BlockApex

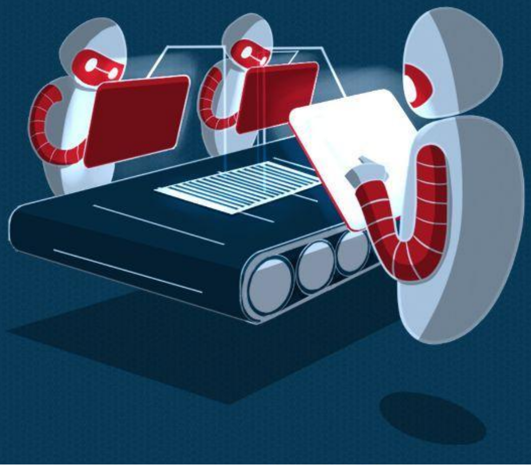
SMART CONTRACT SECURITY ANALYSIS REPORT

```
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";
    }
}
```



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

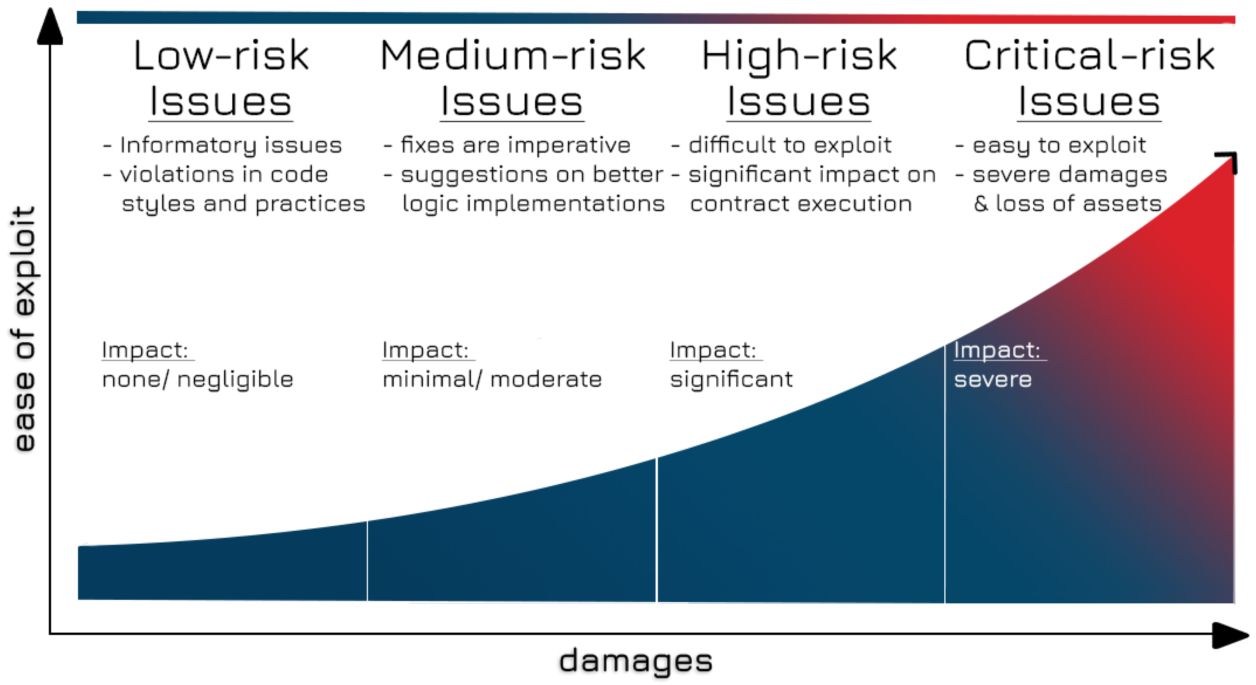


TABLE OF CONTENTS

INTRODUCTION

BlockApex (Auditor) was contracted by Borderless (Client) for the purpose of conducting a Smart Contract Audit/Code Review. This document presents the findings of our analysis which started on 14th September, 2022.

Name
Brrderless Protocol
Auditor
BlockApex
Platform
Polygon | Solidity
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/dedmonkes/ded-social-programs
White paper/ Documentation
https://www.ded.social
Document log
Initial Audit Completed: July 30th, 2022
Final Audit Completed: August 19th, 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

Phase Protocol is a NFT Marketplace infrastructure built on Solana Protocol, a reliable and scalable L1 solution. The on-chain Fundraising solution offered by DedMonke provides a crowdfunding experience to DeFi users.

Phase Protocol, founded by DedMonke is an innovative escrow service that incentivizes credible projects teams for the community approved roadmaps and deliverables. They are a type of fundraising opportunity through the use of NFTs.

System Architecture

Fundraising Through NFTs
With the use of Phase protocol, development teams can easily raise funds for their projects by minting NFTs for their project deliverable. They are one of the finest examples of how NFTs can be used in different ways.

Treasury Control
All the funds are properly controlled by the treasury to assure the community about the transparency of the system and how much funds are currently allocated by the team.

Fund Redistribution
In any event of failure the funds will be redistributed back to the current holders. There will also be a option to create a DAO in the near future.

Methodology & Scope

The codebase was audited using a filtered audit technique. A band of four (4) auditors scanned the codebase in an iterative process spanning over a time of two (2) weeks.

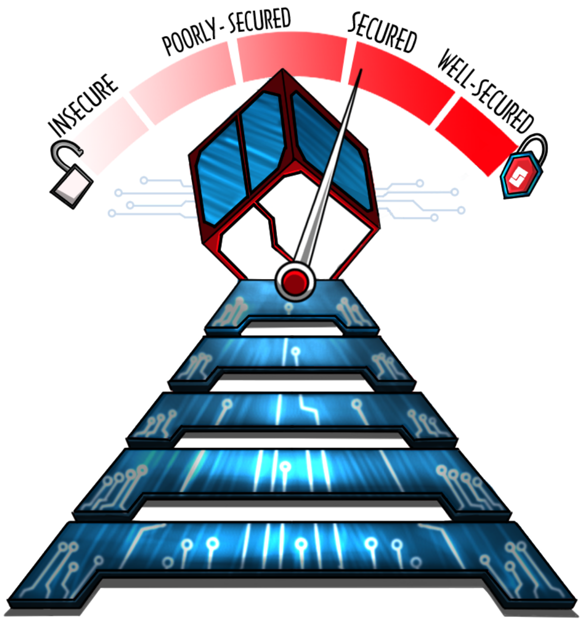
Starting with the recon phase, a basic understanding was developed and the auditors worked on developing presumptions for the developed codebase and the relevant documentation/whitepaper. Furthermore, the audit moved on with the manual code reviews with the motive to find logical flaws in the codebase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

AUDIT REPORT

Executive Summary

The analysis indicates that the contracts under scope of audit are **working properly** excluding swap functionality which contains one recent issue.

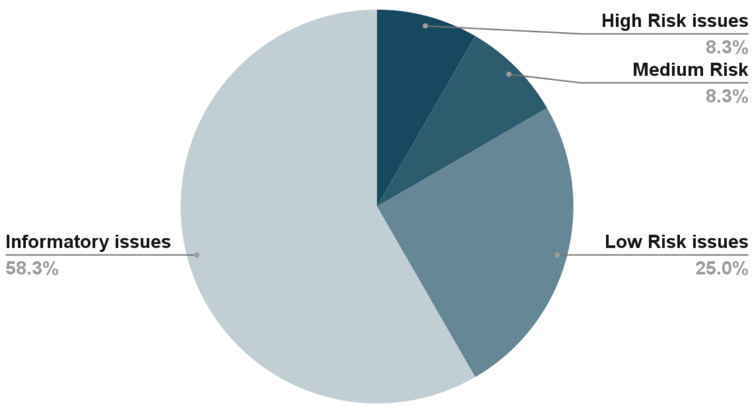
Our team performed a technique called "Filtered Audit", where the contract was separately audited by four individuals. After a thorough and rigorous process of manual testing, an automated review was carried out using cargo-audit & cargo-tarpaulin for static analysis and cargo-fuzz for fuzzing invariants. 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)
1	High Risk issue(s)
1	Medium Risk issue(s)
2	Low Risk issue(s)
2	Informatory issue(s)

Proportion of Vulnerabilities



Key Findings

#	Findings	Risk	Status
1.	Lack of Proper Checks Could Potentially Lead To Bypass When Changing Phases	High	Fixed
2.	Too Much Usage Of Helper Methods	Medium	Fixed
3.	Use Of Insecure Math In Arithmetic	Low	Fixed
4.	Use Of Outdated And Vulnerable Crates	Low	Fixed
6.	Inexistent math Overflow Checks	Informatory	Fixed
7.	Low Test Coverage	Informatory	Acknowledge

Detailed Overview

Critical-risk issues

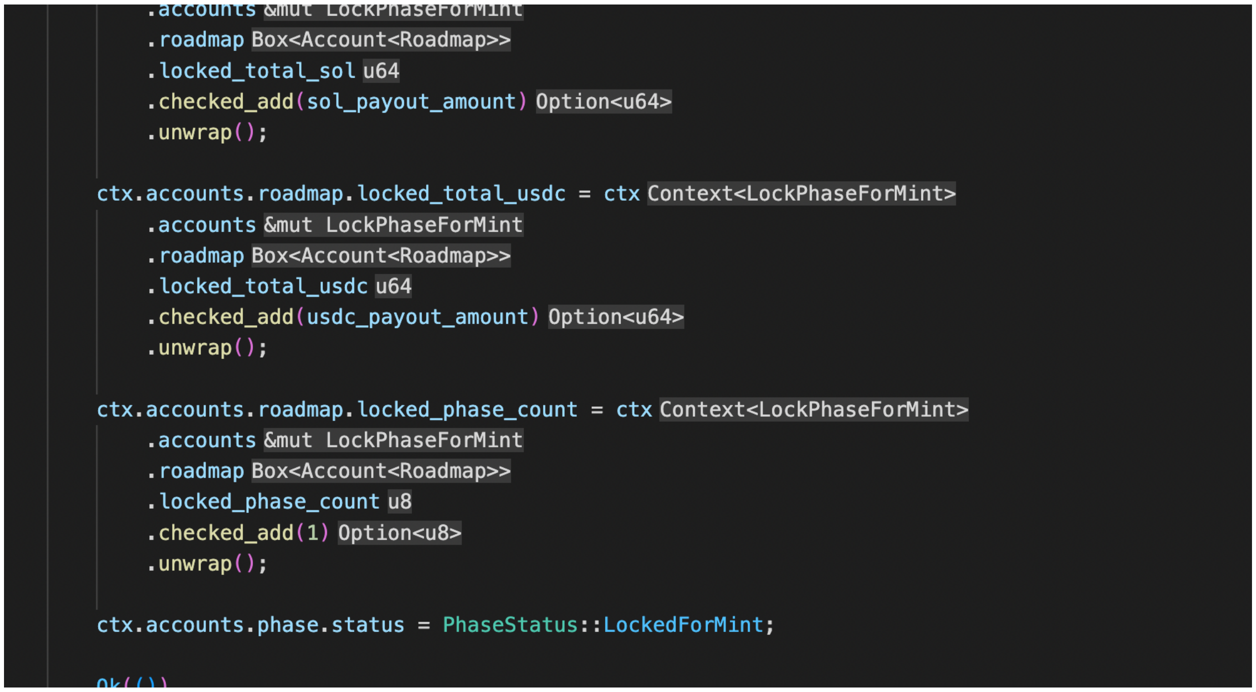
No issues were found.

High-risk issues

1. **Lack Of Proper Checks Could Potentially Lead to Bypass When Changing Phases**

   File: In the programs/phase-protocol/src/instructions/phase

   Description:
   In the contracts for different phases under the given directory, when changing the state from one phase into another there is not any proper conditional verification and statements to check if the phase is even eligible to change the state. In the contract lock_phase_for_mint.rs only the phases that are approved could be locked, and the phases in the draft cannot be locked. Although the protection will not be bypassed in every scenario, there is a risk that this lack of checks could cause issues in the future.



```
            .accounts &mut LockPhaseForMint
            .roadmap Box<Account<Roadmap>>
            .locked_total_sol u64
            .checked_add(sol_payout_amount) Option<u64>
            .unwrap();

        ctx.accounts.roadmap.locked_total_usdc = ctx Context<LockPhaseForMint>
            .accounts &mut LockPhaseForMint
            .roadmap Box<Account<Roadmap>>
            .locked_total_usdc u64
            .checked_add(usdc_payout_amount) Option<u64>
            .unwrap();

        ctx.accounts.roadmap.locked_phase_count = ctx Context<LockPhaseForMint>
            .accounts &mut LockPhaseForMint
            .roadmap Box<Account<Roadmap>>
            .locked_phase_count u8
            .checked_add(1) Option<u8>
            .unwrap();

        ctx.accounts.phase.status = PhaseStatus::LockedForMint;

        Ok(())
```

Remedy:
According to the documentation provided the phases usually go through the following phases DRAFT —> APPROVED —> ACTIONED —> STAGED FOR COMPLETION —> COMPLETE.
Now before moving on to the next phase it should check whether the provided phase is even in the phase that is required. As stated above parameters type will usually stop these types of bypasses but adding a proper layer of checks is always good to keep everything properly secured, because optimization is not such an issue in Solana.

```
        assert!(
            availability.allow_place || availability.allow_cancel,
            ""
        );
```

Status:
Fixed

Dev Response:
Added in explicit state transition checks

Medium-risk issues

2. Too Much Usage Of The Helper Methods.

File: In the programs/phase-protocol/src

Description:

There is a lot of usage of helper function **.unwrap()** inside every contract in the given directory. Although helper methods like **.unwrap()** are extremely helpful during the development and testing phase, making use of these functions in a production environment is an extremely bad practice that should be avoided because this usually causes the program to panic! and does not even show any helpful messages to the user to help solve or understand the problem.

Remedy:
Some usage of **.unwrap()** is justified. Proper conditional statements or **Some**/ **None** should be made use of because they are more safe and secure.

Status:
Fixed

Dev Response:
Added Error handling and messages for client

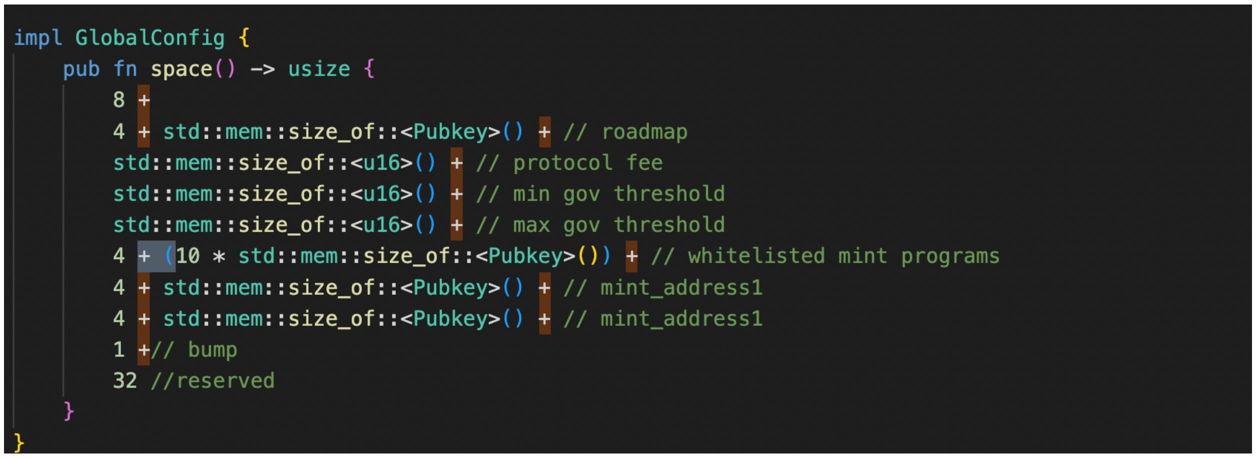
Low-risk issues

3. Use of Insecure Math In Arithmetic Operations

File: In the programs

Description:

Overflow/ Underflow usually happens when the result of any arithmetic operation is outside the range of datatype. There is one example attached below, but it is recommended to use checked_mul(), checked_add() and checked_sub() in all the complex arithmetic operations because they make sure that the variables don't overflow or underflow. The code below is from programs/phase-protocol/src/state/global_config.rs



```rust
impl GlobalConfig {
    pub fn space() -> usize {
        8 +
        4 + std::mem::size_of::<Pubkey>() + // roadmap
        std::mem::size_of::<u16>() + // protocol fee
        std::mem::size_of::<u16>() + // min gov threshold
        std::mem::size_of::<u16>() + // max gov threshold
        4 + (10 * std::mem::size_of::<Pubkey>()) + // whitelisted mint programs
        4 + std::mem::size_of::<Pubkey>() + // mint_address1
        4 + std::mem::size_of::<Pubkey>() + // mint_address1
        1 +// bump
        32 //reserved
    }
}
```

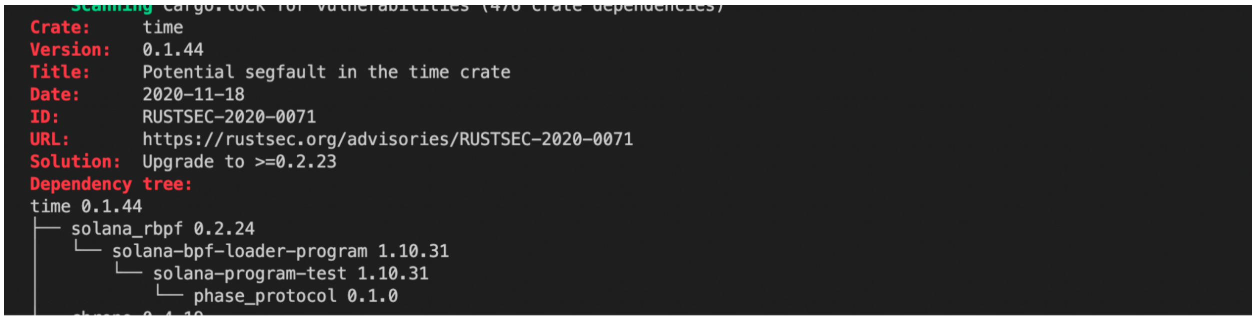
Remedy:
checked_mul(), **checked_add()** and **checked_sub()** in all arithmetic operations in the codebase.

Status:
Fixed

Dev Response:
Added in a max length constraint to any dynamic sized pda's. The rest are static sizes so there is not overflow issues from user input

4. Use Of Outdated And Vulnerable Crates

Description:
We used the **`cargo audit`** to test and detect any outdated and vulnerable crates that are used by the phase protocol and we found out that the contract is utilizing the outdated version of **`time`** crate. The version in use is **0.1.44** while the latest version is **0.2.23**.



```
Crate:     time
Version:   0.1.44
Title:     Potential segfault in the time crate
Date:      2020-11-18
ID:        RUSTSEC-2020-0071
URL:       https://rustsec.org/advisories/RUSTSEC-2020-0071
Solution:  Upgrade to >=0.2.23
Dependency tree:
time 0.1.44
├── solana_rbpf 0.2.24
│   └── solana-bpf-loader-program 1.10.31
│       └── solana-program-test 1.10.31
│           └── phase_protocol 0.1.0
```

Remedy:
We recommend that the given crate should be upgraded to the latest version in order to protect the contracts from any malicious actions. Moreover the old versions are usually less optimized when compared to newer releases.

Status:
Fixed

Dev Response:
Updated to latest version of solana-sdk to solve issues. But dep potential seg fault issues still exist in the latest versions

Informatory issues and Optimizations

5. **Inexistent math overflows checks**

   Description:
   In the overall code, there is a lack of underflow and overflow checks inside the cargo.toml.

   Remedy:
   In all the cargo.toml files, underflow and overflow checks should be defined.

   Status:
   Fixed

   Dev Response:
   "overflow-checks = true" added to cargo.toml to ensure overflow checks are in release

6. **Low Test Coverage**

   File: In the programs/phase-protocol/tests

   Description:
   The test coverage provided in the tests directory is low, although it covers most of the functionalities but it does not contain the negative test cases. Negative test cases are equally important as compared to positive test cases. The test coverage can be found via the command **`cargo tarpaulin`**.

   Moreover the test coverage provided contains the positive test cases and it does not have enough negative tests to fully test it out.

   Remedy:
   Writing proper test scenarios for each of the access points and functionality is among the best practices that must be included.

Status:
Acknowledged

Dev Response:
This will be worked on further in coming months but not currently able to be prioritised

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.