vestingTerm change. This constraint guarantee that vesting terms of all bonds ended. However, if the currentDebt() returns zero, but totalDebt was not nullified with debtDecay() before, vestingTerm change still provokes problems described above.

Recommendation: Call decayDebt() function and check totalDebt to be equal to zero before vestingTerm change.

8. totalDebt can be bigger than terms.maxDebt - FIXED (EthBondDepository)

Max debt capacity reach L:181 is checked before the debt increase L:212. Consequently, totalDebt may exceed terms.maxDebt.

Update: Max debt requirement was moved to the line below of debt addition.

9. Possible underflow error - ACKNOWLEDGED (EthBondDepository)

In case of terms.controlVariable decrease in adjust() function, if adjustment.rate > terms.controlVariable L:296 subtraction will cause precision loss.

10. No checks on input parameters in function setAdjustment() on arguments _target and _buffer - ACKNOWLEDGED (EthBondDepository)

Since there are no requirements or checks of the passed _target and _buffer parameters, the admin can harshly and significantly increase or decrease terms.controlVariable.

11. Typical issues - PARTIALLY FIXED (VLPBondDepository)

Issues 1-5 are inherent.

12. Possible underflow error - ACKNOWLEDGED (StakingDistributor)

In case of rate decrease in the adjust() function, there is a possibility of an underflow error occurring. If adjustments[index].rate > info[index].rate, subtraction in L:82 will lead to loss of precision.

13. Index is not deleted - FIXED (StakingDistributor)

While deleting the recipient for distribution, the removeRecipient() function just nullifies the index, but does not pop it from the array. Thereby the info array length never decreases and gas consumption on array iteration in distribute() function is constantly raised.

Update: index is removed with memory freeing.

14. Typical issues - PARTIALLY FIXED (BondDepository)

Issues 1-5 are inherent.

Low severity issues

1. INDEX variable is unchangeable - ACKNOWLEDGED (sKeplerERC20)

It's impossible to change the INDEX variable after the initialize() function is called because of the L:62 requirement in setIndex().

2. Gas optimization - ACKNOWLEDGED (sKeplerERC20)

In the function rebase() the variable _totalSupply is read multiple times.

3. Gas optimization - PARTIALLY FIXED (Staking)

a. In the rebase() function there are multiple calls of contractBalance() and totalStaked() functions;

b. In the function rebase() there are multiple reads of the variables epoch.number and distributor;

c. In the stake() function there are multiple calls of the gonsForBalance() function of sKEEPER contract.

4. Gas optimization - ACKNOWLEDGED (StakingDistributor)

In the function In the function distribute() there are multiple reads of nextEpochTime, info.length and info[i].rate variables.

5. Gas optimization - FIXED (Treasury)

In the function valueOfToken() there is a call to KEEPER token to get decimals. But this value can be stored in the contract as an immutable variable.

Conclusion

KeplerDAO sKeplerERC20, Staking, EthBondDepository, VLPBondDepository, KeplerERC20, StakingDistributor, Treasury, BondDepository, wTROVE, StandardBondingCalculator, VaultOwned contracts were audited. 6 high, 14 medium, 5 low severity issues were found. There are 9 fixed and 4 partially fixed issues out of a total of 25.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability)set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without 0xGuard prior written consent.

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts 0xGuard to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model, or legal compliance.

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

0x Guard