Multiplier Finance “TAKO” Public Report

PROJECT: Multiplier Finance TAKO Review

September 2020

Prepared For:

Kim Hui | Multiplier Finance

business@multiplier.finance

Prepared By:

Jonathan Haas | Bramah Systems, LLC.

jonathan@bramah.systems

Table of Contents

Multiplier Finance Review

Executive Summary

Scope of Engagement

Bramah Systems, LLC was engaged in September of 2020 to perform a comprehensive security review of multiple Multiplier Finance smart contracts (specific contracts denoted below). Our review was conducted over a period of three business days by a member of the Bramah Systems, LLC. executive staff.

Bramah's review pertains to smart contract Solidity code (*.sol) as of commit **2457c5f3b4387f75f86adcda75763c945e8c7c5a**, as per request of Multiplier Finance. Bramah Systems completed the assessment using manual, static and dynamic analysis techniques.

Timeline

Review Commencement: September 24, 2020

Report Delivery: September 27, 2020

Engagement Goals

The primary scope of the engagement was to evaluate and establish the overall security of the Multiplier Finance system, with a specific focus on trading actions. In specific, the engagement sought to answer the following questions:

- Is it possible for an attacker to steal or freeze tokens?
- Does the Solidity code match the specification as provided?
- Is there a way to interfere with the contract mechanisms?
- Are the arithmetic calculations trustworthy?

Protocol Specification

Very little in the way of unique documentation exists for the codebase, primarily due to the forked nature of the project from SushiSwap. Relevant risk discussion for these considerations exists within "Supply Chain Risk". Bramah was provided a summary of the relevant modifications, all of which made by the Multiplier team, which primarily concerned removing overall complexity from the smart contract. While an additional pair was added (supporting the

TAKO token), modifications beyond this were all feature reductive, thereby reducing the overall complexity of the protocol.

Overall Assessment

Bramah Systems was engaged to evaluate and identify multiple security concerns in the codebase of the Multiplier Finance protocol architecture. During the course of our engagement, Bramah Systems denoted numerous instances wherein the protocol deviated from established best practices and procedures of secure software development. With limited exceptions (as described in this report), these instances were a result of structural limitations of Solidity or directly attributed to usage of the SushiSwap smart contracts. While reviewed by multiple parties, SushiSwap development lacked many common traits of secure software development, as discussed in depth within “Supply Chain Risk”.

This report reinforces learnings and modifications since made to the SushiSwap series of contracts, which has since received much public scrutiny and visibility, presented in concert with the nature of Multiplier Finance modifications (decreasing the overall complexity by removing non-core elements of the protocol and thereby reducing overall attack surface). Additive modifications to the protocol beyond introduction of an additional pair were immaterial or resolving known issues in the smart contract.

While risk associated with using known vulnerable components is described in depth within this report, the “battle-tested” nature of the Sushi smart contracts aids in this protocol's overall testing and security. Potential attack scenarios against Sushi swap were replayed against these contracts and in each case saw identical execution patterns (save for variable renaming and functionality changes as discussed). The existence of previous security reviews also offers deeper insight into the overall SDLC of the contracts, which during our evaluation proved useful in determining which functionality changes could result in potential concern.

Disclaimer

As of the date of publication, the information provided in this report reflects the presently held, commercially reasonable understanding of Bramah Systems, LLC.'s knowledge of security patterns as they relate to the Multiplier Finance Protocol, with the understanding that distributed ledger technologies ("DLT") remain under frequent and continual development, and resultantly carry with them unknown technical risks and flaws. The scope of the review provided herein is limited solely to items denoted within "Scope of Engagement" and contained within "Directory Structure". The report does NOT cover, review, or opine upon security considerations unique to the Solidity compiler, tools used in the development of the protocol, or distributed ledger technologies themselves, or to any other matters not specifically covered in this report.

The contents of this report must NOT be construed as investment advice or advice of any other kind. This report does NOT have any bearing upon the potential economics of the Multiplier Finance protocol or any other relevant product, service or asset of Multiplier Finance or otherwise. This report is not and should not be relied upon by Multiplier Finance or any reader of this report as any form of financial, tax, legal, regulatory, or other advice.

To the full extent permissible by applicable law, Bramah Systems, LLC. disclaims all warranties, express or implied. The information in this report is provided "as is" without warranty, representation, or guarantee of any kind, including the accuracy of the information provided. Bramah Systems, LLC. makes no warranties, representations, or guarantees about the Multiplier Finance Protocol. Use of this report and/or any of the information provided herein is at the users sole risk, and Bramah Systems, LLC. hereby disclaims, and each user of this report hereby waives, releases, and holds Bramah Systems, LLC. harmless from, any and all liability, damage, expense, or harm (actual, threatened, or claimed) from such use.

Timeliness of Content

All content within this report is presented only as of the date published or indicated, to the commercially reasonable knowledge of Bramah Systems, LLC. as of such date, and may be superseded by subsequent events or for other reasons. The content contained within this report is subject to change without notice. Bramah Systems, LLC. does not guarantee or warrant the accuracy or timeliness of any of the content contained within this report, whether accessed through digital means or otherwise.

Bramah Systems, LLC. is not responsible for setting individual browser cache settings nor can it ensure any parties beyond those individuals directly listed within this report are receiving the most recent content as reasonably understood by Bramah Systems, LLC. as of the date this report is provided to such individuals.

General Recommendations

Best Practices & Solidity Development Guidelines

Usage of ABIEncoderV2

A majority of the contracts associated with the protocol make usage of an experimental Solidity version (**pragma experimental ABIEncoderV2**) which enables usage of the new ABI encoder. **ABIEncoderV2** allows for the usage of structs and arbitrarily nested arrays (such as **string[]** and **uint256[][]**) in function arguments and return values.

As no present non experimental version for these constructs exists, one must acknowledge the associated risk in utilizing non release-candidate ("RC") software. It is understood that software in the beta phase will generally have more bugs than completed software as well as speed/performance issues and may cause crashes or data loss.

Lack of consistency in Solidity version

Contracts within the repository lack a singular Solidity version. As structural changes may occur between these versions, we suggest consolidating to a singular cohesive version (especially in the case of). Notably, for multiple versions below, their version option is not

Versions used: **['0.6.12', '=0.6.12', '>=0.4.25<0.7.0', '>=0.5.0', '>=0.6.0', '>=0.6.2', '^0.6.0', '^0.6.2']**

Time considerations and usage of block.timestamp

Miners can affect block.timestamp for their benefits. Thus, one should not rely on the exact value of block.timestamp. As a result of such, **block.timestamp** and **now** should traditionally only be used within inequalities.

Multiple variables are set relying upon Solidity's inexact time system. As not every year equals 365 days and not every day has 24 hours because of leap seconds, Solidity's one day/week/year values are inexact. As leap seconds cannot be predicted, an exact calendar match would require updating by an external oracle.

Note, the direct comparison of these variables (e.g. comparing two times) within their respective functions poses additional concern, as discussed above (namely, a proper comparison may not be set). It is worth noting that this has downstream implications on calculations utilising this passage of time.

Using components with known vulnerabilities

Third party integrations present a significant risk if untrusted parties are involved, especially with forked codebases. While the general security posture of organisations Multiplier Finance has integrated with (and resultantly, built protocol integrations for) is quite high, this report (and present security analysis) cannot say for certain these integrations will be without flaw. The original SushiSwap protocol has known deployment considerations which present security risk. While the majority of these have been addressed within code changes, some exist primarily as operational security concerns -- and have not inherently been addressed. Privileged roles are natively a requirement of the current construction of the platform, so diligence is suggested given the known permissive nature of the smart contract owner role.

It is notable that all integrations have seen some form of security scrutiny (be it a bug bounty, security review, or security focused testing via the development team). That said, the scope of this review does not cover the security of these integrations beyond the protocol integrations themselves (e.g. this is not a comprehensive audit of openzeppelin contracts, which are used within, beyond validating that they are the newest and most recently audited version).

The previous audit performed by Peckshield should be evaluated against the Multiplier Finance codebase. In their audit, Peckshield denotes numerous areas of concern both explicitly denoted by the developer (such as "// XXX DO NOT add the same LP token more than once. Rewards will be messed up if you do.") as well as those not stated as material concern by the developer -- which exist primarily as attack vectors during deployment and migration. In particular, the code does not particularly handle deployment of additional functional components (such as adding a staking pool with an identical **_lpToken** to a previously existing staking pool). As there are a number of mitigating factors to successful execution of this vulnerability, not all of these factors were implemented by the SushiSwap team (and through adoption of the fork, the Multiplier Finance team). This held true for a number of known issues. It is Bramah's suggestion that the Multiplier Finance team evaluate their overall risk concerns. While functional areas of concern do exist, they are of traditionally low severity, likelihood, and impact -- many of which can be handled by offchain discussion and control (such as in the case of the staking pools). That noted, numerous other potential areas of concern listed by Peckshield are addressed within the Multiplier Finance codebase, as the codebase adopts changes from the time of fork following the Peckshield security assessment. This can be validated through the usage of tools like GitDiff, but also by comparing relevant merged commits to the primary repository, specifically commit hash **8ed4f7fde186682fac2cb6493ed1a94a7717eeb0**. While other security audits / reviews do exist online, aspects of their review either were covered by

Peckshield, or by the protocol complexity reduction introduced by the Multiplier Finance team.

With previous security reviews of the originating SushiSwap protocol in mind, Bramah wishes to raise any additional concerns that may exist with the execution and deployment of the forked protocol.

Functions can be marked external

Functions within a contract that the contract itself does not call should be marked external in order to optimize for gas costs. In our review, most functions within **MasterChef.sol** should consider this implementation after review of any instances in which the contract may call them.

Slither denotes the following functions as potential areas of concern:

- Migrator.migrate(IUniswapV2Pair) (Migrator.sol#26-44)
- MasterChef.setRewardPerBlock(uint256,bool) (MasterChef.sol#96-101)
- MasterChef.add(uint256,IERC20,bool) (MasterChef.sol#105-117)
- MasterChef.set(uint256,uint256,bool) (MasterChef.sol#120-126)
- MasterChef.setMigrator(IMigratorChef) (MasterChef.sol#129-131)
- MasterChef.migrate(uint256) (MasterChef.sol#134-143)
- MasterChef.deposit(uint256,uint256) (MasterChef.sol#186-202)
- MasterChef.withdraw(uint256,uint256) (MasterChef.sol#205-220)
- MasterChef.emergencyWithdraw(uint256) (MasterChef.sol#223-230)
- UniswapV2Router02.quote(uint256,uint256,uint256) (uniswapv2/UniswapV2Router02.sol#403-405)
- UniswapV2Router02.getAmountOut(uint256,uint256,uint256) (uniswapv2/UniswapV2Router02.sol#407-415)
- UniswapV2Router02.getAmountIn(uint256,uint256,uint256) (uniswapv2/UniswapV2Router02.sol#417-425)
- UniswapV2Router02.getAmountsOut(uint256,address[]) (uniswapv2/UniswapV2Router02.sol#427-435)
- UniswapV2Router02.getAmountsIn(uint256,address[]) (uniswapv2/UniswapV2Router02.sol#437-445)
- Timelock.setDelay(uint256) (Timelock.sol#51-58)
- Timelock.acceptAdmin() (Timelock.sol#60-66)
- Timelock.setPendingAdmin(address) (Timelock.sol#68-79)
- Timelock.queueTransaction(address,uint256,string,bytes,uint256) (Timelock.sol#81-90)
- Timelock.cancelTransaction(address,uint256,string,bytes,uint256) (Timelock.sol#92-99)

- Timelock.executeTransaction(address,uint256,string,bytes,uint256) (Timelock.sol#101-126)
- TakoToken.mint(address,uint256) (TakoToken.sol#11-14)
- Migrations.setCompleted(uint256) (Migrations.sol#16-18)

Typographical Errors

Minor typographical errors exist within the code comments, appearing to largely be intentional by the author. While they do not functionally change the code, they are superfluous content which may present concern to individuals not knowledgeable about the history of the protocol.

Multiplication before division can result in loss of precision

Integer math brings arithmetic considerations that should be kept in mind when structuring order of operations within the smart contract (e.g. division may truncate, such as is described within Solidity Design Patterns).

Multiplication before division can result in a loss of overall precision within the smart contract. MasterChef.pendingTako(uint256,address) (MasterChef.sol#146-157) and MasterChef.updatePool(uint256) (MasterChef.sol#168-183) both fail to adopt this practice, which should be evaluated by the team on a case by case basis.

Specific Recommendations

Unique to the Multiplier Finance Protocol

Changing reward per block may result in gas exhaustion

The **massUpdatePools** functionality may run out of gas when too many tokens are added at once. As of commit **4cdce3c3b0f93b342adbba4e0447d6013d9bf56f**, **massUpdatePools** may be invoked upon change of the Sushi reward per block, so gas costs should be taken into account upon changes.

Toolset Warnings

Unique to the Multiplier Finance Protocol

Overview

In addition to our manual review, our process involves utilizing concolic analysis and dynamic testing in order to perform additional verification of the presence security vulnerabilities. An additional part of this review phase consists of reviewing any automated unit testing frameworks that exist.
The following sections detail warnings generated by the automated tools and confirmation of false positives where applicable, in addition to findings generated through manual inspection.

Compilation warnings

Two warnings are presented at time of compilation, one with library code that may not allow for modifications and the other within a modifiable source file by the Multiplier team.

UniSwap library code located within **UniswapV2Router02.sol** upon compile time raises a warning that as the contract code size exceeds 24576 bytes (a limit introduced in Spurious Dragon), this contract may not be deployable on mainnet. The Multiplier team should consider enabling the optimizer (with a low "runs" value) or turning off revert strings.

Within the **contracts/Timelock.sol** file, on line 120, the usage of "**.value(...)**" is deprecated. The Multiplier team should Use "**{value: ...}**" instead.

Test coverage

The contract repository lacks unit test coverage throughout. This testing traditionally would provide a variety of unit tests which encompass the various operational stages of the contract. Presently, the Multiplier Finance protocol does not possess tests validating functionality and ensuring that certain behaviors (those relating to erroneous or overflow-prone input) do not see successful execution.

Static analysis coverage

The contract repository underwent heavy scrutiny with multiple static analysis agents, including:

- Securify
- MAIAN
- Mythril
- Oyente
- Slither

In each case, the team had either mitigated relevant concerns raised by each of these tools or provided adequate justification for the risk (e.g. inherent risk of using native Ethereum constructs such as **timestamp**).

Certain tools, like Oyenete, do not run on newer versions of Solidity. Where applicable, Bramah manually performed validation checks based upon our understanding of the tool. With the exception of known issues stemming from the usage of SushiSwap code and the singular issue denoted above, Bramah did not locate any instances of concern within the modified code suggested by static analysis covered by Oyente.

Other tools, such as Slither, we recommend including into a general CI/CD pipeline for usage (such as through the Crytic platform). Slither's extensive ability to replicate contract state was invaluable in overall testing, and Slither quickly denotes areas in which best practices were deviated from.

Multiple analysis platforms denoted a key risk of re-entrancy throughout the contracts, however, in each instance, the called contract was a trusted entity (e.g. another smart contract deployed by the development team). In these cases, as with all others, we suggest strong adherence to the checks-effects-interactions design paradigm.