BlockÂpex

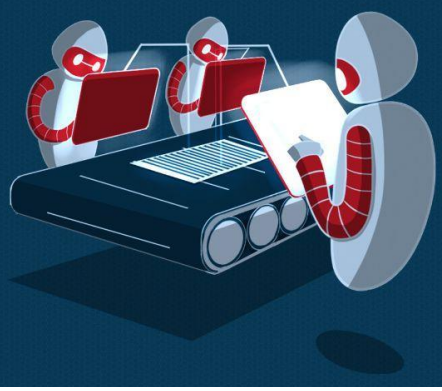
SMART CONTRACT SECURITY ANALYSIS REPORT

```
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";
    }
}
```



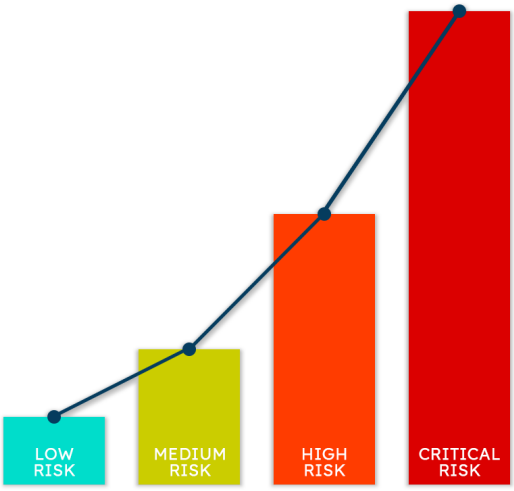
PREFACE

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.

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.

TABLE OF CONTENTS

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.

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.

INTRODUCTION

BlockApex (Auditor) was contracted by Dafi Protocol (Client) for the purpose of conducting a Smart Contract Audit/Code Review. This document presents the findings of our analysis which took place from 16th Dec 2021 to 14th Jan 2022 .

Name
Dafi Staking V2
Auditor
Moazzam Arif | Muhammad Jarir Uddin
Platform
Ethereum/Solidity
Type of review
Staking, Mathematics, Oracle feeds
Methods
Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Git repository
https://github.com/DAFIProtocol/dDAFI/tree/main/contracts/V2
White paper/ Documentation
https://docs.dafiprotocol.io/super-staking/overview-super-staking
Document log
Initial Audit: 31st December 2021 (complete)
Formal verification using property-based testing: 15th January 2021
Final Audit: 17th January 2022 (complete)

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.

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.

Project Overview

A comprehensive explanation of Staking module V2 for the Dafi Protocol is present in the official documentation and can be viewed at:
https://docs.dafiprotocol.io/super-staking/super-staking-v2

Dafi Protocol Super Staking V2 is an update to the V1 module released earlier this year (2021) in the month of July. Super Staking V2 claims to offer a more stable APY rate and enhanced distribution of dDAFI rewards by modifying the math behind reward calculation to rather depend only on accumulated amounts of reward every time demand factor changes to calculating the rewards of users as the sum of all rewards in the past adjusted to the latest demand factor.

Aside from the new reward formula, V2 also holds a couple of security improvements where demand factor is now enumerated using the latest price and is fortified by introduction of a delay i.e. a variance tolerance mechanism to ultimately prevent a sudden change in price. This new demand factor is supported by a TWAP calculation to make the price curve less steeper. Although strict monitoring is required, in cases of hackable oracle feeds, the protocol is able to recover from any kind of exploits by updating to an entirely new oracle service.

System Architecture

Codebase:

The system consists of 5 main smart contracts (namely: Staking Manager V2, Staking Database, Rebase Engine, Network Demand, Token Pool) and supplied with external data through 2 more contracts of Price Feeds and TVL Feeds.

Note: All of the contracts mentioned above contain ***onlyOwner*** modifiers to add, set and/or update configurations.

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.

SHA-256 fingerprints of Contracts included:

- **Staking Manager V2 :** ./StakingManagerV2.sol
  759bb0b2eb3f3d18b1d835f1b1b89733613ff6d1
- **Staking Database :** ./StakingDatabase.sol
  e4bb62c5ac3cb28c6f875af06158c4bff448d7410f84e147e8de08fcd72667d0
- **Rebase Engine :** ./rebase engine/RebaseEngine.sol
  61ab614f761737a42134cdaef991a0f7d5dd7ca1244993b2ba8bb277217432c4
- **TokenPool :** ./TokenPool.sol
  62c376c91256035f1601be81f9ce07d069acd1ccae3c00f330507f8618aac986
- **Network Demand :** ./network demand/NetworkDemand.sol
  bf71534344959ba076b03b0ac83f34a0ac20fa2ed2986bd7d85889e2b56fbb30
- **Price Feeds :** ./network demand/PriceFeeds.sol
  75b13b5e14cd516aaf65ac82526e7a1c317eebb57d306d6a0bbff7033317c5c7
- **TVL Feeds :** ./network demand/TVLFeeds.sol
  119130bd33ef19f8e901e7c81f6d3089d7c8df8f98586c2c1881c71021dcded2

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.

Static-Analysis summary

```
'npx hardhat compile --force' running
Compiling 30 files with 0.8.0
Generating typings for: 31 artifacts in dir: typechain for target: ethers-v5
Successfully generated 56 typings!
Compilation finished successfully


Compiled with Builder
Number of lines: 1945 (+ 291 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 26 (+ 5 in dependencies, + 0 tests)

Number of optimization issues: 10
Number of informational issues: 109
Number of low issues: 14
Number of medium issues: 10
Number of high issues: 15

Use: Openzeppelin-Ownable
ERCs: ERC20

+------------------+-------------+------+------------+--------------+--------------------+
|       Name       | # functions | ERCS | ERC20 info | Complex code |      Features      |
+------------------+-------------+------+------------+--------------+--------------------+
| StakingDatabase  |      51     |      |            |      No      |                    |
| StakingManagerV1 |      38     |      |            |      No      | Tokens interaction |
|    TokenPool     |      13     |      |            |      No      | Tokens interaction |
| StakingDatabase  |      53     |      |            |      No      |                    |
| StakingManagerV2 |      37     |      |            |      No      | Tokens interaction |
|    TokenPool     |      13     |      |            |      No      | Tokens interaction |
|  NetworkDemand   |      34     |      |            |     Yes      |                    |
|    PriceFeeds    |      3      |      |            |      No      |                    |
|     TVLFeeds     |      11     |      |            |      No      |                    |
|   RebaseEngine   |      19     |      |            |      No      |                    |
|    IDIAOracle    |      1      |      |            |      No      |                    |
|   DIAPriceFeed   |      3      |      |            |      No      |                    |
|  NetworkDemand   |      25     |      |            |      No      |                    |
|    PriceFeeds    |      3      |      |            |      No      |                    |
|     TVLFeeds     |      11     |      |            |      No      |                    |
|   RebaseEngine   |      19     |      |            |      No      |                    |
+------------------+-------------+------+------------+--------------+--------------------+
. analyzed (31 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.

Methodology & Scope

Audit log

Manual Review: The audit launched with a recon phase where a manual code review was conducted to clarify the layers of understanding of the complexities and the general flow of the program. We started by reviewing the two main contracts against common solidity flaws. After the reconnaissance phase we wrote unit-test cases to ensure that the functions are performing their intended behavior. Then we began with the line-by-line manual code review.

Property Testing: From the reconnaissance, a handful of properties were also extracted and labeled as Invariants. In the following days of the audit procedure, the invariants were thoroughly tested against a setup flow of Dafi Staking V2 to ensure each of them held its proper definition.

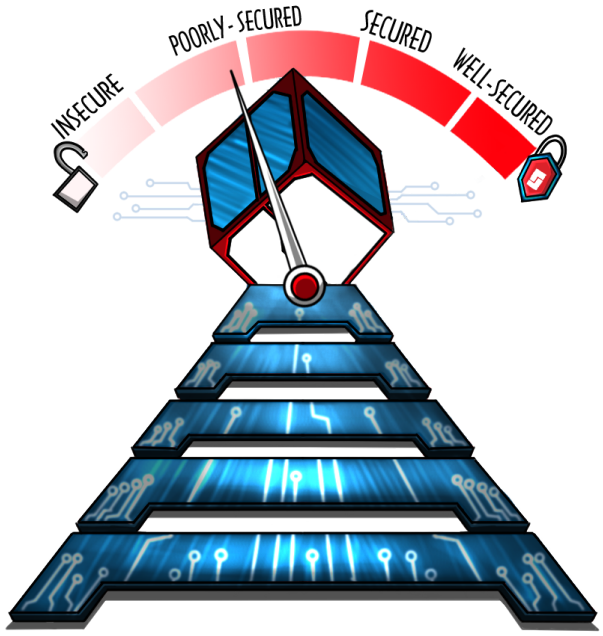
The contents of this document are proprietary and highly confidential. Information from this document should not be extracted/disclosed in any form to a third party without the prior written consent of BlockApex.

AUDIT REPORT

Executive Summary

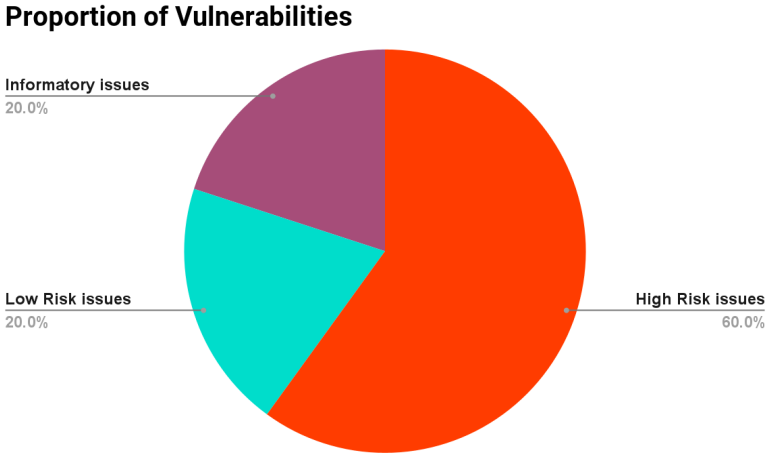
The analysis indicates that some of the functionalities in the contracts audited are **poorly-secured.**

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 Hardhat, Dapptools, Echidna and 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)
3	High Risk issue(s)
0	Medium Risk issue(s)
1	Low Risk issue(s)
1	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.

Findings

Below is the summary of our findings from the complete audit. This includes any flags raised from the manual/automated code review, behavioral/scenario testing and the properties tested for formal verification.

S#	Properties/Findings	Risk/Impact	Status
CR-1.	Oracle integrity	**Critical**	Passed
HR-1.	MAX_DAFI always greater than totaldDafiDistributed	**High**	Failed
HR-2.	RewardPool should not have balance if the Program is ended and all users have unstaked	**High**	Failed
HR-3.	If a program has ended, users should not be able to stake	**High**	Failed
LR-1.	Reward is claimable even after program has ended	**Low**	Passed
LR-2.	Unit tests in the testing and main branch of repo at /test/v2.ts should pass	**Low**	Failed
IR-1.	Memory optimization by using uint8 in place of true/ false	**Informatory**	-

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.

Critical-risk

CR-1. Oracle Integrity

Status : Passed

Description

In recent events, Oracles and external data sources have been manipulated to the adversary's benefit. We started off with our focus on the integrity of Oracles (Chainlink in this case) in order to make sure that the data imported into the contracts are not compromised.

The Chainlink oracle library method was tested against the following parameters of a standard audit technique.

- Third party (data source) function calls
- Data type intact for functions' return values
- Access modifiers and state variables (if any)

```
23      /**
24       * Returns the latest price
25       */
         ftrace | funcSig
26      function getThePrice() public view override returns (uint) {
27          (
28          ,
29          int price,
30          ,
31          ,
32          ) = priceFeed.latestRoundData();
33          return uint(price);
34      }
35  }
```

Suggestion: It is advised that a more tested and bonafide data source of TWAPs be considered to make the Chainlink's oracle integrity and reliability as the most optimal.

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.

High-risk

HR-1. MAX_DAFI is always greater than totaldDafiDistributed

Status : Failed

Description

If a program is marked as “ended” before the preset ending duration, that is to say, the elapsed time of the program is less than the program duration, users get revert transactions for staking into the ended program (as seen in the testnet event and confirmed over a fuzzing test scenario with the reason of MAX_DAFI being calculated less than total DAFI distributed till the ending timestamp). This property does not hold at fuzzed inputs;

```
function test_ConfirmMaxDafiOverflowFuzz(
    uint256 md,
    uint32 pd,
    uint32 bn
) public {
    SetupContracts caller = dapping.users(0).setupContract();

    caller.wrapSetStakingParams(1, md + 1, pd + 1);

    hevm.warp(block.timestamp + bn);

    caller.wrapRebasePool();
    caller.database().getPool();

    uint256 MaxDafi = caller.rebaseEngine().MAX_DAFI();
    uint256 TdDafiDist = caller.rebaseEngine().totaldDAFIDistributed();

    assertGt(MaxDafi, TdDafiDist, 'maxDafi < tdDafiDist; program ended; dDafi overflows');

    emit log named uint('value of MaxDafi', MaxDafi);
    emit log named uint('value of TdDafiDist', TdDafiDist);
}
```

Suggestion: It is advised that program duration should always be maintained greater than the time elapsed by supplying checks that assure it.

```
if (database.isProgramEnded() && pool.lastUpdatedOn > database.getProgramEndedAt()) {
    return;
} else if (database.isProgramEnded() && pool.lastUpdatedOn < database.getProgramEndedAt()) {
    maxTimestampForCalc = database.getProgramEndedAt();
} else {
    maxTimestampForCalc = block.timestamp;
}
```

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.

HR-2. Reward Pool should not have balance if the program has ended and all users have unstaked

Status : Failed

Description

This property claims that the reward pool should be empty after the program has ended and that all the users have unstaked.

HR-2(a). "markProgramEnded()" by owner's mistake can lock funds of the reward pool

In case of human error if the program is marked ended the reward pool retains an amount of tokens unclaimed against the percentage of tokens staked in first place.

```
ftrace | funcSig
function test_RewardPoolUnemptyAfterProgramEnded() public {
    SetupContracts caller = dapping.users(0).setupContract();
    hevm.warp(block.timestamp + 5000);


    caller.wrapSetStakingParams(1, 1000 ether, 12);

    uint256 distPoolBalanceBefore = caller.stakingManager().distributionPool().balance();

    hevm.warp(block.timestamp + 90000);
    caller.wrapStake(100);

    caller.wrapMarkProgramEnded();

    hevm.warp(block.timestamp + 90000);
    caller.wrapUnstake(100);

    uint256 distPoolBalanceAfter = caller.stakingManager().distributionPool().balance();

    assertGt(distPoolBalanceBefore, distPoolBalanceAfter);

    emit log_named_uint('poolBefore', distPoolBalanceBefore);
    emit log_named_uint('poolAfter', distPoolBalanceAfter);
}
```

```
poolBefore: 100000000000000000000
poolAfter: 991366859243697481900
```

Suggestion: It is advised to have a refund mechanism if program duration is ended. The refund amount of tokens should be equal to *MAX_DAFI - (MDI*totalElapsedTime)*.

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.

HR-3. If a program has ended, users should not be able to stake

Status : Failed

Description

In a simple scenario test, it was confirmed that Staking was allowed even if the program duration is completed and the program is marked as ended which in the understanding of the Auditor team is an (incomprehensible feature) and can be proved to be a loophole of the system in some scenarios.

Suggestion: The modifier *stakeCheck* should be supplied with some additional check to ensure monitoring of programs marked as ended not to allow staking.

```
ftrace | funcSig
function test_StakeAfterProgramEnded() public {
    SetupContracts caller = dapping.users(0).setupContract();
    hevm.warp(block.timestamp + 500);

    // _ms, _md, _pd
    uint256 minStakeDays = 1;
    uint256 maxDafi = 1000 ether;
    uint32 progDuration = 12;

    caller.wrapSetStakingParams(minStakeDays, maxDafi, progDuration);

    // stakin/ unstaing multiple times for seeding
    for (uint256 i = 0; i < 3; i++) {
        hevm.warp(block.timestamp + 90000);
        caller.wrapStake(1000000000);
        hevm.warp(block.timestamp + 90000);
        caller.wrapUnstake(1000000000);
    }

    caller.wrapMarkProgramEnded();

    // stakin/ unstaing multiple times for seeding
    for (uint256 i = 0; i < 3; i++) {
        hevm.warp(block.timestamp + 90000);
        caller.wrapStake(1000000000);
        hevm.warp(block.timestamp + 90000);
        caller.wrapUnstake(1000000000);
    }

    assertTrue(false);
}
```

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.

Low-risk

LR-1. Reward is claimable even after program is ended

Status : Passed

Description

Regardless of the program being ended, users should be able to claim their rewards for their staked dafi tokens. This claim should not be bound by any type of time-related constraints. We checked whether this property would fail in any circumstance but it passed on all fuzzed inputs.

LR-2. Unit tests in the testing and main branch of repo at /test/v2.ts should pass

Status : Failed

Description

Unit tests are critical in proving the developer's expected intention and behavior of the working code hence the set of tests not passing entirely and partially in the testing repository code is slightly questionable.

Informatory issues and Optimizations

IR-1. Memory optimization by using uint8 in place of true/false

Description

Following the best practices and the Solidity design patterns guide and since the client code uses a good number of state variables to manage the switches for staking, unstaking and alike. It is therefore suggested by the auditing team that uint8 type variables can be replaced by the developer in place of true/false on multiple occasions to minimize the memory usage and reduce the code size as an optimization.

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.

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.