BlockÂpex

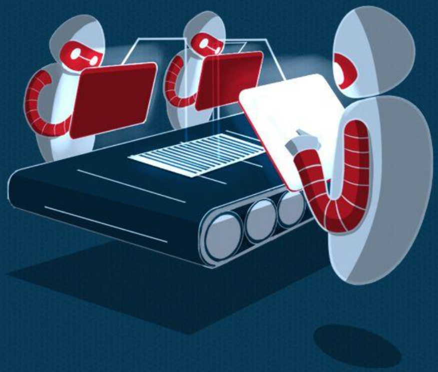
SMART CONTRACT SECURITY ANALYSIS REPORT

```
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

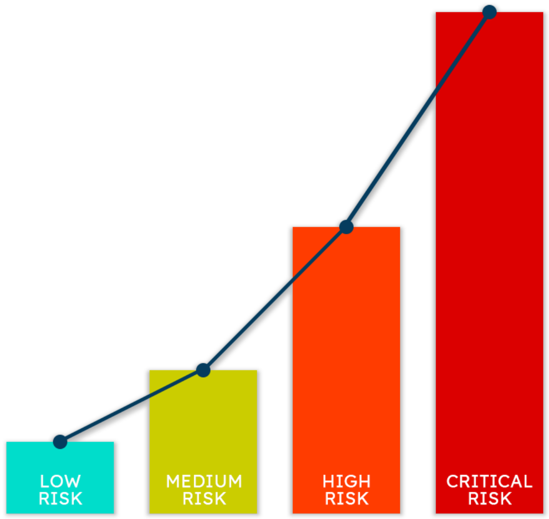
PREFACE

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.

Table of Contents

INTRODUCTION

BlockApex (Auditor) was contracted by __Unipilot__ (Client) for the purpose of conducting a Smart Contract Audit/Code Review. This document presents the findings of our analysis which took place on __13th August 2021__.

Name
Unipilot Index Fund
Auditor
Moazzam Arif | Kaif Ahmed
Platform
Ethereum/Solidity
Type of review
Index Fund
Methods
Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Git repository
https://github.com/VoirStudio/unipilot-index-fund-contract/tree/66a3655c152c2c3f2cbab1aa35712972ee086453
White paper/ Documentation
No white paper was provided
Document log
Initial Audit: 12-08-2021
Final Audit: 13-08-2021

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

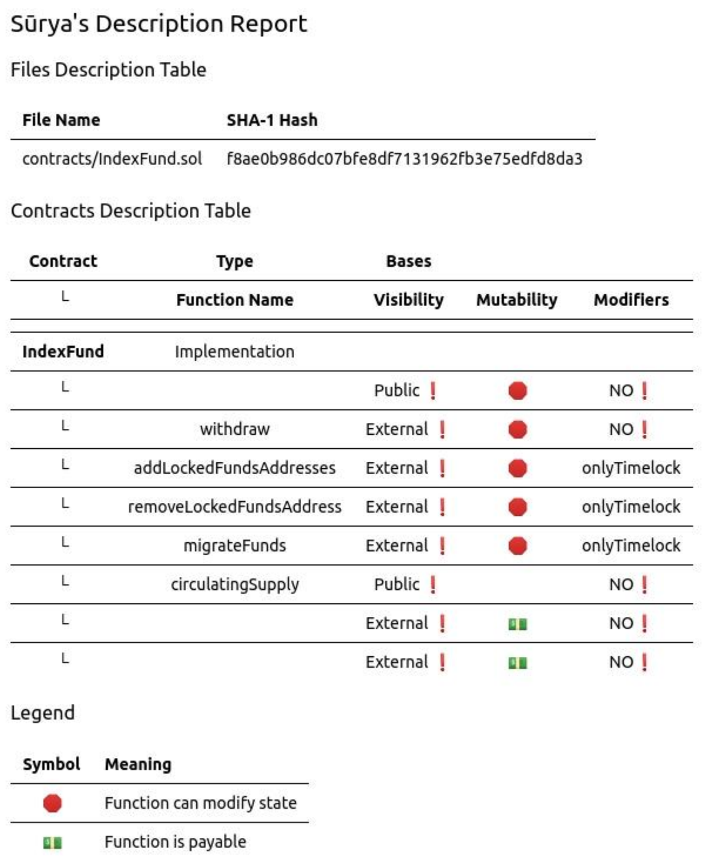
Project Overview

Index Fund serves as a repository for the fees earned by the Unipilot protocol, its primary purpose is to underpin the value of the $PILOT token.

System Architecture

* No System Architecture Diagram was provided *

There is a single smart contract for managing Index Fund. The main functionality of this smart contract is to enable withdrawal of different tokens by burning $PILOT.



Sūrya's Description Report

Files Description Table

File Name	SHA-1 Hash
contracts/IndexFund.sol	f8ae0b986dc07bfe8df7131962fb3e75edfd8da3

Contracts Description Table

Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
IndexFund	Implementation			
└		Public ❗	🛑	NO ❗
└	withdraw	External ❗	🛑	NO ❗
└	addLockedFundsAddresses	External ❗	🛑	onlyTimelock
└	removeLockedFundsAddress	External ❗	🛑	onlyTimelock
└	migrateFunds	External ❗	🛑	onlyTimelock
└	circulatingSupply	Public ❗		NO ❗
└		External ❗	💵	NO ❗
└		External ❗	💵	NO ❗

Legend

Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

Manual Review

The smart contract code is very straightforward. The errors reported by slither were verified manually. We also found some coding style issues and gas optimization suggestions, given in the description below.

Test Cases

The test cases provided by the Unipilot's dev team were too basic to adequately verify the functionality of the index fund code. The test cases can be found here.

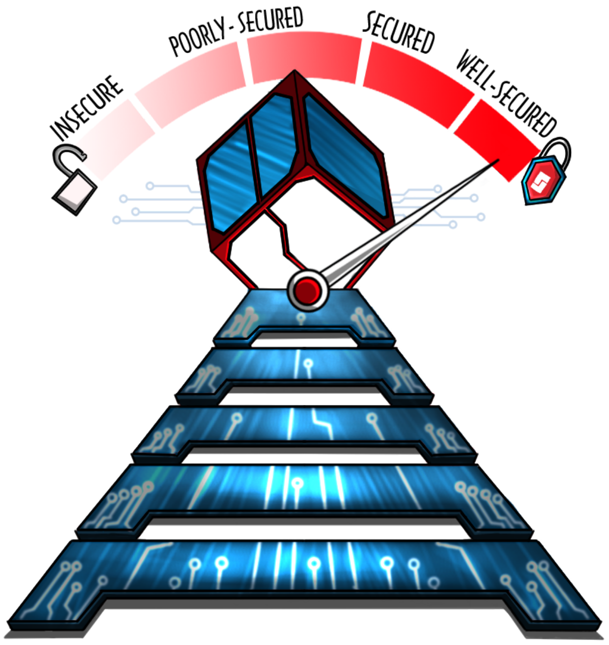
We have written our own test cases to verify the intended behaviour. Following are the results of the test cases.

AUDIT REPORT

Executive Summary

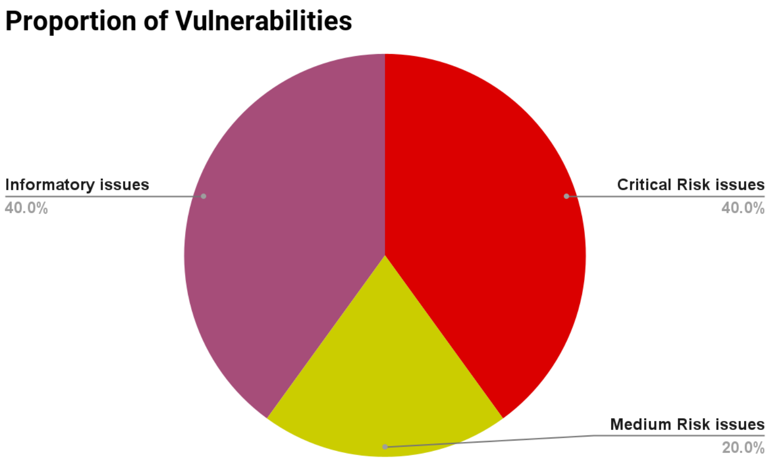
The analysis indicates that the contracts audited are **well-secured.**

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. All the flags raised were manually reviewed and re-tested.



Our team found:

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



Findings

Critical-risk issues

Issue #1

The withdraw function accepts an array of token addresses and an amount of $PILOT. The function will loop on tokens array and transfer the tokens proportionally(amonut_of_$PILOT/circulating_supply_$PILOT * token_balance_of_index_fund). In essence, if **index_fund** has 1000 $USDT and 1000 $USDC and the user holds 10% of $PILOT supply he/she will be entitled to 10% of both tokens(USDT and USDC). But this function can be exploited to deplete all the funds from **index_fund**.

Attack Scenario:

User XXX holds 20% of $PILOT and calls the withdraw function by passing the same token address multiple times. The user will be able to get more than 20%.

```
const test  = await IndexFundContract.withdraw(
[contractUsdt.address,contractUsdt.address,contractUsdt.address,cont
ractUsdt.address,contractUni.address],web3.utils.toWei('2000000'))
```

Remedy:

Duplicate addresses shouldn't be allowed in token arrays or If token[i] is already transferred, do not transfer if you encounter the token[i] has already been transferred.

Dev Response:

They acknowledged it and have implemented the correct logic. The new logic has been verified in the latest commit.

Issue #2

The migrateFunds function is supposed to migrate funds to the new version of the contract. This function is protected by timeLock, but this function fails on transferring the funds. It uses the safeTransferFrom function while transferring the funds to the new version. safeTransferFrom needs approval before transferring the function even if the sender is the owner of tokens. We tested that with some major tokens like $USDT and $LINK (it fails).

Remedy:

Use safeTransfer of SafeERC20 instead of safeTransferFrom.

Dev Response:

They have acknowledged it and applied the fix.

High-risk issues

No High-Risk issues were found.

Medium-risk issues

Issue #1

We raised a concern related to user Experience that if the **index_fund** has no balance of any token passed in the token addresses array while calling the withdraw function. The **index_fund** simply burns $PILOT.

Dev Response:

They appreciated the suggestion and added a check, if the token balance of the contract is zero, then revert the transaction.

Low-risk issues

No Low-Risk issues were found.

Informatory issues

1. **Gas cost Optimization**:

   Use calldata instead of memory while passing the args to `withdraw`, `migrateFunds addLockedFundsAddresses`
2. Cross check the **PILOT_ADDRESS** value, as it's hardcoded.

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.