DYP Finance

Security Assessment

Jan 21st 2021

By:
Boxi Li | CertiK
boxi.li@certik.org
Jialiang Chang | CertiK
jialiang.chang@certik.org

Summary

This report has been prepared for DYP.Finance smart contracts, Governance and Staking, to discover issues and vulnerabilities in the source code as well as any 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 security experts.

The security assessment resulted in 12 findings that ranged from minor to informational. We recommend to address these findings as potential improvements that can benefit the long run as both smart contracts would lock a significant amount of DYP tokens for a significant amount of time. We have done 2 rounds of communications over the general understanding and DYP team has resolved the questions promptly. Overall the source code is well written and the logic is straightforward, yet we suggest adding enough unit tests to cover the possible use cases given they are currently missing in the repository.

Overview

Project Summary

Name	DYP.Finance | https://dyp.finance/
Codebase	https://github.com/dypfinance/DYP-staking-governance-dapp

Engagement Summary

Delivery Date	Jan 20th, 2021
Methodology	Static analysis, manual review and testnet simulation
Contracts in Scope	2
Contract - GOV	governance-2.0
Contract - STAKING	constant-return-staking

Finding Summary

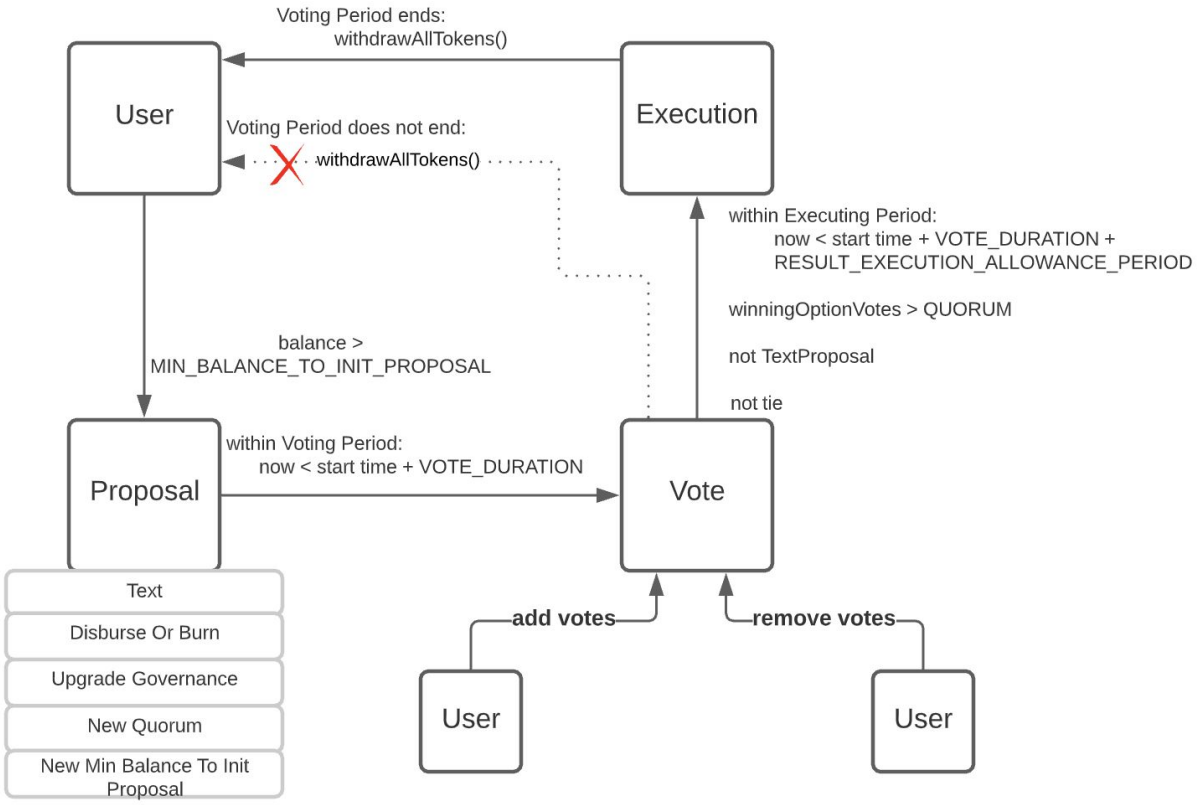
Total	12
Critical	0
Medium	0
Minor	1
Informational	11

Understanding of Core Logics

Governance

The Governance contract handles mainly on token disburses of the staking pools of 4 DYP pairs (WBTC, WETH, USDT, USDC), and internal events such as quorum change. When a proposal enters its execution stage (and meets the voting quorum), anyone can make the transaction for the execution of the proposal. The proposal will execute the action based on the option that may involve the external calls of the staking pools. Notice that it is by intention that an executed proposal does not always guarantee the execution statuses of the external calls. For example, a disburse-token proposal is still considered executed when some of the pools failed to disburse the tokens.

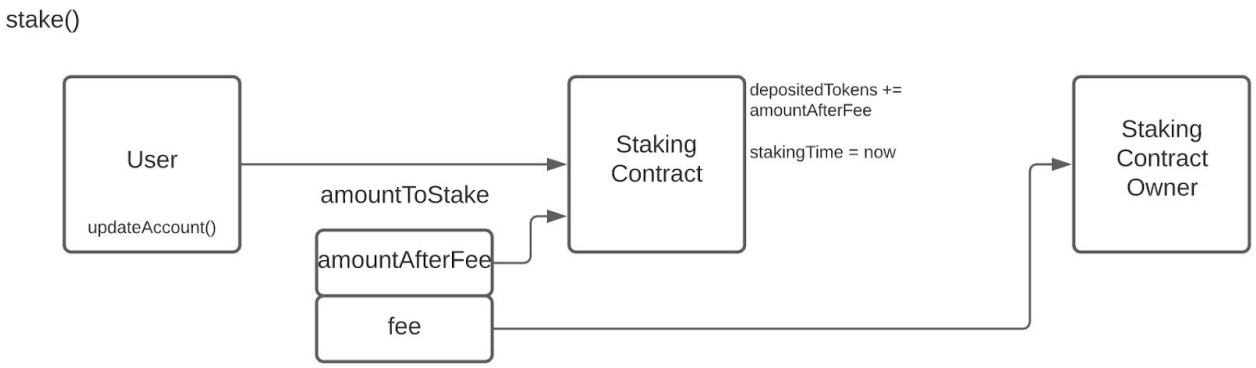
The admin privilege is limited as the contract owner has no direct access to the DYP tokens deposited to the contract as votes, thus leaving the tokens from voters intact. The admin has the permission to update the governance addresses of the staking pools via a proposal type, that may lead to a different governance contract in use, however the assets of the voters still remain and are able to withdraw.

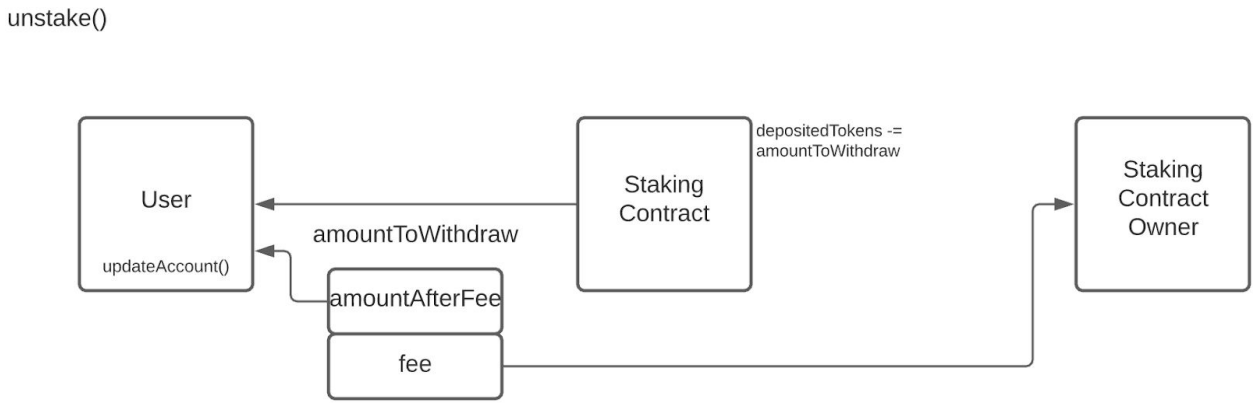


ConstantReturnStaking

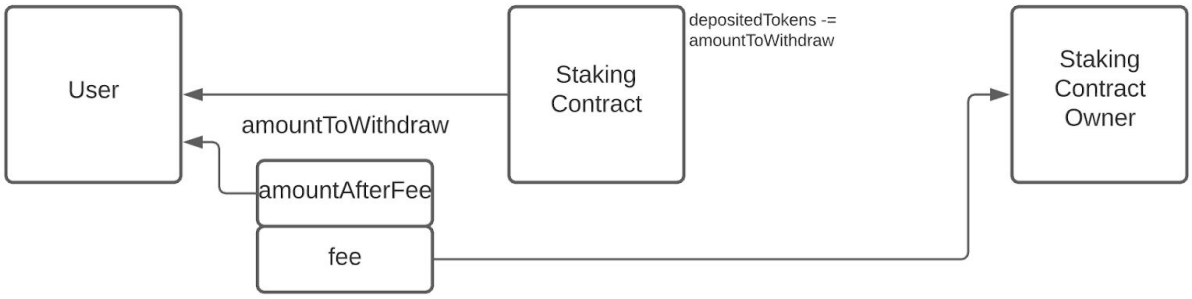
The staking contract requires a DYP user to lock its token to the contract for a period of time in return receiving the reward once the staking period ends. Pending rewards could be reinvested similar to the compound interest. The role referrer is introduced and plays an important role as major actions like stake or reinvest would incur the DYP token transfer to the referrer in a format of referral fee.

Though the staking period and reward rate are defined in the contract, there is no way to pre-calculate the total token rewards that requires the DYP team to provide funds beforehand. It is possible that the token balance of the staking contract would go below the total deposited tokens by users, which means the capital from stakers are not guaranteed to remain intact. We are informed that the DYP team will actively monitor the contracts and provide funds promptly, and a recommendation is provided in the report that could prevent the issue from happening.



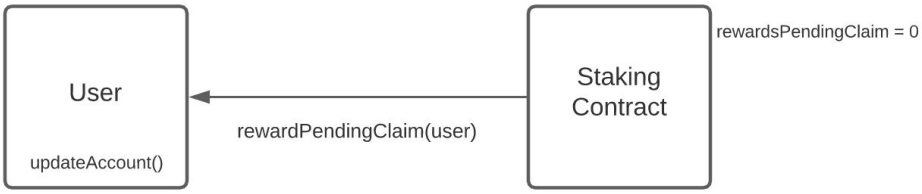


emergencyUnstake():
unstake() without updateAccount()

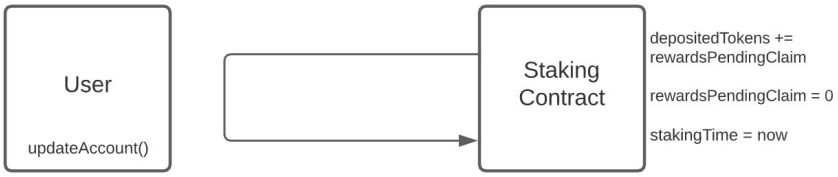


updateAccount()

claim()



reInvest()



Findings

ID	Title	Severity	Response
CTK-DYP-1	Reentrancy on Several Locations	Minor	Acknowledged
CTK-DYP-2	Temporary Denial of Service	Informational	Acknowledged
CTK-DYP-3	Total Balance Equation	Informational	Acknowledged
CTK-DYP-4	Calculation Simplification	Informational	Acknowledged
CTK-DYP-5	Duplicate Checks	Informational	Acknowledged
CTK-DYP-6	Presence of Unused Variable	Informational	Acknowledged
CTK-DYP-7	Magic Number	Informational	Acknowledged
CTK-DYP-8	State Variable Update	Informational	Acknowledged
CTK-DYP-9	Timestamp Dependence	Informational	Acknowledged
CTK-DYP-10	Event Emitting and Logging	Informational	Acknowledged
CTK-DYP-11	Require and Revert	Informational	Acknowledged
CTK-DYP-12	Ambiguous Comments	Informational	Resolved

CTK-DYP-1 | Reentrancy on Several Locations

Type	Severity	Location
Volatile Code	Minor	GOV-L571, 619 STAKING-L580, 600, 663, 669, 695-696, 719-720

Description

One of the major dangers of calling external contracts is that they can take over the control flow. In the reentrancy attack (a.k.a. recursive call attack), a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways.

```
function withdrawAllTokens() external noContractsAllowed {
    require(now >
lastVotedProposalStartTime[msg.sender].add(VOTE_DURATION), "Tokens are
still in voting!");
    require(Token(TRUSTED_TOKEN_ADDRESS).transfer(msg.sender,
totalDepositedTokens[msg.sender]), "transfer failed!");
    totalDepositedTokens[msg.sender] = 0;
}
```

Recommendation

We understand that `TRUSTED_TOKEN_ADDRESS` will be hardcoded and correctly reflected as the DYP token address (ERC20 interface), still it is considered as a good practice to follow the pattern.

The best practice to avoid Reentrancy weaknesses is to make sure all internal state changes are performed before the call is executed. This is known as the Check-Effects-Interaction Pattern.

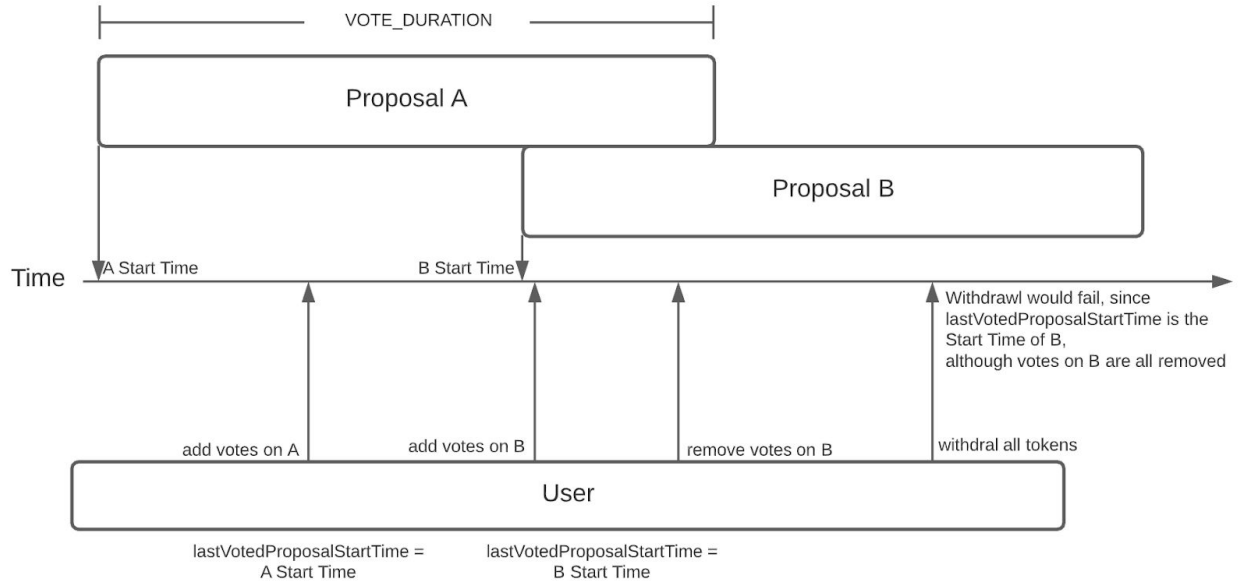
CTK-DYP-2 | Temporary Denial of Service

Type	Severity	Location
Volatile Code	Informational	GOV-L567-621

Description

Function `withdrawAllTokens()` would fail in certain situations. For example, a user:

1. Add votes on proposal A
2. Add votes on proposal B
3. Remove **all** votes on proposal B
4. When the voting duration of proposal A ends, and the voting duration of proposal B does not end, the `lastVotedProposalStartTime` is still the start time of B. Therefore, the call of `withdrawAllTokens()` would fail



Recommendation

It is debatable whether a total vote removal on B will mark the user's proposal time reverted back to A. Consider this corner case is unlikely to happen, no code change is requested on this finding.

CTK-DYP-3 | Total Balance Equation

Type	Severity	Location
Business Model	Informational	STAKING

Description

The DYP admin team will make its best effort to fund the contracts with enough tokens to cover the staking rewards. It is in theory possible that the withdrawal of staking rewards or referral fees could go beyond the funded amount and impact the tokens deposited. Ideally the capitals locked in the contracts shall remain intact.

Recommendation

Since the staking reward is not a fixed number that could be calculated ahead and funded by the admin, consider adding a condition check when the side effect of DYP token transfer happens that involves rewards or referral fees. For example:

```
require(token.balanceOf(this) >= totalDeposits + rewardToWithdraw)
```

CTK-DYP-4 | Calculation Simplification

Type	Severity	Location
Logical Issue	Informational	STAKING-L620

Description

Variable `timeDiff` is used as a multiplier on the calculation of the `pendingDivs`, when it is `0`, the final `pendingDivs` value would be `0` directly.

```
function getPendingDivs(address _holder) public view returns (uint) {
    ...
    if (lastClaimedTime[_holder] >= _now) {
        timeDiff = 0;
    } else {
        timeDiff = _now.sub(lastClaimedTime[_holder]);
    }

    uint stakedAmount = depositedTokens[_holder];

    uint pendingDivs = stakedAmount
                        .mul(REWARD_RATE_X_100)
                        .mul(timeDiff)
                        .div(REWARD_INTERVAL)
                        .div(1e4);

    return pendingDivs;
}
```

Recommendation

Consider early return when `timeDiff` is `0`.

CTK-DYP-5 | Duplicate Checks

Type	Severity	Location
Logical Issue	Informational	GOV-L630 & 714

Description

The check of `optionOneVotes[proposalId] != optionTwoVotes[proposalId]` duplicates in `executeProposal()` and `isProposalExecutible()`, where `isProposalExecutible()` is called in `executeProposal()`.

```
function executeProposal(uint proposalId) external noContractsAllowed {
    require (actions[proposalId] != Action.TEXT_PROPOSAL, "Cannot
programmatically execute text proposals");
    require (optionOneVotes[proposalId] != optionTwoVotes[proposalId],
"This is a TIE! Cannot execute!");
    require (isProposalExecutible(proposalId), "Proposal Expired!");
    ...
}

function isProposalExecutible(uint proposalId) public view returns (bool) {
    if ((!isProposalOpen(proposalId)) &&
        (now <
proposalStartTime[proposalId].add(VOTE_DURATION).add(RESULT_EXECUTION_ALLOW
ANCE_PERIOD)) &&
        !isProposalExecuted[proposalId] &&
        optionOneVotes[proposalId] != optionTwoVotes[proposalId]) {
            return true;
        }
    return false;
}
```

Recommendation

Remove the duplicated check from `executeProposal()` to make the logic clearer.

CTK-DYP-6 | Presence of Unused Variable

Type	Severity	Location
Volatile Code	Informational	STAKING-L673, 701, 725

Description

Variable `holders`, type of `EnumerableSet.AddressSet`, have their elements added and removed in `state()`, `unstake()` and `emergencyUnstake()`. The boolean return values of functions `add(AddressSet, address)` and `remove(AddressSet, address)` are not properly handled.

```
function emergencyUnstake(uint amountToWithdraw) external
noContractsAllowed {
  ...
  if (holders.contains(msg.sender) && depositedTokens[msg.sender] == 0) {
    holders.remove(msg.sender);
  }
}
```

Recommendation

Use variables to receive the return value of the above mentioned functions, and handle the cases for both success and failure if needed by the business logic.

CTK-DYP-7 | Magic Number

Type	Severity	Location
Magic Number	Informational	STAKING-L577

Description

In `updateAccount()`, the `referralFee` is directly calculated by `REFERRAL_FEE_RATE_X_100` and `100e2`.

```
function updateAccount(address account) private {
   uint pendingDivs = getPendingDivs(account);
   if (pendingDivs > 0) {
       uint referralFee =
pendingDivs.mul(REFERRAL_FEE_RATE_X_100).div(100e2);
   ...
}

function stake(uint amountToStake, address referrer) external
noContractsAllowed {
  ...
  uint fee = amountToStake.mul(STAKING_FEE_RATE_X_100).div(1e4);
  ...
}
```

Recommendation

Prefer using `1e4` (aligned across other places used in the contract) or having a constant declared for data precision purpose.

CTK-DYP-8 | State Variable Update

Type	Severity	Location
Data Flow	Informational	STAKING-L588

Description

Functions like `stake()` and `unstake()` trigger the `updateAccount()` that mark the number changes, however to a user it has not claimed any reward with those function invocations.

```
function updateAccount(address account) private {
    uint pendingDivs = getPendingDivs(account);
    if (pendingDivs > 0) {
        ...
        totalEarnedTokens[account] =
totalEarnedTokens[account].add(amount);
        totalClaimedRewards = totalClaimedRewards.add(amount);
    }
    lastClaimedTime[account] = now;
}
```

Recommendation

Consider the updates of `totalEarnedTokens` and `totalClaimedRewards` move to `claim()` function, not inside the `updateAccount()`.

CTK-DYP-9 | Timestamp Dependence

Type	Severity	Location
Volatile Code	Informational	GOV-L487, 492, 618, 701, 712, 723, 730 STAKING-L615, 619, 688, 711, 784, 791

Description

Multiple comparisons are using the timestamp as a condition. Please make sure the users could understand the security risk level and trade-off of using `now` as one of the core factors in the contract.

Recommendation

Correct use of the 12-confirmation rule to make sure there are enough blocks confirmations. Referring to Vitalik's Answer and other related questions.

CTK-DYP-10 | Event Emitting and Logging

Type	Severity	Location
Coding Style	Informational	STAKING-L594

Description

The function `updateAccount()` is frequently called in other core functions and has numbers of changes on the state variables. Currently the event emit happens at `transferReferralFeeIfPossible`, however when the referrer is `address(0x00)` no event would emit.

```
function updateAccount(address account) private {
    uint pendingDivs = getPendingDivs(account);
    if (pendingDivs > 0) {
        ...

        bool success = transferReferralFeeIfPossible(referrals[account],
referralFee);

        ...

    }
    lastClaimedTime[account] = now;
}
```

Recommendation

Recommend adding a new event and emitting it in the function.

CTK-DYP-11 | Require and Revert

Type	Severity	Location
Coding Style	Informational	GOV-L578, 648

Description

Solidity has two syntax options to trigger exceptions from within other code blocks to flag an error and revert the current call, which are `require()` and `revert()`. The two syntax options are equivalent, it's developer preference which to use. Referring to Solidity Documentation.

```
function addVotes(uint proposalId, Option option, uint amount) external
noContractsAllowed {
    ...
    // if user is voting for this proposal first time
    if (votesForProposalByAddress[msg.sender][proposalId] == 0) {
        votedForOption[msg.sender][proposalId] = option;
    } else {
        if (votedForOption[msg.sender][proposalId] != option) {
            revert("Cannot vote for both options!");
        }
    }
    ...
}
```

Recommendation

Recommend keeping the consistency of coding style.

CTK-DYP-12 | Ambiguous Comments

Type	Severity	Location
Coding Style	Informational	GOV-L452

Description

Comment for state variable `lastVotedProposalStartTime` conflicts the variable name and the followup usages.

```
// address user => uint proposal id for the latest proposal the user voted
on
mapping (address => uint) public lastVotedProposalStartTime;
```

Recommendation

Recommend changing the comment to something like `“address user => start time for the latest proposal the user voted on”`.

Appendix | Finding Categories

Gas Optimization

Refer to exhibits that 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

Refer to exhibits that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Refer to exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

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

Volatile Code

Refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an instorage one.

Language Specific

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

Coding Style

Usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.

Business Model

Refer to contract or function logics that are debatable or not clearly implemented according to the design intentions.

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 CertiK's prior written consent.

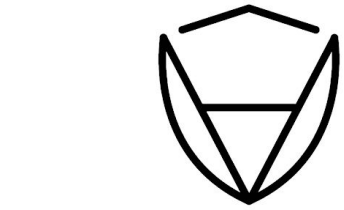
This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security 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.

About CertiK

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.



CERTIK

Provable Trust For All