PROTOCOL SECURITY AUDIT REPORT

TETHYS FINANCE

Decurity, 2021

CONTENTS

1 GENERAL INFORMATION

This report contains information about the results of the security audit of the Tethys Finance smart contracts, conducted by Decurity (DeFiSecurity.io) in the period from 12/23/21 to 12/26/21.

1.1 Introduction

Tasks solved during the work are:

- Review the protocol design and the deviations from the original contracts (SushiSwap, Uniswap, etc),
- Audit the contracts implementation,
- Develop the recommendations and suggestions to improve the security of the contracts.

1.2 Scope of Work

The audit scope included the following contracts:

- Factory (UniswapV2Factory and UniswapV2Pair): https://andromeda-explorer.metis.io/address/0x2CdFB20205701FF01689461610C9F321D1d00F80/contracts
- xTETHYS (TethysTemple): https://andromeda-explorer.metis.io/address/0x716678968fd6E518cb1d56C9720fC8eeEBA6CeAb/contracts

- MasterChef: https://andromeda-explorer.metis.io/address/0x54A8fB8c634dED694D270b78Cb931cA6bF241E21/contracts.

1.3 Threat Model

The assessment presumes actions of an intruder who might have capabilities of any role (an external user, token owner, token service owner, a contract).

The main possible threat actors are:

- User,
- Protocol owner,
- Liquidity Token owner/contract.

The table below contains sample attacks that malicious attackers might carry out.

Table 1. Theoretically possible attacks

Attack	Actor
Contract code or data hijacking *Deploying a malicious contract or submitting malicious data*	Contract owner Token owner
Financial fraud *A malicious manipulation of the business logic and balances, such as a re-entrancy attack or a flash loan attack*	Anyone
Attacks on implementation	Anyone

Attack	Actor
Exploiting the weaknesses in the compiler or the runtime of the smart contracts	

1.4 Weakness Scoring

An expert evaluation scores the findings in this report, an impact of each vulnerability is calculated based on its ease of exploitation (based on the industry practice and our experience) and severity (for the considered threats).

2 SUMMARY

As a result of this work, we have analyzed the contract. As a result, we haven't discovered any exploitable security issues.

The Tethys team has given the feedback for the suggested changes and explanation for the underlying code.

2.1 Suggestions

The table below contains the discovered issues, their risk level, and their status as of 12/26/2021.

Table 2. Discovered weaknesses

Issue	Risk Level	Probability	Status
No sanity check in getMultiplier in MasterChef	Low	Medium	Accepted
Insufficient event emission	Low	Low	Accepted
Insufficient sanity checks	Medium	Low	The Tethys team is going to be careful when changing the dev address or updating the liquidity pool
Possible gas savings	Low	Low	Accepted
Possible funds drain via the deflationary tokens in MasterChef	High	Low	The Tethys team is only going to use the Uniswap Pair tokens

3 GENERAL RECOMMENDATIONS

This section contains general recommendations how to fix discovered during the testing weaknesses and vulnerabilities and how to improve overall security level.

Section 3.1 contains a list of general mitigations against the discovered weaknesses, technical recommendations for each finding can be found in section 4.

Section 3.2 describes a brief long-term action plan to mitigate further weaknesses and bring the product security to a higher level.

3.1 Current findings remediation

- Fix the issues described in the section 4.2.
- Perform review and verification of all the business scenarios of the contracts,
- Test the production environment of the off-chain service when deployed.

3.2 Security process improvement

- Keep the whitepaper and documentation updated to make it consistent with the implementation and the intended use cases of the system,
- Launch a public bug bounty campaign for the contracts.

4 DETAILS

4.1 Code deviations

During the audit, we've performed differential analysis of the contracts code against the original implemention by Uniswap.

As a result, the following changes have been detected:

1) Everywhere:
    a. Solidity version updated,
2) UniswapV2Pair.sol:
    a. Fee denominator coefficient changed from 5 to 3,
    b. The adjusted balances calculation changed (needed for the Uniswap K invariant check).
3) UniswapV2Factory.sol:
    a. *pairCodeHash* function added,
    b. override modifier added to the functions.
4) UniswapV2ERC20.sol:
    a. Name and symbol changed
    b. *chainid* changed to *chainid().*
5) MasterChef.sol: (similar to spookytoken, popsicle finance, etc)
    a. Variable names changed,
    b. *withdrawalTime* variable added,
    c. *maxTethysPerSecond* and *MaxAllocPoint* variables added,
    d. *startTime* used instead of *startBlock*,
    e. *_bonusEndBlock* variable removed,
    f. *setTethysPerSecond* variable added,

g. *getMultiplierByWithdrawalTime* function added,
h. *checkForDuplicate* function added,
i. *_withUpdate* flag removed from the *add* and *set* functions,
j. Migration mechanism removed,
k. *user.amount > 0* check removed from deposit

The key security improvements compared to the original implementations are:

1) The migration mechanism removal: migration was a direct way to perform a rug pull,
2) The duplication check added: such sanity checks provide additional level of protection against a human factor,
3) The optional *_withUpdate* flag removed: while this might lead to additional gas costs for the pool token updates, this also excludes a malicious scenario where the contract owners dilute the pool without prior reward update.

4.2 Findings

4.2.1 No sanity check in getMultiplier in MasterChef

Summary:

The *getMultiplier* method of the *MasterChef* contract checks that the _to argument is not less than the *startTime* value but does not check that it's not less than *_from*.

It's possible that under certain conditions someone (a user or another contract) would call this function with incorrect arguments and wouldn't get a meaningfull error message.

Remediation:

Add a check that *_to* is not less than *_from*.

4.2.2 Insufficient event emission

Summary:

Some of the important functions such as *add*, *set*, and *setTethysPerSecond* in *MasterChef* or *setFeeTo* and *setFeeToSetter* in *UniswapV2Factory* don't emit any events.

Remediation:

Add an event emission whenever an important change to the contract state is made.

4.2.3 Insufficient sanity checks

Summary:

Some of the functions such as *dev*, *safeSushiTransfer* *setTethysPerSecond* in *MasterChef* or *setFeeTo* and *setFeeToSetter* in *UniswapV2Factory* don't perform zero address checks.

Additionally, the pool manipulation functions in *MasterChef* don't check that the pool corresponding to the specified id actually exists.

Remediation:

Add a check that the input address is not zero or the input pool id is valid.

4.2.4 Possible gas savings

Summary:

The functions intended only for the external use such as *enter* and *leave* in *TethysTemple* or *deposit* and *withdraw* in *MasterChef* could be explicitly marked with the *external* modifier to save some gas for the users.

Remediation:

Add a corresponding modifier to the external functions.

4.2.5 Possible funds drain via the deflationary tokens in MasterChef

Summary:

As it already happened with several projects, the protocol funds theft is possible in case the owners add deflationary tokens to the *MasterChef* liquidity pool.

Remediation:

Rework the *MasterChef deposit* and *withdraw* mechanisms to support the deflationary tokens.

5 APPENDIX

5.1 About us

The Decurity (former DeFiSecurity.io) team consists of experienced hackers who have been doing application security assessments and penetration testing for over a decade.

During the recent years, we've gained an expertise in blockchain field and have conducted numerous audits for both centralized and decentralized projects: exchanges, protocols, and blockchain nodes.

Our efforts have helped to protect hundreds of millions of dollars and make web3 a safer place.