BlockApex

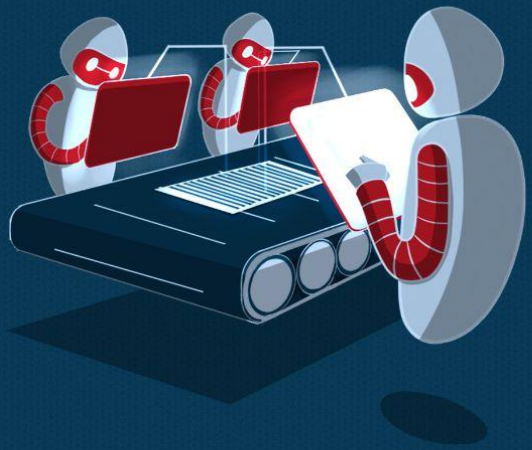
SMART CONTRACT SECURITY ANALYSIS REPORT

```
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 any 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

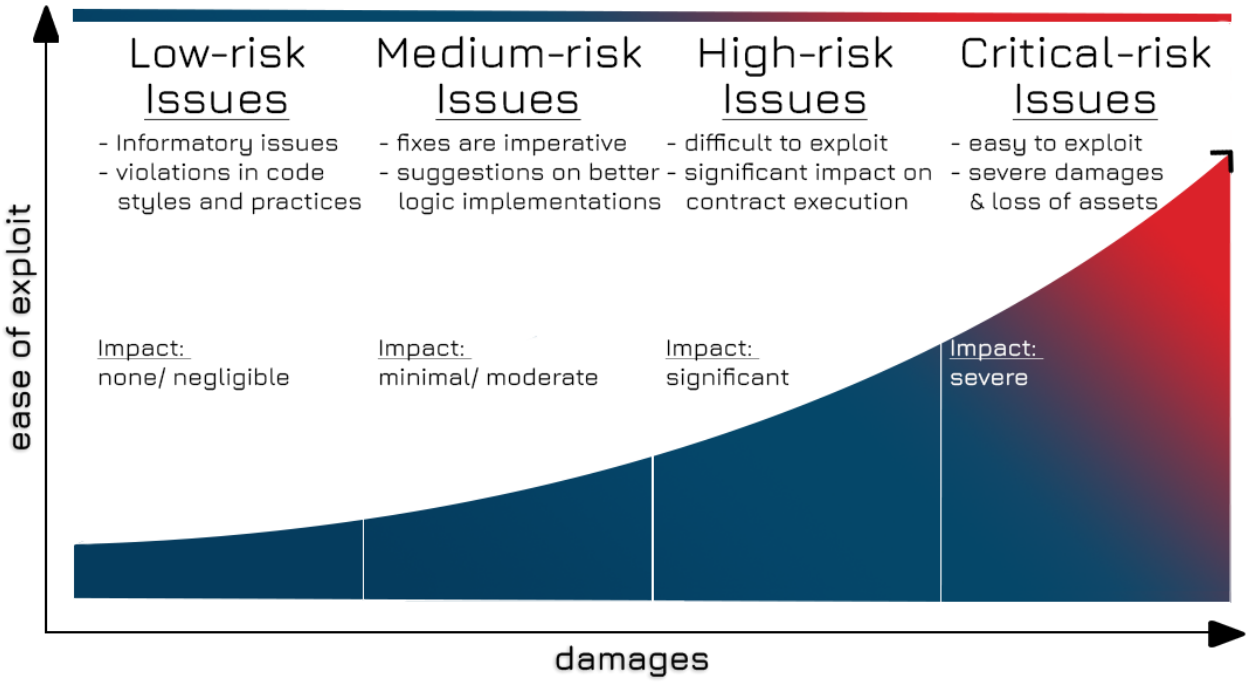


TABLE OF CONTENTS

INTRODUCTION

BlockApex (Auditor) was contracted by Voir Studio (Client) for the purpose of conducting a Smart Contract Audit/Code Review. This document presents the findings of our analysis which started on 22nd June 2022 .

Name
Unipilot Staking
Auditors
Kaif Ahmed | Faizan Nehal
Platform
EVM
Type of review
Manual Code Review | Automated Tools Analysis
Methods
Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Git repository/ Commit Hash
https://github.com/unipilot
White paper/ Documentation
Docs | Medium
Document log
Initial Audit Completed: June 25th, 2022
Final Audit Completed: June 27th, 2022

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

Project Overview

Unipilot Staking is a Staking infrastructure built on Ethereum, a reliable and scalable L1 solution. The staking solution offered by Unipilot provides the stakers a way to get incentives.

Founded in May 2021, Unipilot was the first of its kind liquidity optimizer product to offer the users to further optimize their liquidity on Uniswap. The advantages of using a liquidity optimizer include better returns on the provided liquidity on Uniswap.

System Architecture

The Unipilot protocol earns 20% of all revenues generated on the platform. Under the old model, all of this revenue was sent to the Index Fund, which already has a value of $245,000 at the time of writing. With the introduction of staking, you will be able to stake your $PILOT tokens on the Unipilot dApp to earn a share of protocol revenues. Initially, 40% of Treasury revenues will be distributed to stakers, though this percentage could change in the future.

Rewards will be distributed with every block and paid in $ETH, as chosen by the community. Stakers will be able to collect their $ETH rewards at any time, and remain in full control of their $PILOT, with the ability to unstake at any time without penalty.

As staking rewards originate from revenue earned by the protocol and not from inflationary $PILOT token rewards, returns are therefore not fixed. As TVL rises, the protocol will earn more revenue and staking rewards will increase.

Methodology & Scope

The codebase was audited using a filtered audit technique. A band of four (2) auditors scanned the codebase in an iterative process spanning over a time of two (1) week.

Starting with the recon phase, a basic understanding was developed and the auditors worked on developing presumptions for the developed codebase and the relevant documentation/whitepaper. Furthermore, the audit moved on with the manual code reviews with the motive to find logical flaws in the codebase complemented with code optimizations, software and security design patterns, code styles, best practices and identifying false positives that were detected by automated analysis tools.

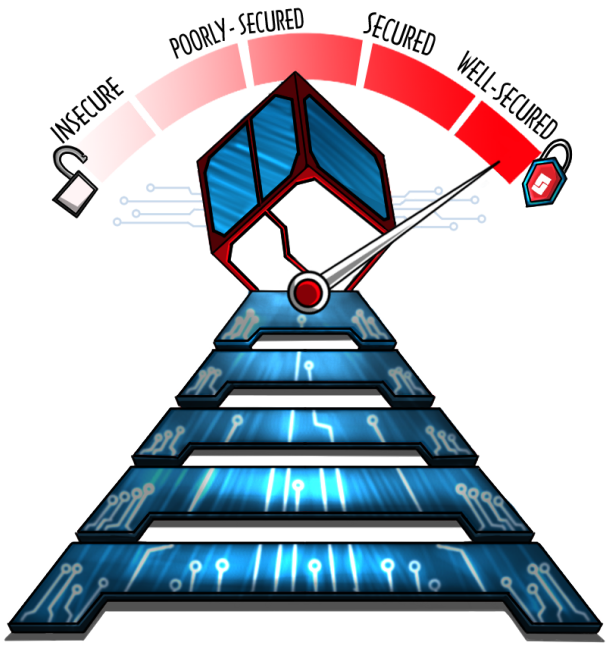
```
||||||
| **UnipilotStaking** | Implementation |  |||
| └ | <Constructor> | Public ❗ | 🛑 |NO❗ |
| └ | <Receive Ether> | External ❗ | 💵 |NO❗ |
| └ | setGovernance | External ❗ | 🛑 | onlyGovernance |
| └ | updateRewardToken | External ❗ | 🛑 | onlyGovernance |
| └ | updateRewards | External ❗ | 🛑 | onlyGovernance |
| └ | updateRewardEndBlock | External ❗ | 🛑 | onlyGovernance |
| └ | migrateFunds | External ❗ | 🛑 | onlyGovernance |
| └ | stake | External ❗ | 🛑 |NO❗ |
| └ | unstake | External ❗ | 🛑 |NO❗ |
| └ | emergencyUnstake | External ❗ | 🛑 |NO❗ |
| └ | claim | External ❗ | 🛑 |NO❗ |
| └ | calculatePendingRewards | External ❗ |  |NO❗ |
| └ | lastRewardBlock | External ❗ |  |NO❗ |
| └ | _stakeOrUnstakeOrClaim | Private 🔐 | 🛑 | |
| └ | _resetDebtIfNewRewardToken | Private 🔐 | 🛑 | |
| └ | _updateRewardPerPilotAndLastBlock | Private 🔐 | 🛑 | |
| └ | _calculatePendingRewards | Private 🔐 |  | |
| └ | _lastRewardBlock | Private 🔐 |  | |
| └ | _computeScalingFactor | Private 🔐 |  | |
| └ | _downscale | Private 🔐 |  | |
||||||
```

AUDIT REPORT

Executive Summary

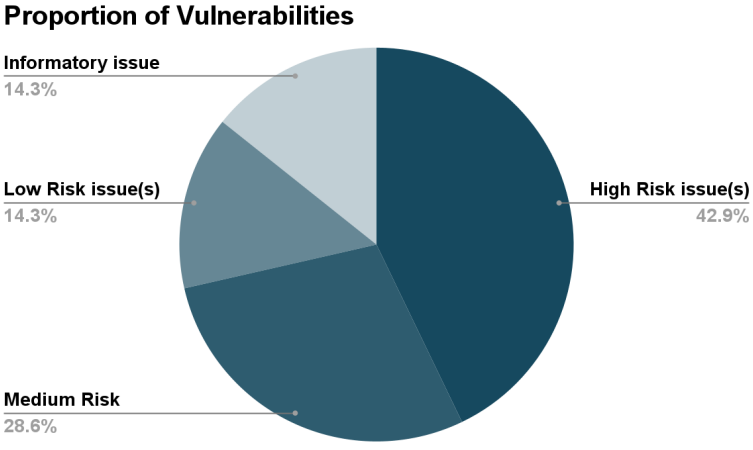
The analysis indicates that some of the functionalities in the contracts under the scope of audit are **working properly**.

Our team performed a technique called “Filtered Audit”, where the contract was separately audited by four individuals. After a thorough and rigorous process of manual testing, an automated review was carried out using slither for static analysis and Hardhat and Foundry for edge case testing. All the flags raised were manually reviewed and re-tested to identify the false positives.



Our team found:

# of issues	Severity of the risk
0	Critical Risk issue(s)
3	High Risk issue(s)
2	Medium Risk issue(s)
1	Low Risk issue(s)
1	Informatory issue(s)



Key Findings

#	Findings	Risk	Status
1.	Pending rewards for the previous reward token are getting lost	High	Fixed
2.	Incorrect behavior for pending reward tokens	High	Fixed
3.	Incorrect behavior for staking when no one has staked their tokens	High	Fixed
4.	Arithmetic overflow error on updating the reward token	Medium	Fixed
5.	Inconsistent behavior if the reward token is other than 18 decimals	Medium	Fixed
6.	Less optimized checks	Low	Acknowledged
7.	Redundant event emission	Informatory	Fixed

Detailed Overview

Critical-risk issues

No issues were found.

High-risk issues

1. **Pending Rewards For Previous Reward Tokens are Getting Lost**

   Description:
   If the reward token is changed by the governor then the pending reward for the previous reward tokens are lost, users will not receive their already accumulated rewards and the accumulated rewards will reset to 0. Contract does not hold any logic for giving out the pending rewards for previous token.

   Developers Response:
   Developers said that those pending rewards for the user will not be completely lost and they would be paid out to users eventually through a merkle tree.

   Remedy:
   There must be a map that will check the previous pending reward that is not paid to the stakers.

   Status:
   Fixed as per BlockAPex recommendation.

2. **Incorrect Behavior For Pending Reward Tokens**

   Description:

   When the reward token is changed and the previous reward token already had a reward for the user in it, then the calculated pending reward for the new reward token will add previous token reward as well as new token reward and return it to the user.

   Remedy:
   The contract must check if the whole reward this is provided is given out to the stakers.

   Status:
   Fixed as per BlockAPex recommendation.

3. **Incorrect Behavior For Staking When No One has Staked Their Tokens**

   Description:
   If the staking in the contract has already started and no one has staked their tokens yet, then the reward for empty past blocks will not be assigned to any staker.

   Developers Response:
   Initial permanent PILOT stake will be done by the foundation to set the initial state of the contract, rewards earned by the foundation will be distributed back to staking participants.

   Remedy:
   This default behavior of the contract should be changed and is not as per the intended behavior.

   Status:
   Fixed as per BlockAPex recommendation.

Medium-risk issues

4. **Arithmetic Overflow Error On Updating the Reward Token**

   Description:
   If the governance recalls the updateRewardToken() function with the address of the same reward token, then the contract is throwing an arithmetic overflow error.

   Remedy:
   The arithmetic overflow error should be checked properly.

   Status:
   Fixed as per BlockAPex recommendation.

5. **Inconsistent Behavior if the Reward Token is Other Than 18 Decimals**

   Description:

   If the reward token decimal is 6, 8 and 12 then the contract won't let the users to stake the PILOTs. There is a functionality in contract, when the user will come to stake their token, it will check whether the contract has the reward tokens in it or not, if the reward tokens will not be present then it will not let the user to stake their token.
   Now in case of different decimal places other than 18, the contract will get its own balance of reward token

   Remedy:
   The property should be added for the 6,8 and 12 decimals , just like for the 18 decimals.

   Status:
   Fixed as per BlockAPex recommendation.

Low-risk issues

6. Less Optimized Checks

Description:
In the updateRewards() function, it is checking the condition if (_rewardDurationInBlocks == 0) instead in should check if (_rewardDurationInBlocks == 0 || _reward == 0) because even if the rewardDurationInBlocks are provided but rewards are zero then there is no point in moving forward. 7 lines below this condition it is checking if (_reward == 0). So it would be more optimized if both of these conditions are checked in the same line.

Remedy:
These checks should be made into more optimized ones to save gas.

Developers Response:
Not valid as passing zero indicates the contract to distribute the existing debt rewards

Status:
Acknowledged.

Informatory issues and Optimizations

7. Redundant Event Emissions

Description:

There is no reason to emit the events at the end of the constructor, when the constructor is completed, two events GovernanceChanged() and RewardTokenChanged() are emitted. The code can be more optimized by removing these event emission.

Remedy:
Redundant events fired in the constructor should be removed.

Status:
Acknowledged.

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.