CERTIK

Security Assessment

Ankr - BAS - Smart Contract Audit

Jul 25th, 2022

Table of Contents

Summary

This report has been prepared for Ankr to discover issues and vulnerabilities in the source code of the Ankr - BAS - Smart Contract Audit 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	Ankr - BAS - Smart Contract Audit
Platform	BSC
Language	Solidity
Codebase	https://github.com/Ankr-network/bas-genesis-config/tree/devel/contracts
Commit	b1ed612bdd1e08146d09c48b827fe4ed4cf65b3a c07cb96779e47a75cd3745bdd724c96184d2d609

Audit Summary

Delivery Date	Jul 25, 2022 UTC
Audit Methodology	Static Analysis, Manual Review

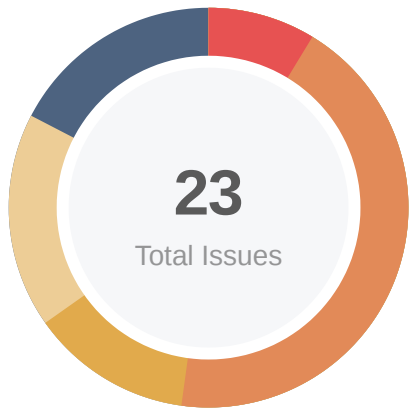
Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Mitigated	Partially Resolved	Resolved
Critical	2	0	0	1	0	0	1
Major	10	0	0	4	0	0	6
Medium	3	0	0	2	0	0	1
Minor	4	0	0	3	0	0	1
Optimization	0	0	0	0	0	0	0
Informational	4	0	0	3	0	0	1
Discussion	0	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
SAB	Staking.sol	ad2fdf8565190b1b9972fe91fa6fa4e044c7f783a5b0423381663f6330d20f83
IAB	Injector.sol	37a7d2351fa0e9e42907231de3a54651be952c045c45562e846eb1b2787902bf
SPA	StakingPool.sol	1eca905566e42760e6cedcb0e0d9d6ad35e94b3f1d5dd8a857afe1c14cef70bd
GAB	Governance.sol	5c76fc9e0b25d805bc0045a3ecbde8da89b577a243886d99f35a4c8637b3e234
SAU	Staking.sol	0e2d549ca9924a77477dda70c1b9d9f103bac5b346b18aa0870de82431728050
GAU	Governance.sol	2caf68fedf5e6ead15f496a8d06dc5c63f003e7bcb8672dd497c55745550e497
IAU	Injector.sol	37a7d2351fa0e9e42907231de3a54651be952c045c45562e846eb1b2787902bf
STA	StakingPool.sol	1eca905566e42760e6cedcb0e0d9d6ad35e94b3f1d5dd8a857afe1c14cef70bd
GAH	Governance.sol	c0eb1b155318950df358b08f5ddea838ab1a5ee355f544dc716caebf7ef3b082
ICH	InjectorContextHolder.sol	045339e287422c7584d65c89b5583c0459b2cbd21bd555382f73c7a6daa88ab6
SAH	Staking.sol	1c09e950280eff68fc650297c3e35b07c3ab264c6a8d51434b0abd25429bc007
STK	StakingPool.sol	dda4003ee442c7e20ea28ced43f3d60095a9a6a2fc272184a8a6f4e89b4f9970
RPA	RuntimeProxy.sol	fd40ac5d90fcb9ef807e2ce72688604be5d9c9a435ab666484c9e1c592666342

Findings



Critical	**2** (8.70%)
Major	**10** (43.48%)
Medium	**3** (13.04%)
Minor	**4** (17.39%)
Informational	**4** (17.39%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
CON-01	`_processDelegateQueue()` And `_calcDelegatorRewardsAndPendingUndelegates()` May Exceed Block Gas Limit	Volatile Code	Critical	Acknowledged
CON-02	Implementation Contract Not Initialized Automatically	Language Specific	Minor	Acknowledged
COT-01	Missing Error Messages	Coding Style	Informational	Acknowledged
COT-02	Lack Of Access Control	Control Flow	Informational	Acknowledged
GAB-01	Centralization Risks In Governance.sol	**Centralization / Privilege**	**Major**	Acknowledged
GAB-02	Upgradeable Contracts Must Inherit OpenZeppelin Upgradeable Contracts	Language Specific	Major	Resolved
GAB-03	`initializer()` Must Be Used Instead Of `whenNotInitialized()`	Logical Issue	Major	Resolved
GAU-01	Immutables In `EIP712` Will Not Work After Upgrade	Language Specific	Major	Resolved
IAB-01	Upgradeable Contracts Must Have Empty Constructor	Language Specific	Major	Resolved
IAB-02	Modifier `whenNotInitialized()` Must Be Used Instead Of `initializer()`	Logical Issue	Major	Resolved

ID	Title	Category	Severity	Status
IAB-03	Initializer Functions Of Abstract/base Contracts Should Be Internal	Logical Issue	Medium	Resolved
ICH-01	Incorrect Slot Number For `Initializable` Contract State Variables	Language Specific	Major	Resolved
ICH-02	Contract `Multicall` Is Not Used And It Is Not The Upgradeable Version	Logical Issue	Medium	Acknowledged
ICH-03	No Error Messages Defined For Errors	Language Specific	Informational	Acknowledged
RPA-01	Centralization Related Risks	**Centralization / Privilege**	**Major**	Acknowledged
SAB-01	Centralization Risks In Staking.sol	**Centralization / Privilege**	**Major**	Acknowledged
SAB-02	Missing Emit Events	Coding Style	Informational	Resolved
SAH-01	Validators With Non-zero Amount Of Delegation Or Pending Validator Rewards Should Not Be Removable	Logical Issue	Major	Acknowledged
SAH-02	Validators Can Not Be Slashed Multiple Times	Logical Issue	Medium	Acknowledged
SPA-01	Checks Effects Interaction Pattern Not Used	Volatile Code	Minor	Resolved
STA-01	Incorrect Calculation And Potential Integer Underflow	Mathematical Operations	Critical	Resolved
STA-02	Inaccurate Calculation	Mathematical Operations	Minor	Acknowledged
STK-01	Usage Of `transfer()` For Sending Ether	Volatile Code	Minor	Acknowledged

CON-01 | `_processDelegateQueue()` And `_calcDelegatorRewardsAndPendingUndelegates()` May Exceed Block Gas Limit

Category	Severity	Location	Status
Volatile Code	Critical	Staking.sol (v3): 412, 421, 460, 469; StakingPool.sol (v3): 88, 96	Acknowledged

Description

The identified loop logic in `_processDelegateQueue()` and `_calcDelegatorRewardsAndPendingUndelegates()` may exceed block gas limit when parameter `beforeEpochExclude`/`beforeEpoch` is far bigger than the epoch of last processed delegation. It can happen if functions `redelegateDelegatorFee()`/`claimDelegatorFee()`/`claimDelegatorFeeAtEpoch()` have not been called by a delegator for a long time. This is especially critical for `StakingPool` because it does not use `claimDelegatorFeeAtEpoch()` to process delegation little by little. Thus the modifier `advanceStakingRewards()` in `StakingPool` may always fail due to the use of `redelegateDelegatorFee()`/`calcAvailableForRedelegateAmount()`. Then the functions `stake()`/`unstake()`/`claim()` in `StakingPool` will always fail and users' funds are locked in the contract and lost.

Recommendation

We recommend handling the mentioned situation properly.

CON-02 | Implementation Contract Not Initialized Automatically

Category	Severity	Location	Status
Language Specific	Minor	Governance.sol (v3): 31~32; Staking.sol (v3): 137~138; StakingPool.sol (v3): 56~57	Acknowledged

Description

"Do not leave an implementation contract uninitialized. An uninitialized implementation contract can be taken over by an attacker, which may impact the proxy." See https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable#initializing_the_implementation_contract

Recommendation

We recommend invoking the `_disableInitializers()` function in the constructor.

Alleviation

[Ankr_BAS team]: Didn't get the problem. We have initializers in each smart contract and we use them.

COT-01 | Missing Error Messages

Category	Severity	Location	Status
Coding Style	Informational	Staking.sol (v1): 121, 724, 764; StakingPool.sol (v1): 185	Acknowledged

Description

The **require** can be used to check for conditions and throw an exception if the condition is not met. It is better to provide a string message containing details about the error that will be passed back to the caller.

Recommendation

We advise adding error messages to the linked **require** statements.

Alleviation

[Ankr-BAS team]: Agree, but storing this data in the code is very expensive in terms of bytecode size. Our contract has already spent almost 24kB.

COT-02 | Lack Of Access Control

Category	Severity	Location	Status
Control Flow	Informational	Governance.sol (v1): 17; Staking.sol (v1): 120	Acknowledged

Description

The following functions have no access control and anyone can call them when they are not initialized. In the contract `Staking`, the function `ctor()` will add any validator with an initial stake and commission rate. In the contract `Governance`, the function `ctor()` will set the voting period.

Recommendation

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

Alleviation

[Ankr-BSC team]: Agree, but it'll never happen because init function is called by consensus. Just to be 100% protected from this we can disallow anyone excpect coinbase to call this transaction, but it might break unit tests and requires some refactoring.

GAB-01 | Centralization Risks In Governance.sol

Category	Severity	Location	Status
Centralization / Privilege	**Major**	Governance.sol (v1): 29, 42	Acknowledged

Description

In the contract `Governance`, the owner of the validator has authority over the functions.

Any compromise to the owner of the validator may allow the hacker to take advantage of this authority and

- proposes with custom `VotingPeriod` through `proposeWithCustomVotingPeriod()`
- proposes with default `VotingPeriod` through `propose()`

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., multisignature 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

[Ankr-BSC team]: Any validator owner can only create a proposal. Without collecting 2/3 of the votes, the proposal won't be executed. Validators' owners can use a multi-sig wallet and manage their validator using multi-sig. Whether to use multi-sig or not is up to the validator's operator, and it's out of the BAS team's control.

GAB-02 | Upgradeable Contracts Must Inherit OpenZeppelin Upgradeable Contracts

Category	Severity	Location	Status
Language Specific	Major	Governance.sol (v1): 4~6, 14~15	Resolved

Description

All upgradeable contracts in an inheritance chain must follow the same rules for writing upgradeable contracts. Otherwise it will malfunction. See https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable#use-upgradeable-libraries

Recommendation

We recommend using OpenZeppelin upgradeable contracts and removing constructor.

Alleviation

Fixed in commit 34bec2ae5ff538b2247331dc835dc5229be97cd2

GAB-03 | `initializer()` Must Be Used Instead Of `whenNotInitialized()`

Category	Severity	Location	Status
Logical Issue	Major	Governance.sol (v1): 17	Resolved

Description

Initializer modifier for end contract(the contract actually deployed on chain) must be `initializer`, not `whenNotInitialized`. Otherwise, the initializer function can be called many times by anyone, including hackers.

Recommendation

We recommend replacing `whenNotInitialized` with `initializer`.

Alleviation

Fixed in commit 34bec2ae5ff538b2247331dc835dc5229be97cd2

GAU-01 | Immutables In `EIP712` Will Not Work After Upgrade

Category	Severity	Location	Status
Language Specific	Major	Governance.sol (v2): 10	Resolved

Description

The immutables in `EIP712` will not work after upgrade because the runtime upgrade evm hook does not run deployment code during upgrade. Thus the functions `castVoteBySig()/castVoteWithReasonAndParamsBySig()` in `Governor` will stop working after upgrade.

See https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.6.0/contracts/utils/cryptography/draft-EIP712.sol

Recommendation

We recommend using OpenZeppelin upgradeable contracts.

Alleviation

Fixed in commit 34bec2ae5ff538b2247331dc835dc5229be97cd2

IAB-01 | Upgradeable Contracts Must Have Empty Constructor

Category	Severity	Location	Status
Language Specific	Major	Injector.sol (v1): 40~41, 57~60, 73~74, 97~98, 101~117	Resolved

Description

RuntimeUpgrade feature allows new versions of system smart contracts to be redeployed at the same old address to replace old versions of code. Constructor must be empty to avoid re-initialization. All initialization must be done in the initializer function. See https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable#initializers

Recommendation

We recommend 1. removing constructor and `_ctor` state variable; 2. making the initializer functions of child contracts call `init()/initManually()` of `InjectorContextHolder`.

Alleviation

Fixed in commit 34bec2ae5ff538b2247331dc835dc5229be97cd2

IAB-02 | Modifier `whenNotInitialized()` Must Be Used Instead Of `initializer()`

Category	Severity	Location	Status
Logical Issue	Major	Injector.sol (v1): 62, 86	Resolved

Description

The modifier for `init()/initManually()` must be `whenNotInitialized()`, not `initializer()`. Otherwise, in case when child contracts use multiple inheritance, `initializer()` will not work.

Recommendation

We recommend using modifier `whenNotInitialized()` for `init()/initManually()`.

Alleviation

Fixed in commit 34bec2ae5ff538b2247331dc835dc5229be97cd2

IAB-03 | Initializer Functions Of Abstract/base Contracts Should Be Internal

Category	Severity	Location	Status
Logical Issue	Medium	Injector.sol (v1): 62, 86	Resolved

Description

Initializer functions of abstract/base contracts should be internal because they should only be called by initializer functions of child contracts. They should never be called externally. An example: https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/v4.4.2/contracts/token/ERC20/ERC20Upgradeable.sol

Recommendation

We recommend making `init()/initManually()` internal.

Alleviation

Fixed in commit 34bec2ae5ff538b2247331dc835dc5229be97cd2

ICH-01 | Incorrect Slot Number For `Initializable` Contract State Variables

Category	Severity	Location	Status
Language Specific	Major	InjectorContextHolder.sol (v3): 94	Resolved

Description

"data is stored contiguously item after item starting with the first state variable, which is stored in slot 0." So the slot number for `Initializable` contract state variables is zero, NOT one. By using incorrect slot number one, the function `isInitialized()` will always return incorrect value. See https://docs.soliditylang.org/en/v0.8.14/internals/layout_in_storage.html

Recommendation

We recommend changing the slot number from one to zero.

Alleviation

Fixed in commit 0ba8a2ce9437281a51b202bb6b393bb9058a6f87

ICH-02 | Contract `Multicall` Is Not Used And It Is Not The Upgradeable Version

Category	Severity	Location	Status
Logical Issue	Medium	InjectorContextHolder.sol (v3): 6, 19	Acknowledged

Description

The functionality of contract `Multicall` is not used at all. And it is not the upgradeable version. So it may impact future upgrades. See https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/v4.6.0/contracts/utils/MulticallUpgradeable.sol

Recommendation

We recommend either removing the `Multicall` contract or using the upgradeable version.

Alleviation

From BAS team: "Issue acknowledged. I won't make any changes for the current version."

ICH-03 | No Error Messages Defined For Errors

Category	Severity	Location	Status
Language Specific	Informational	InjectorContextHolder.sol (v3): 43~46	Acknowledged

Description

No error messages are defined for errors. It will make debugging/investigation very difficult. See https://docs.soliditylang.org/en/v0.8.14/contracts.html#errors-and-the-revert-statement

Recommendation

We recommend adding error messages for defined errors.

RPA-01 | Centralization Related Risks

Category	Severity	Location	Status
Centralization / Privilege	**Major**	RuntimeProxy.sol (v3): 15	Acknowledged

Description

In the contract `[RuntimeProxy]`, the role `[runtimeUpgrade]` has authority over the following functions:

- changeAdmin()
- upgradeTo()
- upgradeToAndCall()

Any compromise to the `[runtimeUpgrade]` account may allow a hacker to take advantage of this authority and upgrade the proxy to a malicious implementation contract to cause catastrophic consequence and potentially steal funds.

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.

Noted: Recommend considering the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

SAB-01 | Centralization Risks In Staking.sol

Category	Severity	Location	Status
Centralization / Privilege	**Major**	Staking.sol (v1): 198, 508, 540, 573, 587, 601, 612, 675, 709, 718, 779	Acknowledged

Description

In the contract `Staking` the role `Governance` has authority over the functions.

Any compromise to the `Governance` account may allow the hacker to take advantage of this authority and

- adds a validator through `addValidator()`
- removes a validator through `removeValidator()`
- activates a validator through `activateValidator()`
- disables a validator through `disableValidator()`

In the contract `Staking` the owner of the validator has authority over the functions.

Any compromise to the owner of the validator may allow the hacker to take advantage of this authority and

- changes the commission rate of himself/herself through `changeValidatorCommissionRate()`
- changes the owner of the validator through `changeValidatorOwner()`
- release himself/herself from jail through `releaseValidatorFromJail()`
- claims validator fee through `claimValidatorFee()`/`claimValidatorFeeAtEpoch()`

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., multisignature 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

[Ankr-BSC team]: We can suggest validator owners to use multisig wallet to avoid possible disclosure of the private key, but its not required. Right now we have 2/3 of quorum required to be reached between all validators to execute any governance and each validator owner can also "split" their key shares using threshold signatures or multisig. Also strict requirement on using of multisig smart contract here might disallow developers to use threshold signatures that might optimize all governance-related operations in terms of gas.

SAB-02 | Missing Emit Events

Category	Severity	Location	Status
Coding Style	Informational	Staking.sol (v1): 198, 675	Resolved

Description

There should always be events emitted in the sensitive functions that are controlled by centralization roles.

Recommendation

It is recommended emitting events for the sensitive functions that are controlled by centralization roles.

Alleviation

The client revised the code and resolved the issue in this commit.

SAH-01 | Validators With Non-zero Amount Of Delegation Or Pending Validator Rewards Should Not Be Removable

Category	Severity	Location	Status
Logical Issue	Major	Staking.sol (v3): 584~585	Acknowledged

Description

If a validator with non-zero delegation or pending validator rewards is removed, the functions `undelegate()`/`claimValidatorFee()`/`claimValidatorFeeAtEpoch()` will malfunction and users' funds can not be withdrawn.

Recommendation

We recommend not removing a validator when it has non-zero delegation or pending validator rewards.

SAH-02 | Validators Can Not Be Slashed Multiple Times

Category	Severity	Location	Status
Logical Issue	Medium	Staking.sol (v3): 826	Acknowledged

Description

The condition checking `slashesCount == _CHAIN_CONFIG_CONTRACT.getFelonyThreshold()` indicates that a validator can only be slashed once. A validator will not be slashed when `slashesCount` is a multiple of `_CHAIN_CONFIG_CONTRACT.getFelonyThreshold()`.

Recommendation

We recommend handling the mentioned situation properly.

Alleviation

From the BAS team: "Issue acknowledged. I won't make any changes for the current version."

SPA-01 | Checks Effects Interaction Pattern Not Used

Category	Severity	Location	Status
Volatile Code	Minor	StakingPool.sol (v1): 71, 134	Resolved

Description

In functions `stake()` and `unstake()` of the contract, the Checks Effects Interaction Pattern is not strictly followed. Using interfaces, the implementation of `claimDelegatorFee` or `delegate` is unknown and may have a malicious logical implementation that calls back to the function `stake()`. This is dangerous to the calculation like the user's rewards, the pool's totalStakedAmount, etc.

Recommendation

We advise developers to strictly follow the Checks-Effects-Interactions Pattern to avoid reentrancy and potential assets lost.

Alleviation

The client revised the code and resolved the issue in this commit.

STA-01 | Incorrect Calculation And Potential Integer Underflow

Category	Severity	Location	Status
Mathematical Operations	Critical	StakingPool.sol (v2): 100	Resolved

Description

The `_stakingContract.getDelegatorFee()` returns the amount of delegator rewards plus undelegated tokens in `_stakingContract`. The `validatorPool.pendingUnstake` is the total amount of undelegated tokens, which is the sum of undelegated tokens in `_stakingContract` and withdrawn undelegated tokens in current `StakingPool`. The unclaimed rewards should be `_stakingContract.getDelegatorFee()` minus undelegated tokens in `_stakingContract`, NOT `validatorPool.pendingUnstake`. The incorrect calculation may cause integer underflow and revert. The incorrect calculation is used by most functions, including `stake()`, `unstake()` and `claim()`, which means the contract may become totally useless and users' funds are locked and lost forever.

Recommendation

We recommend handling the mentioned situation properly.

Alleviation

Fixed in commit 34bec2ae5ff538b2247331dc835dc5229be97cd2

STA-02 | Inaccurate Calculation

Category	Severity	Location	Status
Mathematical Operations	Minor	StakingPool.sol (v2): 115	Acknowledged

Description

The correct ratio should be `validatorPool.sharesSupply * 1e18 / stakeWithRewards`. The returned ratio has 18 decimals which is more than enough to maintain precision.

Recommendation

We recommend using the suggested correct calculation.

STK-01 | Usage Of `transfer()` For Sending Ether

Category	Severity	Location	Status
Volatile Code	Minor	StakingPool.sol (v3): 184	Acknowledged

Description

After EIP-1884 was included in the Istanbul hard fork, it is not recommended to use `.transfer()` or `.send()` for transferring ether as these functions have a hard-coded value for gas costs making them obsolete as they are forwarding a fixed amount of gas, specifically `2300`. This can cause issues in case the linked statements are meant to be able to transfer funds to other contracts instead of EOAs.

Recommendation

We recommend replacing `transfer()` with the OpenZeppelin `Address` library function `sendValue()`.

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.

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.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

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.