Security Assessment

Sealem Lab

Jun 10th, 2022

Table of Contents

Summary

This report has been prepared for Sealem Lab to discover issues and vulnerabilities in the source code of the Sealem Lab project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	Sealem Lab
Platform	BSC
Language	Solidity
Codebase	https://github.com/sealemlab/sealemlab-core/tree/master/contracts
Commit	04989b908dba2db280d3c2b69de82683069ab1fb

Audit Summary

Delivery Date	Jun 10, 2022 UTC
Audit Methodology	Static Analysis, Manual Review

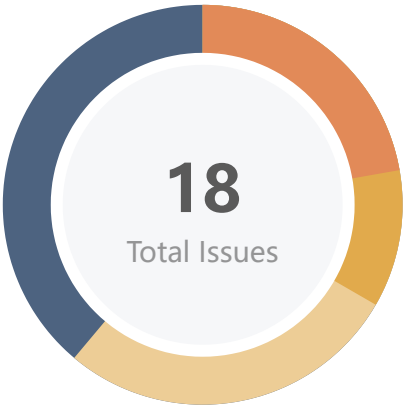
Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Mitigated	Partially Resolved	Resolved
Critical	0	0	0	0	0	0	0
Major	4	0	0	4	0	0	0
Medium	2	0	0	1	0	0	1
Minor	5	0	0	5	0	0	0
Informational	7	0	0	4	0	0	3
Discussion	0	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
BDB	BondDepository.sol	6cfc296408d2216f718563239208716233c3fbd17e2617b9a4823a0705c805bd
INV	Inviting.sol	deea2da87d7fe20af96700df81bc80b69e3dec823d5d3f11aa4c92fb8ccfcd07
STS	STStaking.sol	0367a2241a62300b756b54c5e2b14ce20b6d9a81ea66dec2022bb5e5ca3b4144

Findings



Critical	**0** (0.00%)
Major	**4** (22.22%)
Medium	**2** (11.11%)
Minor	**5** (27.78%)
Informational	**7** (38.89%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
GLOBAL-01	Solidity Version Not Recommended	Language Specific	Informational	Acknowledged
BDB-01	Centralization Risk In BondDepository.sol	**Centralization / Privilege**	**Major**	Acknowledged
BDB-02	Potential Flashloan Attack	Logical Issue	Major	Acknowledged
BDB-03	Mismatch Of The Price	Logical Issue	Medium	Acknowledged
BDB-04	Third Party Dependencies	Volatile Code	Minor	Acknowledged
BDB-05	Unknown Selling Token	Volatile Code	Minor	Acknowledged
BDB-06	Potential Sandwich Attacks	Logical Issue	Minor	Acknowledged
BDB-07	Source Of The Selling Token	Volatile Code	Minor	Acknowledged
BDB-08	Variables That Could Be Declared As `constant`	Gas Optimization	Informational	Resolved
BDB-09	Redundant Code In The Function Claim()	Logical Issue	Informational	Acknowledged
BDB-10	Recommended Explicit `bondId` Validity Checks	Logical Issue	Informational	Resolved
BDB-11	Divide Before Multiply	Mathematical Operations	Informational	Acknowledged

ID	Title	Category	Severity	Status
INV-01	Centralization Risk In Inviting.sol	**Centralization / Privilege**	**Major**	Acknowledged
INV-02	Logic Issue On `userInviter[user]`	Logical Issue	Informational	Acknowledged
POO-01	Lack Of Reasonable Boundary	Volatile Code	Medium	Resolved
POO-02	Financial Models	Logical Issue	Minor	Acknowledged
POO-03	Missing Zero Address Validation	Coding Style	Informational	Resolved
STS-01	Centralization Risk In STStaking.sol	**Centralization / Privilege**	**Major**	Acknowledged

GLOBAL-01 | Solidity Version Not Recommended

Category	Severity	Location	Status
Language Specific	Informational		Acknowledged

Description

Solidity frequently releases new compiler versions. Using an old version prevents access to new Solidity security features. Also, recent versions may be too early to be trusted.

```
Pragma version>=0.8.12 (contracts/pool/interface/ISTStaking.sol#2) necessitates a version too recent to be trusted.
```

Recommendation

We recommend deploying with any of the following Solidity versions:

- 0.5.16 - 0.5.17
- 0.6.11 - 0.6.12
- 0.7.5 - 0.7.6

Use a simple pragma version that allows any of these versions. Also, consider using the latest version of Solidity for testing.

Alleviation

The team acknowledged this issue and they stated the following:

"They will adjust the version of solidity to 0.7.6 when deploying the contract."

BDB-01 | Centralization Risk In BondDepository.sol

Category	Severity	Location	Status
Centralization / Privilege	**Major**	BondDepository.sol	Acknowledged

Description

In the contract `BondDepository`, the role `MANAGER_ROLE` has the authority over the following function:

- function `setRate()`: change the tax, stake rates of the contract,
- function `setAddrs()`: change the addresses `treasury`, `STLP`, `inviting` and `stStaking`,
- function `create()`: create new `market` to sell STLP,
- function `closeBond()`: change the `markets[bondId].conclusion` to `block.timestamp` to block the functions `swapAndAddLiquidityAndBond()/bond()`,
- function `setBlackList()`: add/remove the users to/from the black list, and the black-listed users cannot perform function `claim()`.

Also, the role `DEFAULT_ADMIN_ROLE` which is the contract deployer has authority over the following functions:

- function `AccessControl.grantRole()`, which grants an address a privileged role,
- function `AccessControl.revokeRole()`, which revokes an address a privileged role.

Any compromise to the `MANAGER_ROLE` and contract deployer(DEFAULT_ADMIN_ROLE) accounts may allow the hacker to take advantage of this authority and users' assets may suffer loss.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement;
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles;
  OR
- Remove the risky functionality.

Alleviation

The team acknowledged this issue and they stated that they will use DAO voting and multi-signature wallet to control all the owner functions.

BDB-02 | Potential Flashloan Attack

Category	Severity	Location	Status
Logical Issue	Major	BondDepository.sol	Acknowledged

Description

Flash loans are a way to borrow large amounts of money for a certain fee. The requirement is that the loans need to be returned within the same transaction in a block. If not, the transaction will be reverted.

An attacker can use the borrowed money as the initial funds for an exploit to enlarge the profit and/or manipulate the token price in the decentralized exchanges.

We find that the `BondDepository` rely on price calculations that are based on-chain, meaning that they would be susceptible to flash-loan attacks by manipulating the price of given pairs to the attacker's benefit.

Recommendation

If a project requires price references, it needs to be caution of flash loans that might manipulate token prices. To minimize the chance of happening, we recommend the client to consider following according to the project's business model.

1. Use multiple reliable on-chain price oracle sources, such as Chainlink and Band protocol.
2. Use Time-Weighted Average Price (TWAP). The TWAP represents the average price of a token over a specified time frame. If an attacker manipulates the price in one block, it will not affect too much on the average price.
3. If the business model allows, restrict the function caller to be a non-contract/EOA address.
4. Flash loans only allow users to borrow money within a single transaction. If the contract use cases allowed, force critical transactions to span at least two blocks.

Alleviation

The team acknowledged this issue and they stated the following:

"They currently only have one liquidity pool for their tokens, so they cannot use LINK's oracles for the time being. Using TWAP will increase the cost of their project and increase the risk of user arbitrage. Therefore, their current plan is to monitor abnormal data and set the account with abnormal data as a blacklist."

BDB-03 | Mismatch Of The Price

Category	Severity	Location	Status
Logical Issue	Medium	BondDepository.sol	Acknowledged

Description

Like described in the `Financial Models` issue, the contract `BondDepository` is for users to swap token/LP for another specific token of the trading pair `STLP`. When calculating the selling price in the function `getStPrice()`, the token price is based on the price of BUSD. However, when calculating the buy price in the function `getLpPrice()`, it calls the function `getLpLiquidity()` which only calculate the LP price when either of token0/token1 is `WBNB`. If both of the tokens in the `markets[bondId].LP` pair are normal tokens, the price calculated is just its deposited amount while adding liquidity.

Recommendation

We recommend regulating the tokens in the `markets[bondId].LP` pair or adding related business logic to support all the cases.

Alleviation

The team acknowledged this issue and they stated the following:

"Their LP has only 2 cases, one consists of BUSD and the other consists of WBNB."

BDB-04 | Third Party Dependencies

Category	Severity	Location	Status
Volatile Code	Minor	BondDepository.sol	Acknowledged

Description

The contract is serving as the underlying entity to interact with third party `IPancakeRouter`, `STLP`, `treasury` contracts. The scope of the audit treats 3rd party entities as black boxes and assume their functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of 3rd parties can possibly create severe impacts, such as increasing fees of 3rd parties, migrating to new LP pools, etc.

Recommendation

We understand that the business logic requires interaction with `IPancakeRouter`, `STLP`, `treasury` etc. We recommend ensuring the third-party implementations and the way these functions are called can meet the requirements. We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

Alleviation

The team acknowledged this issue and they stated the following:

"Except for pancake's routing contracts, LP trading pairs and treasury's multi-signature wallets are managed by their team."

BDB-05 | Unknown Selling Token

Category	Severity	Location	Status
Volatile Code	Minor	BondDepository.sol	Acknowledged

Description

The contract `BondDepository` is actually selling one of the token in the `STLP` pair. However, the `STLP` pair address can be changed arbitrarily by the `MANAGER_ROLE`. Thus, the correctness of the `STLP` pair address, the actual token implementations and the price of the selling are all uncertain.

Recommendation

We recommend ensuring the token address correctness. Also ensure that the token implementations can meet the requirement.

Alleviation

The team acknowledged this issue and they stated the following:

"STLP will not be modified under normal circumstances, unless there is an unexpected situation, it will be modified through DAO voting."

BDB-06 | Potential Sandwich Attacks

Category	Severity	Location	Status
Logical Issue	Minor	BondDepository.sol	Acknowledged

Description

A sandwich attack might happen when an attacker observes a transaction swapping tokens or adding liquidity without setting restrictions on slippage or minimum output amount. The attacker can manipulate the exchange rate by frontrunning (before the transaction being attacked) a transaction to purchase one of the assets and make profits by backrunning (after the transaction being attacked) a transaction to sell the asset.

Potential sandwich attacks could happen if calling `router.swapExactTokensForETH()/swapExactTokensForTokens()/swapExactTokensForETH()` and `router.addLiquidity()` without setting restrictions on slippage.

Recommendation

We recommend setting reasonable minimum output amounts, instead of 0, based on token prices when calling the aforementioned functions.

Alleviation

The team acknowledged this issue and they stated the following:

"When a user makes a large and quick exchange on the official website, they will pop up a pop-up window for a risk reminder. It is recommended to go to the DEX to set a low slippage for exchange."

BDB-07 | Source Of The Selling Token

Category	Severity	Location	Status
Volatile Code	Minor	BondDepository.sol	Acknowledged

Description

According to the implementations of the contract `BondDepository`, the users can spend the LP tokens or normal tokens in the `markets.LP` pair to get an unknown token in the `STLP` pair.

The selling tokens are all sent from this contract, so the users may not get full amount when the balance of the selling token in the contract `BondDepository` is insufficient.

We would like to confirm with the client if more reward tokens would be transferred to this contract by the admin.

Recommendation

We recommend ensuring the `BondDepository` contract addresses have enough selling tokens.

Alleviation

The team acknowledged this issue and they stated the following:

"They will ensure that the amount of ST in the contract is sufficient when we start each bond sale."

BDB-08 | Variables That Could Be Declared As `constant`

Category	Severity	Location	Status
Gas Optimization	Informational	BondDepository.sol: 24, 27	Resolved

Description

The linked variables `BUSD`, `WBNB` could be declared as `constant` since these state variables are never modified.

Recommendation

We recommend to declare these variables as `constant`.

Alleviation

The team heeded our advice and resolved this issue in commit `73139d2aa4cbfb3497b2991d444e16a8271e91b4`.

BDB-09 | Redundant Code In The Function Claim()

Category	Severity	Location	Status
Logical Issue	Informational	BondDepository.sol: 435~438	Acknowledged

Description

In the function claim(), ST token will be used to pay the claimable no matter it's token0 or token1 of the STLP.

```
        (address token0, address token1) = getLPTokensAddrs(STLP);
        ...
        IERC20(token0 == BUSD || token0 == WBNB ? token1 : token0).safeTransfer(
                msg.sender,
                stPayout
            );
```

Recommendation

We recommend the client remove the redundant codes and use the safeTransfer function of ST token.

Alleviation

The team acknowledged this issue and they stated the following:

"To prevent unexpected situations, the ST contract address is not stored in the contract."

BDB-10 | Recommended Explicit `bondId` Validity Checks

Category	Severity	Location	Status
Logical Issue	Informational	BondDepository.sol: 231, 254	Resolved

Description

There's no sanity check to validate if a `bondId` is existing.

Recommendation

We advise the client to adopt following modifier `validatePoolByPid` to the linked functions.

```
modifier validateBondId(uint256 _pid) {
    require (bondId < markets.length , "bondId does not exist") ;
    _;
}
```

Alleviation

The team heeded our advice and resolved this issue in commit `73139d2aa4cbfb3497b2991d444e16a8271e91b4`.

BDB-11 | Divide Before Multiply

Category	Severity	Location	Status
Mathematical Operations	Informational	BondDepository.sol: 777~779, 790~791, 803~804, 812~813, 844~846	Acknowledged

Description

Performing integer division before multiplication truncates the low bits, losing the precision of calculation.

For example, in the `BondDepository` contract, the dynamic rates are calculated as follows,

```
function getBondRate(uint256 liquidity) public view returns (uint256) {
    return (liquidity / 1e22) * bondDynamicRate + bondBaseRate;
}
```

Recommendation

We recommend applying multiplication before division to avoid loss of precision.

Alleviation

The team acknowledged this issue and they stated that their interest rates are designed to be rounded.

INV-01 | Centralization Risk In Inviting.sol

Category	Severity	Location	Status
Centralization / Privilege	**Major**	Inviting.sol	ⓘ Acknowledged

Description

In the contract `Inviting`, the role `MANAGER_ROLE` has the authority over the following function:

- function `managerBindInviter()`, update the user's inviter, `userInviter[user]`.

Also, the role `DEFAULT_ADMIN_ROLE` which is the contract deployer has authority over the following functions:

- function `AccessControl.grantRole()`, which grants an address a privileged role,
- function `AccessControl.revokeRole()`, which revokes an address a privileged role.

Any compromise to the `MANAGER_ROLE` and contract deployer(DEFAULT_ADMIN_ROLE) accounts may allow the hacker to take advantage of this authority and users' assets may suffer loss.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations; AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;

AND

- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement;
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles;
  OR
- Remove the risky functionality.

Alleviation

The team acknowledged this issue and they stated that they will use DAO voting and multi-signature wallet to control all the owner functions.

INV-02 | Logic Issue On `userInviter[user]`

Category	Severity	Location	Status
Logical Issue	● Informational	Inviting.sol: 33	ⓘ Acknowledged

Description

According to the below code snippet, the `userInviter[user]` cannot be updated once it has been updated to another non-zero address. Thus, the first inviter will be the only invite beneficiary and get all the benefits.

```
30          if (
31              inviter != address(0) &&
32              inviter != user &&
33              userInviter[user] == address(0) &&
34              userInviter[inviter] != user
35          ) {
36              userInviter[user] = inviter;
37
38              emit BindInviter(user, inviter);
39          }
```

Recommendation

We recommend allowing `userInviter[user]` updating for non-zero address or record the `affiliateStakedST` amount for each inviters.

Alleviation

The team acknowledged this issue and they stated that their invitation mechanism is designed to be bound only once.

POO-01 | Lack Of Reasonable Boundary

Category	Severity	Location	Status
Volatile Code	Medium	BondDepository.sol; STStaking.sol: 66	Resolved

Description

In both contracts `BondDepository` and `STStaking`, the variables `taxBaseRate` do not have a reasonable upper limit, why would the contracts charge 50% tax fee?

```
require(_taxMaxRate <= 5000, "The tax max rate cannot exceed 50%");
```

Recommendation

We recommend adding reasonable upper boundary to all the above-mentioned variables.

Alleviation

The team heeded our advice and resolved this issue by updating the max tax rate to 30% in commit `73139d2aa4cbfb3497b2991d444e16a8271e91b4`.

POO-02 | Financial Models

Category	Severity	Location	Status
Logical Issue	Minor	BondDepository.sol; Inviting.sol; STStaking.sol	Acknowledged

Description

The main contract in the protocol is `BondDepository`, in which the users deposit normal token or LP of a specific trading pair (`markets[bondId].LP`) to get one of the token in the trading pair `STLP`. If normal token is deposited, half of the deposit amount will be swapped into the other token then used to add liquidity with the other left tokens. The result LP amount or the original LP deposit amount will be charged a fee of maximum 50%. `lpAmountTax` amount of LP will be sent to the `treasury` address, while the other `lpAmountPay` amount will be sent to the `market.receivingAddr` address. The contract then calculate the `usdPayout` and record the order with the below formula:

```
uint256 usdPayout = (lpAmountPay *
    lpPrice *
    (1e4 + bondRate + extraRates[3])) /
    1e18 /
    1e4;
```

In the formula, the `lpPrice` is calculated by the formula $\Delta LP = min(amount0 * totalSupply/reserve0, amount1 * totalSupply/reserve1)$. The bondRate + extraRates[3] are the extra benefit for staking or invitations.

Then, the user can call the function `claim()` to get a token in the `STLP` pair. The actual amount that users can get is calculated by the below formula:

```
uint256 stPrice = getStPrice();
uint256 stPayout = (usdPayout * 1e18) / stPrice;
```

In the formula, the `stPrice` is the token price based on BUSD.

And then, there are some concerns:

1. The protocol is actually selling a token in the `STLP` pair. The users pay an unknown LP and get the ST tokens. However, the important trading prices `lpPrice` and `stPrice` are not from price oracles, but only determined by the swapping rate with BUSD. Thus, there are risks of flash loan attack and leveraged arbitrage.

2. The actual selling token and the user deposit `market` LP are unknown. Also, the `STLP` pair address can be changed arbitrarily at any time by the `MANAGER_ROLE`. Thus, the current code logic and financial model may be not supported all the scenario.
3. As a protocol of selling token, the robustness of the whole protocol relies on the value of the selling token. However, it is out of the audit scope and we known nothing about it. There is no guarantee of the token price stability.
4. About the contract `STStaking`, it has no rewards and the only benefit of staking is to increase the user stake rate and user invite stake rate in the contract `BondDepository` which will increase the amount of ST tokens the user can claim. What's more, after the staking amount is withdrawn, there will be no rate benefit for the user and inviter.

We would like to confirm with the client if the current implementation aligns with the original project design.

Recommendation

Financial models of blockchain protocols need to be resilient to attacks. They need to pass simulations and verifications to guarantee the security of the overall protocol.

The financial model of this protocol is not in the scope of this audit.

Alleviation

The team acknowledged this issue and they stated the following:

"For issue 1, they will ensure that LP liquidity is not too small to reduce the risk of flash loan attacks. For issue 2, they will use DAO voting and multisig wallet to change these parameters. For issue 3, only this bond protocol will produce ST tokens, which can maximize price stability. For issue 4, before the project is officially launched, they will add sub-coin output to the pledge contract."

POO-03 | Missing Zero Address Validation

Category	Severity	Location	Status
Coding Style	Informational	BondDepository.sol: 181~184; STStaking.sol: 78~80	Resolved

Description

The address parameters should be checked before assignment to make sure it is not zero addresses.

For example:

- contract `BondDepository` : `_treasury`, `stlpAddr`, `invitingAddr`, `stStakingAddr` in the function `setAddrs()`; `lpAddr`, `receivingAddr` in the function `create()`;
- contract `STStaking` : `_treasury`, `stAddr`, `invitingAddr` in the function `setAddrs()`.

Recommendation

We recommend checking that these addresses are not zero with requires statements.

Alleviation

The team heeded our advice and resolved this issue in commit `73139d2aa4cbfb3497b2991d444e16a8271e91b4`.

STS-01 | Centralization Risk In STStaking.sol

Category	Severity	Location	Status
Centralization / Privilege	**Major**	STStaking.sol	ⓘ Acknowledged

Description

In the contract `STStaking`, the role `MANAGER_ROLE` has the authority over the following function:

- function `setOpenStatus()`, change the flag variable `openStatus` to block/unblock the function `deposit()`,
- function `setRate()`, change the variables `taxDynamicRate` and `taxBaseRate`,
- function `setAddrs()`, change the addresses `treasury`, `st` and `inviting`.

Also, the role `DEFAULT_ADMIN_ROLE` which is the contract deployer has authority over the following functions:

- function `AccessControl.grantRole()`, which grants an address a privileged role,
- function `AccessControl.revokeRole()`, which revokes an address a privileged role.

Any compromise to the `MANAGER_ROLE` and contract deployer(DEFAULT_ADMIN_ROLE) accounts may allow the hacker to take advantage of this authority and users' assets may suffer loss.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement;
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles;
  OR
- Remove the risky functionality.

Alleviation

The team acknowledged this issue and they stated that they will use DAO voting and multi-signature wallet to control all the owner functions.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

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 you ("Customer" or 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, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

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 CertiK 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.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we’ re able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.