BlockApex

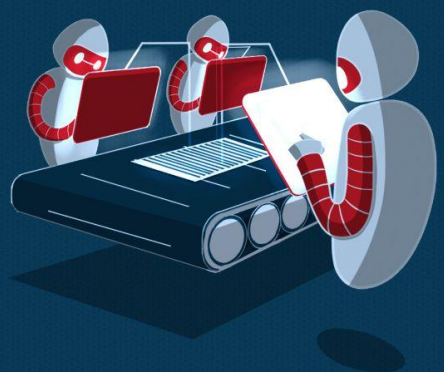
SMART CONTRACT SECURITY ANALYSIS REPORT

```
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

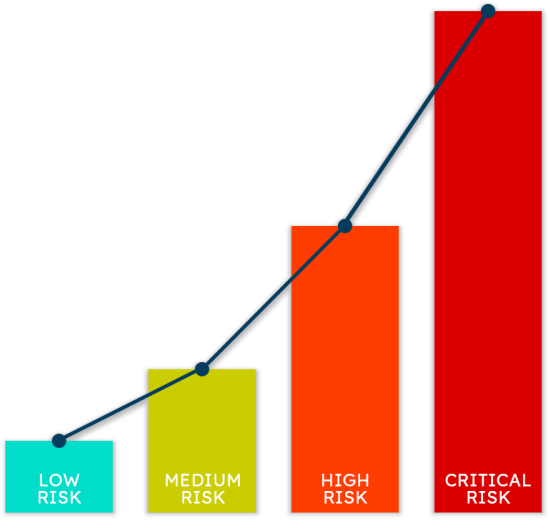
PREFACE

Objectives

The purpose of this document is to highlight 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 ××××	Critical vulnerabilities are too easy to exploit and can lead to damages/loss in assets or manipulations.
HIGH RISK ×××	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution.
MEDIUM RISK ××	Medium-level vulnerabilities are equally imperative to fix but they tend to have minimal impact on asset loss or data manipulations.
LOW RISK ×	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 29th July 2021.

Name
Unipilot Liquidity Locker
Auditor
Moazzam Arif | Kaif Ahmed
Platform
Ethereum/Solidity
Type of review
Vesting
Methods
Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Git Repository
https://github.com/VoirStudio/unipilot-liquidity-locking-contract/tree/cb0903504fed27273fcedd91c6d4b6e488190b02
White paper/ Documentation
Not required
Document log
Day 1: 29-07-2021 (Initial Audit)

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

Most of the projects provide initial liquidity on DEXEs. They lock their liquidity for a specific period of time to ensure trust in the community. There are some locking projects like Unicrypt, but they work with uniswapV2/ERC20 based liquidity. There is no project to lock uniswap v3 liquidity NFT. Unipilot's ull (unipilot liquidity locking) is an attempt to fill this gap. It is used to lock uniswapV3 liquidity NFTs.

System Architecture

No architecture diagram was provided

Unipilot's Liquidity Locking contract imitates the idea of OpenZeppelin's vesting contract. This contract locks the liquidity for startTime + duration amount of time (Note: The startTime can be in the past). It also allows to claim fee on locked liquidity (that can also be turned off).

It has the following major functions: (fetched by using an automated tool called **Slither**)

```
function lockLPToken

function claimLPFee

function updateOwner

function updateFeeReciever

function renounceBeneficiaryUpdate

function unlockToken
```

Manual Review

As important functions like `lockLPToken`, `claimLPFee`, `unlockToken` are simply wrappers around uniswap v3 NonFungiblePositionManager's contract with some time checks, so Fee amounts to be claimed are not tested and are considered out of scope.

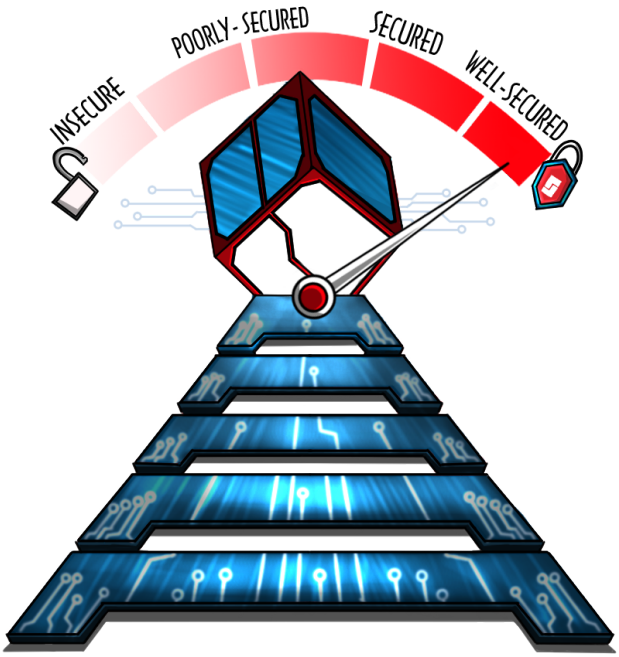
Test Cases

No test cases were provided initially by the team. We wrote our own hardhat tests to verify the contract behaviour. We modified some of the interface to speed up our tests. Link for tests is here.
Major test case scenarios revolved around the locking mechanism of the Uniswap's LP NFT. A detailed report of the test case scenarios can be found here.

AUDIT REPORT

Executive Summary

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

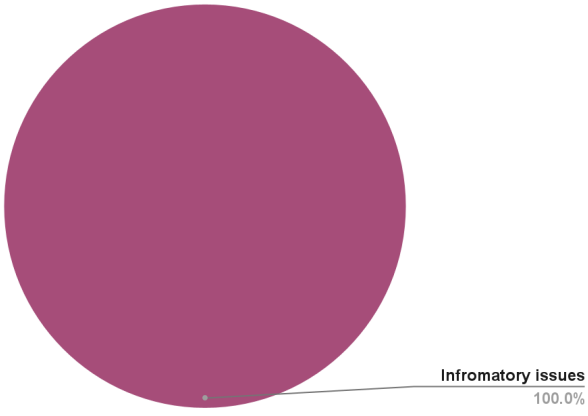


Our team performed a technique called “Filtered Audit”, where the contract was separately audited by two individuals. After their thorough and rigorous process of manual testing, an automated review was carried out using Slither. All the flags raised were manually reviewed and re-tested.

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)
2	Informatory issue(s)

Proportion of Vulnerabilities



Findings

Critical-risk issues

No critical issues were found.

High-risk issues

No high-risk issues were found.

Medium-risk issues

No medium-risk issues were found.

Low-risk issues

No low-risk issues were found.

Informatory issues

1. **Floating pragma versions**

   Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

   Remediation:
   Lock the pragma version and also consider known bugs for the compiler version that is chosen. Please refer to this doc for more details.

2. start (unixTimeStamp) should not be allowed to assign a value of the past (previous to the current timestamp). It is misleading alongside the **duration** variable.

   Devs Response:

   This is intended to suit their scenario (i.e., start+duration is the time limit).

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.