BlockÂpex

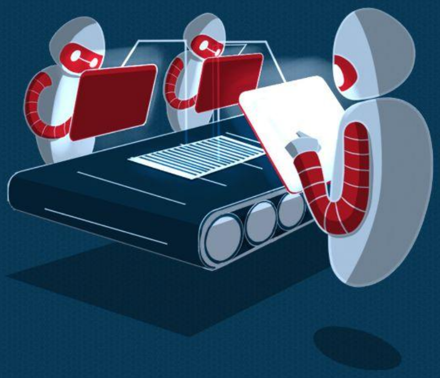
# SMART CONTRACT SECURITY ANALYSIS REPORT

```
pragma solidity 0.7.0;
contract Contract {

    function hello() public returns (string) {
        return "Hello World!";
    }

    function findVulnerability() public returns (string) {
        return "Finding Vulnerability";
    }

    function solveVulnerability() public returns (string) {
        return "Solve Vulnerability";
    }
}
```



Powered by XORD

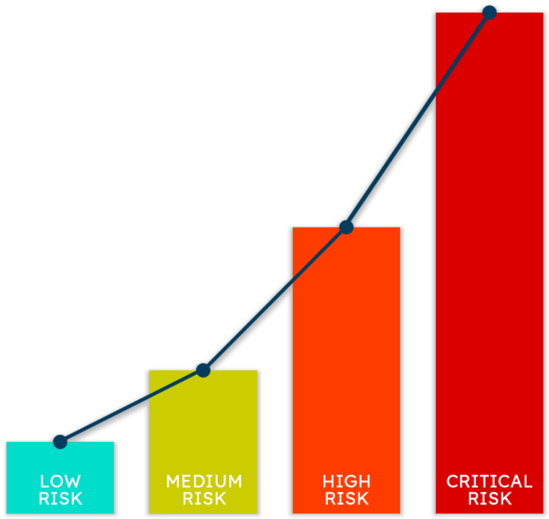
## PREFACE

### Objectives

The purpose of this document is to highlight the identified bugs/issues in the provided codebase. This audit has been conducted in a closed and secure environment, free from influence or bias of any sort. This document may contain confidential information about IT systems/architecture and intellectual property of the client. It also contains information about potential risks and the processes involved in mitigating/exploiting the risks mentioned below.

The usage of information provided in this report is limited, internally, to the client. However, this report can be disclosed publicly with the intention to aid our growing blockchain community; under the discretion of the client.

### Key understandings



| | |
|---|---|
| CRITICAL RISK xxxx | Critical vulnerabilities are too easy to exploit and can lead to damages/loss in assets or manipulations. |
| HIGH RISK xxx | High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution. |
| MEDIUM RISK xx | Medium-level vulnerabilities are equally imperative to fix but they tend to have minimal impact on asset loss or data manipulations. |
| LOW RISK x | Lowest-level vulnerabilities, informational errors, violating code styles/practices usually can't affect smart contract execution; hence they can be ignored. |

The contents of this document are proprietary and highly confidential. Information from this document should not be extracted/disclosed in any form to a third party without the prior written consent of BlockApex.
BlockApex | Fortifying The Move Towards Decentralization

## TABLE OF CONTENTS

**PREFACE** **2**
Objectives 2
Key understandings 2
**INTRODUCTION** **4**
Scope 5
Project Overview 6
**AUDIT REPORT** **7**
Executive Summary 7
Update on initial findings 8
Critical-risk issues 8
High-risk issues 9
Medium-risk issues 10
Low-risk issues 11
Informatory issues 11
**DISCLAIMER** **12**

The contents of this document are proprietary and highly confidential. Information from this document should not be extracted/disclosed in any form to a third party without the prior written consent of BlockApex.
BlockApex | Fortifying The Move Towards Decentralization

## INTRODUCTION

BlockApex (Auditor) was contracted by __PhoenixDAO__ (Client) for the purpose of conducting a Smart Contract Audit/Code Review. This document presents the findings of our analysis which took place on ___28th October 2021____.

| Name |
|---|
| PhoenixDAO |
| **Auditor** |
| Muhammad Jariruddin \| Kaif Ahmed |
| **Platform** |
| Ethereum/Solidity |
| **Type of review** |
| LP staking \| DAO |
| **Methods** |
| Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review |
| **Git repository** |
| https://github.com/Musfirazia/PhoenixStaking/tree/3aec9851bd517e9e8da4d72193367cfa9f6cc863 |
| **White paper/ Documentation** |
| Not provided |
| **Document log** |
| Initial Audit published (4th October 2021) |
| Final Audit published (3rd November 2021) |

The contents of this document are proprietary and highly confidential. Information from this document should not be extracted/disclosed in any form to a third party without the prior written consent of BlockApex.
BlockApex | Fortifying The Move Towards Decentralization

## Scope

The git-repository shared was checked for common code violations along with vulnerability-specific probing to detect **major issues/vulnerabilities**. Some specific checks are as follows:

| **Code review** | | **Functional review** |
|---|---|---|
| Reentrancy | Unchecked external call | Business Logics Review |
| Ownership Takeover | ERC20 API violation | Functionality Checks |
| Timestamp Dependence | Unchecked math | Access Control & Authorization |
| Gas Limit and Loops | Unsafe type inference | Escrow manipulation |
| DoS with (Unexpected) Throw | Implicit visibility level | Token Supply manipulation |
| DoS with Block Gas Limit | Deployment Consistency | Asset's integrity |
| Transaction-Ordering Dependence | Repository Consistency | User Balances manipulation |
| Style guide violation | Data Consistency | Kill-Switch Mechanism |
| Costly Loop | | Operation Trails & Event Generation |

The contents of this document are proprietary and highly confidential. Information from this document should not be extracted/disclosed in any form to a third party without the prior written consent of BlockApex.
BlockApex | Fortifying The Move Towards Decentralization

## Project Overview

**PhoenixDAO**

File: [DaoSmartContract.sol]

DAO smart contracts used to maintain onchain proposals. Voting is maintained off-chain. To avoid spams, proposers must have to deposit collateral in $PHNX.

**Steps to successful proposal:**

1. User submits a proposal and add collateral (Note: collateral is not deducted at at this point)
2. Off-chain voting happens
3. Admins change the status of proposals to ***Upvote*** on smart contract.
4. Collateral amount is deducted from the proposer's account.
5. When the status reaches ***completed***, proposer's collateral is returned back.

**Phoenix Spot Staking**

File: DaoStakeContract.sol

Users can earn spot staking rewards by depositing $PHNX for set period of time. If a user unstakes before the set duration, his portion (according to a formula) of staking amount will be burnt. Regardless, the user will get the full rewards at the time of staking.

**Phoenix LP staking**

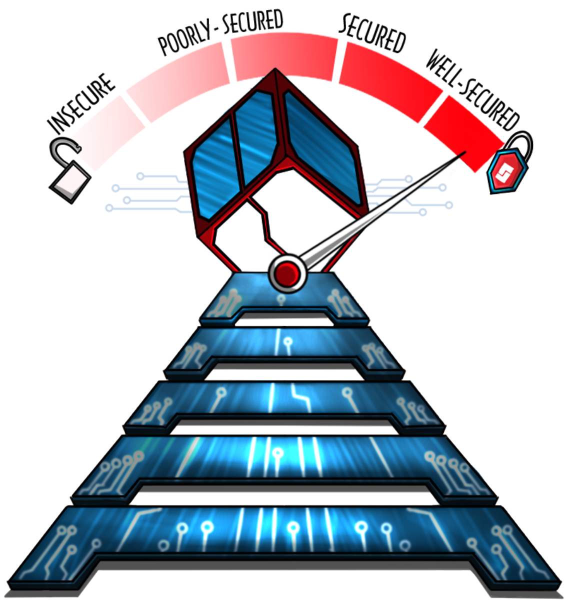
File: phxStake.sol

Users can earn rewards by staking their LP tokens. This staking style is similar to Pancakeswap's MasterChef.

The contents of this document are proprietary and highly confidential. Information from this document should not be extracted/disclosed in any form to a third party without the prior written consent of BlockApex.
BlockApex | Fortifying The Move Towards Decentralization

# AUDIT REPORT

## Executive Summary



The final analysis indicates that the contracts audited are **well-secured.** Most of the issues reported in the initial review have been fixed by the client. The contracts were reviewed again in order to ensure maximum security.

Our team performed a technique called “Filtered Audit”, where the contract was separately audited by two individuals. After their thorough and rigorous process of manual testing, an automated review was carried out using Mythril, MythX and Slither.

**Our team found:**

| # of issues | Severity of the risk |
|---|---|
| 0 | Critical Risk issue(s) |
| 0 | High Risk issue(s) |
| 0 | Medium Risk issue(s) |
| 0 | Low Risk issue(s) |
| 0 | Informatory issue(s) |

The contents of this document are proprietary and highly confidential. Information from this document should not be extracted/disclosed in any form to a third party without the prior written consent of BlockApex.
BlockApex | Fortifying The Move Towards Decentralization

## Update on initial findings

### Critical-risk issues

File: DaoSmartContract.sol

1. **User can withdraw the amount even if the actual deposit didn't happen**

   **Attack Scenario:**
   User submit a proposal with collateral amount **X**. As the collateral is deducted when the proposal status is changed to UpVotes by admins. User can withdraw collateral amount of tokens(even if the tokens are not actually deducted). Only checks in place are:

   ```
   require(!proposalList[proposalId].isClaimed, "Collateral
   Already Claimed");

   require(proposalList[proposalId].proposer == sender, "Only
   Owner of Proposal can withdraw");
   ```

   Though this function requires to be called by admins, we suggest that please verify at the backend that the user is eligible to withdraw or not.

   **Remedy:**
   Verify that the user has his collateral deducted by just adding a flag in proposal struct (***collateralDeposited***).

   **Dev's response:**
   It is only possible when the user has a signature and the user cannot have a signature**. Verified at the backend.**

The contents of this document are proprietary and highly confidential. Information from this document should not be extracted/disclosed in any form to a third party without the prior written consent of BlockApex.
BlockApex | Fortifying The Move Towards Decentralization

2. **User can withdraw even if the collateral is returned when the status of proposal is completed**

   Attack Scenario:
   When the admin changes the status of the  proposal to ***completed***, the contract automatically transfers the collateral back to the proposer. Still there are no checks placed and the collateral is returned back.

   Remedy:
   Add a flag (***collateralReturned***) in the ***proposal*** struct and verify the flag in withdraw.

   Dev's response:
   It is only possible when the user has a signature and the user cannot have a signature**. Verified at the backend.**

## High-risk issues

No high-risk issues were found.

The contents of this document are proprietary and highly confidential. Information from this document should not be extracted/disclosed in any form to a third party without the prior written consent of BlockApex.
BlockApex | Fortifying The Move Towards Decentralization

## Medium-risk issues

### 1. Possible DOS attack in unstake

File: DaoStakeContract.sol

When users deposit to earn spot staking rewards. ***StakerData*** holds the staking information. ***StakerData*** needs stakeId to find staking information. ***stakeId*** is generated as follows.

```
 bytes32 stakeId = keccak256(abi.encode(_timestamp, _beneficiary,
_altQuantity));
```

When the user unstakes, he has to provide ***stakeIds*** to unstake. Now the attacker can use the stakeIds of other users with the following constraint i.e., (sum of withdraw amount from all staking events should be less than attacker's staked amount). Attacker withdraws his amount, but now the original staker does not know his stakeIds.

```
 stakerBalance[sender] = stakerBalance[sender].sub(withdrawAmount);
```

Note:
Although the attacker might have his stakes burnt, the other users will be unable to unstake (at least other users will a have hard time to find ***stakeIds***).

Remedy:
We suggest verifying the ***stakeIds*** before processing.

Status:
Fixed.

2. LPTokens are transferred to $PHNX contract address

   The Lp staking smart contract is similar to PCS MasterChef contract. When the user deposits LPs, ***updatePool*** method is called. This method transfers some of LP tokens to the $PHNX contract address. This will cause issues when the user tries to unstake their LPs because the contract wouldn't have enough LPs.

   Status:
   Fixed.

## Low-risk issues

No high-risk issues were found.

## Informatory issues

1. *Pid*(pool id) arg is never used when a user withdraws in LP staking. We suggest removing that.

   File: PhxStake.sol

   Status:
   Fixed.

The contents of this document are proprietary and highly confidential. Information from this document should not be extracted/disclosed in any form to a third party without the prior written consent of BlockApex.
BlockApex | Fortifying The Move Towards Decentralization

## DISCLAIMER

The smart contracts provided by the client for audit purposes have been thoroughly analyzed in compliance with the global best practices till date w.r.t cybersecurity vulnerabilities and issues in smart contract code, the details of which are enclosed in this report.

This report is not an endorsement or indictment of the project or team, and they do not in any way guarantee the security of the particular object in context. This report is not considered, and should not be interpreted as an influence, on the potential economics of the token, its sale or any other aspect of the project.

Crypto assets/tokens are results of the emerging blockchain technology in the domain of decentralized finance and they carry with them high levels of technical risk and uncertainty. No report provides any warranty or representation to any third-Party in any respect, including regarding the bug-free nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third-party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project.

Smart contracts are deployed and executed on a blockchain. The platform, its programming language, and other software related to the smart contract can have its vulnerabilities that can lead to hacks. The scope of our review is limited to a review of the Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond Solidity that could present security risks.

This audit cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

The contents of this document are proprietary and highly confidential. Information from this document should not be extracted/disclosed in any form to a third party without the prior written consent of BlockApex.
BlockApex | Fortifying The Move Towards Decentralization