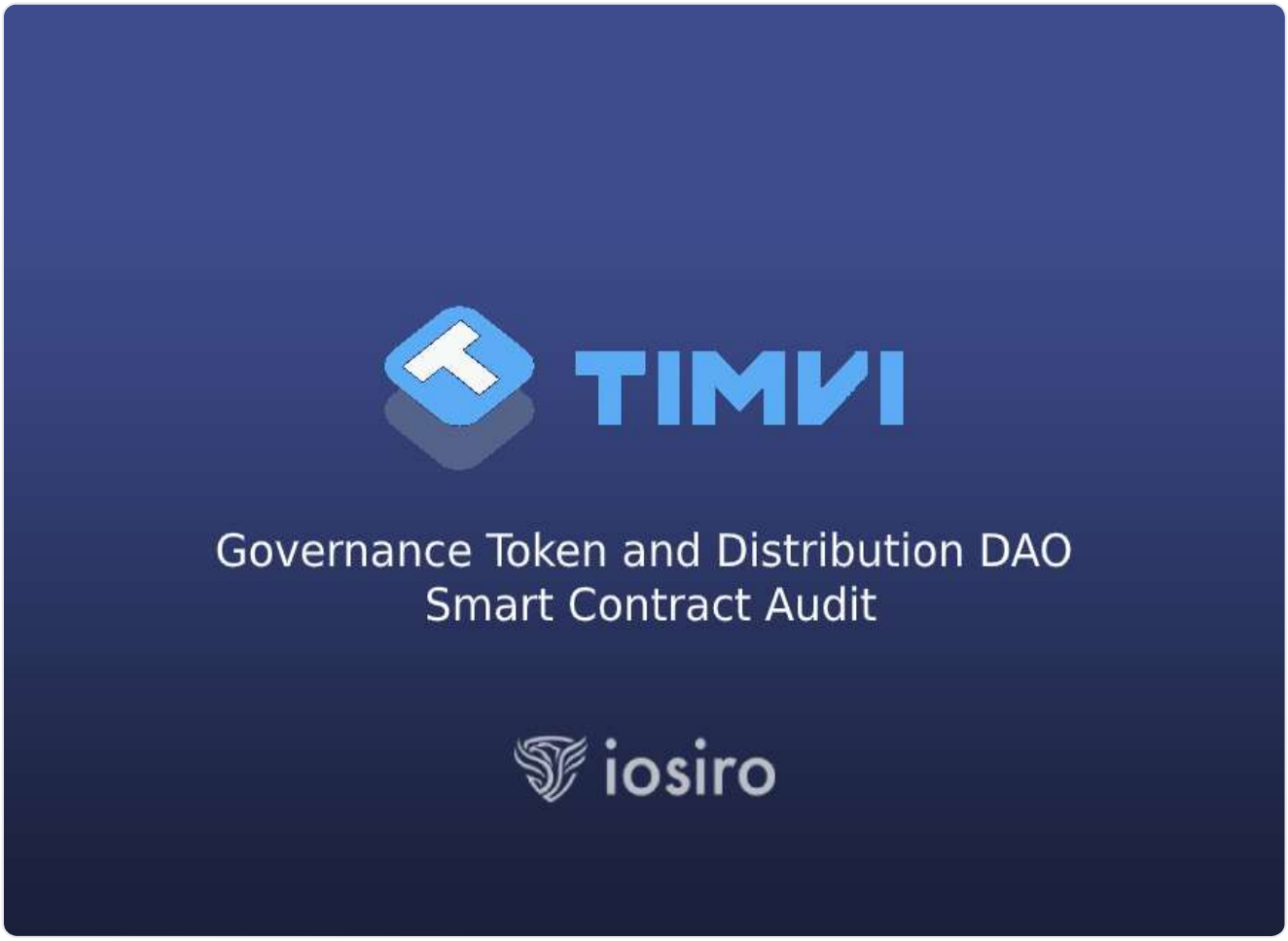
Timvi Governance Token and Distribution DAO Smart Contract Audit



1. Introduction

iosiro was commissioned by Timvi to conduct a smart contract audit on the Timvi Governance Token and reward distribution pool. The audit was performed between 21 and 25 August 2020. A review of remediations made by Timvi to address the audit findings was conducted on 28 August and 3 September 2020.

This report is organized into the following sections.

- **Section 2 - Executive Summary:** A high-level description of the findings of the audit.
- **Section 3 - Audit Details:** A description of the scope and methodology of the audit.
- **Section 4 - Design Specification:** An outline of the intended functionality of the smart contracts.

- **Section 5 - Detailed Findings:** Detailed descriptions of the findings of the audit.

The information in this report should be used to better understand the risk exposure of the smart contracts, and as a guide to improving the security posture of the smart contracts by remediating issues identified. The results of this audit are only a reflection of the source code reviewed at the time of the audit and of the source code that was determined to be in-scope.

The purpose of this audit was to achieve the following:

- Identify potential security flaws.
- Ensure that the smart contracts functioned according to the documentation provided.

Assessing the off-chain functionality associated with the contracts, for example, backend web application code, was out of scope of this audit.

Due to the unregulated nature and ease of transfer of cryptocurrencies, operations that store or interact with these assets are considered very high risk with regards to cyber attacks. As such, the highest level of security should be observed when interacting with these assets. This requires a forward-thinking approach, which takes into account the new and experimental nature of blockchain technologies. Strategies that should be used to encourage secure code development include:

- Security should be integrated into the development lifecycle and the level of perceived security should not be limited to a single code audit.
- Defensive programming should be employed to account for unforeseen circumstances.
- Current best practices should be followed where possible.

2. Executive Summary

This report presents the findings of an audit performed by iosiro on the Timvi Distribution Pool and Governance Token (GTMV). The GTMV token will be used by the Distribution Pool as a reward token.

Audit Findings

Both the Distribution Pool and Governance Token were largely based on existing code with minor additions and changes. All findings were remediated by Timvi during the audit process.

User Considerations

Prospective users of these contracts should keep the following in mind:

- The Timvi Governance Token's supply is fixed at the initial 300 million tokens. More tokens cannot be minted.
- The Timvi Distribution Pool includes functionality that will allow Timvi to distribute extra tokens (that are not either the pool or reward token) which come into the contract's possession according to their discretion, and possibly outside of the pool's contributors.

More detail is provided in 4. Design Specification below.

Recommendations

At a high level, the security posture of the Timvi Distribution Pool and Governance Token could be further strengthened by:

- Remediating the issues identified in this report and performing a review to ensure that the issues were correctly addressed. Remediation should include writing unit tests to cover remedial additions to contract code.
- Performing additional audits at regular intervals, as security best practices, tools, and knowledge change over time. Additional audits over the course of the project's lifespan ensure the longevity of the codebase.
- Creating a bug bounty program to encourage the responsible disclosure of security vulnerabilities in the system.

3. Audit Details

3.1 Scope

The source code considered in-scope for the assessment is described below. Code from all other files is considered to be out-of-scope. Out-of-scope code that interacts with in-scope code is assumed to function as intended and introduce no functional or security vulnerabilities for the purposes of this audit.

3.1.1 Smart Contracts

- **Project Name:** Timvi Governance
- **Commits:** Initial audit: a6e59; Review changes: 7704e, 4c564eb
- **Files:**

```
├── helpers
│   ├── Address.sol
│   ├── Context.sol
│   ├── IERC20.sol
│   ├── IRewardDistributionRecipient.sol
│   ├── Math.sol
│   ├── Ownable.sol
│   ├── SafeERC20.sol
│   └── SafeMath.sol
├── mocks
│   ├── LPTokenMock.sol
│   ├── RandomTokenMock.sol
│   └── TimviGovernanceTokenMock.sol
├── Distribution.sol
├── Migrations.sol
└── TimviGovernanceToken.sol
```

3.2 Methodology

A variety of techniques, described below, were used to conduct the audit.

3.2.1 Code Review

The source code was manually inspected to identify potential security flaws. Code review is a useful approach for detecting security flaws, discrepancies between the

specification and implementation, design improvements, and high risk areas of the system.

3.2.2 Dynamic Analysis

The contracts were compiled, deployed, and tested in a Ganache test environment, both manually and through the Truffle test suite provided. Manual analysis was used to confirm that the code operated at a functional level and to verify the exploitability of any potential security issues identified. The coverage report of the provided Truffle tests as on the final day of the audit is given below.

189-194,207,209

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	93.18	73.33	93.75	93.26	
Distribution.sol	100	66.67	100	100	
TimviGovernanceToken.sol	85.37	83.33	88.24	85.37	189-194,207,209
All files	93.18	73.33	93.75	93.26	

While testing was incomplete for the file `TimviGovernanceToken.sol`, the untested lines were found to be the ERC20 contract's internal burn functions, which were unreachable as the token did not implement external burn functionality.

3.2.3 Automated Analysis

Tools were used to automatically detect the presence of several types of security vulnerabilities, including reentrancy, timestamp dependency bugs, and transaction-ordering dependency bugs. The static analysis results were manually analyzed to remove false-positive results. True positive results would be indicated in this report.

Static analysis tools commonly used include Slither, Securify, and MythX. Tools such as the Remix IDE, compilation output, and linters could also be used to identify potential areas of concern.

3.3 Risk Ratings

Each issue identified during the audit has been assigned a risk rating. The rating is determined based on the criteria outlined below.

- **High Risk** - The issue could result in a loss of funds for the contract owner or system users.
- **Medium Risk** - The issue resulted in the code specification being implemented incorrectly.
- **Low Risk** - A best practice or design issue that could affect the security of the contract.
- **Informational** - A lapse in best practice or a suboptimal design pattern that has a minimal risk of affecting the security of the contract.
- **Closed** - The issue was identified during the audit and has since been addressed to a satisfactory level to remove the risk that it posed.

4. Design Specification

The following section outlines the intended functionality of the system at a high level. The specification is based on the implementation in the codebase and any perceived points of conflict should be highlighted with the auditing team to determine the source of the discrepancy.

4.1 Timvi Distribution Pool

The Timvi distribution pool contract allows users to provide currency in `poolToken` to the contract in exchange for periodic rewards in `rewardToken`. `poolToken` and `rewardToken` can each be any ERC-20 token, and may be the same token. These tokens are defined on deployment.

The Timvi distribution pool contract was based on a deprecated version of Synthetix's StakingRewards contract, called Unipool, which has been audited by SigmaPrime and subsequently used as a basis for YFI, YAM, PASTA and other staking pools.

In addition to minor name changes of functions and events, two notable functionality changes compared to the Unipool functionality were observed.

First, the Timvi distribution pool did not hardcode the pool and reward token addresses in the contract, but allowed them to be defined in the constructor. This

change is in line with more recent versions of StakingRewards and other contracts derived from Unipool. It is important to note that once the contract is deployed, the owners cannot change the reward or pool token.

Second, the Timvi distribution pool includes an additional function, `distributeAdditionalRewards(...)`, which will allow the Timvi-designated rewards distribution address to distribute arbitrary amounts of an ERC-20 token that is *not* the reward or pool token between arbitrary addresses. These amounts and addresses will be at Timvi's discretion, and are not required to be participants in the pool.

4.2 Timvi Governance Token

The Timvi Governance Token is an ERC-20 token based on the OpenZeppelin `ERC20` implementation. It uses OpenZeppelin's `ERC20Detailed` and no other mixins.

Field	Value
Symbol	GTMV
Name	Timvi Governance Token
Decimals	18
Initial Supply	300 million

As the token does not implement the `ERC20Mintable` mixin, more tokens cannot be minted. Upon deployment, all 300 million GTMV tokens are sent to the deployer address.

5. Detailed Findings

The following section details the findings of the audit.

5.1 High Risk

No high risk issues were present at the conclusion of the review.

5.2 Medium Risk

No medium risk issues were present at the conclusion of the review.

5.3 Low Risk

No low risk issues were present at the conclusion of the review.

5.4 Informational

No informational issues were present at the conclusion of the review.

5.5 Closed

5.5.1 Reward Amount Overflow (High Risk)

Distribution.notifyRewardAmount(...)

In July 2020, an overflow bug was discovered in the Synthetix StakingRewards contract. The bug is detailed in this Github gist. Timvi's Distribution contract was found to contain the same issue.

Remedial Action

The recommended remediation for this bug is to require the calculated reward amount to be lower than the contract's remaining balance of the reward token. This could be done by adding the below lines just above the line which reads

```
lastUpdatedTime = now;

uint balance = rewardToken.balanceOf(address(this));
require(rewardRate <= balance.div(DURATION), "Provided reward too high");
```

Timvi Update

Fixed in 7704e4a.

5.5.2 Floating Pragma Version (Low Risk)

General

The pragma statement included in the smart contracts allowed the codebase to be compiled with any version of Solidity ^0.6.0, meaning the highest non-breaking version of 0.6.x. This could lead to a situation where the contracts are compiled using a compiler version they have not been extensively tested on, or which contains undiscovered bugs.

Additionally, one contract (`helpers/Address.sol`) required any version of Solidity ^0.6.2, which was inconsistent with the rest of the project.

Remedial Action

To increase certainty and reliability, it was recommended that all contracts are locked to a single, uniform compiler version. iosiro recommends version 0.6.12. This can be done by changing the pragma statement at the top of each contract to the following:

```
pragma solidity 0.6.12;
```

Timvi Update

Fixed in 7704e4a.

5.5.3 Insufficient Unit Test Quality and Coverage (Informational)

Distribution.notifyRewardAmount(...)

Certain functionality was found to have insufficient unit test quality and coverage. To improve the maintainability of the code, and decrease the likelihood of introducing functional or security issues into the codebase, it is recommended that the test suite be extended to cover lines 133-135 of `Distribution.sol` , in the `notifyRewardAmount(...)` .

Test coverage was also incomplete for `TimviGovernanceToken.sol` , but the affected lines were unreachable code that formed part of a standard ERC-20 library.

Timvi Update

Fixed in 4c564eb.

5.5.4 Design Comments (Informational)

Actions to improve the functionality and readability of the codebase are outlined below.

Use of `now`

General

In Solidity version 0.7.0, the `now` keyword was deprecated. Developers are encouraged to use `block.timestamp` instead, to ensure forward compatibility.

Timvi Update

Fixed in 7704e4a.

Lack of array length check

Distribution.distributeAdditionalRewards(...)

The `distributeAdditionalRewards(...)` function took an array of user addresses and an array of reward amounts as arguments. These arrays were used in a for loop in a manner that assumed they would always be the same length, but this was not validated.

Should the users list be longer than the reward amounts list, the function would fail with an invalid opcode error. Should the users list be shorter than the reward amounts list, the function would complete successfully, but may have unexpected results.

To prevent these error states, an additional require statement similar to the below could be included in the function, increasing its gas cost by approximately 23:

```
require(users.length == rewardAmounts.length, "users and rewardAmounts are different
```

As this function is only executable by Timvi's rewards distribution address, this remediation may not be necessary as long as sufficient caution is taken each time the function is run.

Timvi Update

Fixed in 7704e4a.

Secure your system.

Request a service

START NOW →