YEARN JOINT STRATEGY SMART CONTRACT AUDIT

January 25, 2022

MixBytes()

CONTENTS

1. INTRODUCTION

1.1 DISCLAIMER

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of Yearn. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 SECURITY ASSESSMENT METHODOLOGY

A group of auditors are involved in the work on the audit who check the provided source code independently of each other in accordance with the methodology described below:

01 Project architecture review:
> Reviewing project documentation
> General code review
> Reverse research and study of the architecture of the code based on the source code only
> Mockup prototyping

Stage goal:
Building an independent view of the project's architecture and identifying logical flaws in the code.

02 Checking the code against the checklist of known vulnerabilities:
> Manual code check for vulnerabilities from the company's internal checklist
> The company's checklist is constantly updated based on the analysis of hacks, research and audit of the clients' code
> Checking with static analyzers (i.e Slither, Mythril, etc.)

Stage goal:
Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flashloan attacks, etc.)

03 Checking the code for compliance with the desired security model:
> Detailed study of the project documentation
> Examining contracts tests
> Examining comments in code
> Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit
> Exploits PoC development using Brownie

Stage goal:
Detection of inconsistencies with the desired model

04 Consolidation of interim auditor reports into a general one:
> Cross-check: each auditor reviews the reports of the others
> Discussion of the found issues by the auditors
> Formation of a general (merged) report

Stage goal:
Re-check all the problems for relevance and correctness of the threat level and provide the client with an interim report.

05 Bug fixing & re-check:
> Client fixes or comments on every issue
> Upon completion of the bug fixing, the auditors double-check each fix and set the statuses with a link to the fix

Stage goal:
Preparation of the final code version with all the fixes

06 Preparation of the final audit report and delivery to the customer.

Findings discovered during the audit are classified as follows:

FINDINGS SEVERITY BREAKDOWN

Level	Description	Required action
Critical	Bugs leading to assets theft, fund access locking, or any other loss funds to be transferred to any party	Immediate action to fix issue
Major	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.	Implement fix as soon as possible
Warning	Bugs that can break the intended contract logic or expose it to DoS attacks	Take into consideration and implement fix in certain period
Comment	Other issues and recommendations reported to/acknowledged by the team	Take into consideration

Based on the feedback received from the Customer's team regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The project team is aware of this finding. Recommendations for this finding are planned to be resolved in the future. This finding does not affect the overall safety of the project.
No issue	Finding does not affect the overall safety of the project and does not violate the logic of its work.

1.3 PROJECT OVERVIEW

`Yearn Finance` is a suite of products in Decentralized Finance (DeFi) that provides lending aggregation, yield generation, and insurance on the Ethereum blockchain. The protocol is maintained by various independent developers and is governed by YFI holders.`Hegic Joint strategy` is `Yearn Finance vault` strategy that combines liquidity from two different `Yearn vaults` to use it in the single strategy that provides liquidity to `SushiSwap` and use `Hegic` project to invest into `PUT` and `CALL` options.

1.4 PROJECT DASHBOARD

Client	Yearn
Audit name	Joint Strategy
Initial version	035fd2d1d7bf30e79093675fa64a61267c566f49
Final version	035fd2d1d7bf30e79093675fa64a61267c566f49
Date	December 20, 2021 - January 25, 2022
Auditors engaged	5 auditors

FILES LISTING

HegicJoint.sol	https://github.com/fp-crypto/joint-strategy/blob/035fd2d1d7bf30e79093675fa64a61267c566f49/contracts/HegicJoint.sol
Joint.sol	https://github.com/fp-crypto/joint-strategy/blob/035fd2d1d7bf30e79093675fa64a61267c566f49/contracts/Joint.sol
LPHedgingLib.sol	https://github.com/fp-crypto/joint-strategy/blob/035fd2d1d7bf30e79093675fa64a61267c566f49/contracts/LPHedgingLib.sol
ProviderStrategy.sol	https://github.com/fp-crypto/joint-strategy/blob/035fd2d1d7bf30e79093675fa64a61267c566f49/contracts/ProviderStrategy.sol
SushiJoint.sol	https://github.com/fp-crypto/joint-strategy/blob/035fd2d1d7bf30e79093675fa64a61267c566f49/contracts/SushiJoint.sol

FINDINGS SUMMARY

Level	Amount
Critical	0
Major	1
Warning	2
Comment	8

CONCLUSION

During the audit process, 1 major issue was found and confirmed by developers. This issue can cause temporary malfunction of the strategy, however, it cannot cause permanent loss of funds. Also, 2 warnings and 8 comments were found and confirmed. These issues do not make serious impact on security and perfomance.By the moment of the report issue, the developers have not fixed the above findings.

CONTRACTS DEPLOYMENT

- SushiJoint: 0x997f3e5cae4455cfd225b5e43d2382c7f6b7c6e4
- ProviderWeth: 0x26244f5b4A933C6e595665F08F8c76108195b755,
- ProviderUsdc(proxy): 0xb1bc173c2bcc98e8eede8af0443ace29b8fa2992

2. FINDINGS REPORT

2.1 CRITICAL

Not Found

2.2 MAJOR

MJR-1	Insecure detection of algorithm state
File	Joint.sol
Severity	Major
Status	Acknowledged

DESCRIPTION

It is insecure to rely on `balanceOf` here Joint.sol#L256-L262 because tokens can be transferred by attacker to fake state of contract and break the contract's logic. For example, attacker can transfer small amount of `pair` tokens or `stake` tokens to cause `revert` at `openPosition` until manual recovery procedures to be performed.

RECOMMENDATION

We recommend to avoid using `balanceOf()` as evidence of the contract state. Contract state variables can be used instead.

CLIENT'S COMMENTARY

We have reviewed the potential paths to manipulate the balance of methods and have not found any paths to permanently brick or induce aberrant behavior in the Joint or Provider contracts.

2.3 WARNING

WRN-1	Weak permissions at the `Joint` contract
File	Joint.sol
Severity	Warning
Status	Acknowledged

DESCRIPTION

The `migrateProvider` is using weak modifier `onlyProviders` that does not make difference between `providerA` and `providerB`
Joint.sol#L701
This flaw allows `providerA` to make migration as it was `providerB` and to steal tokens of other provider.

RECOMMENDATION

We recommend to use permission logic that makes difference between `providerA` and `providerB` instead of `onlyProviders` modifier

CLIENT'S COMMENTARY

For this pattern to be exploited, a malicious strategy would need to be approved by governance. In a future implementation, a check could be added, requiring that the provider being migrated from, is the caller.

WRN-2	Unability to liquidate position
File	ProviderStrategy.sol
Severity	Warning
Status	Acknowledged

DESCRIPTION

Unlike common practice of `yearn vault` strategies, this strategy does not implement ability to withdraw (liquidate) funds by user request, the corresponding function does not liquidate position actually ProviderStrategy.sol#L155

Any funds can be released from the strategy only by adjusting `debt ratio` that is generally available only to the `Yearn vault management`.

RECOMMENDATION

We recommend to put this strategy to the end of `Yearn vault` withdraw queue to minimize impact of unability to liquidate positions.

CLIENT'S COMMENTARY

This strategy is intended to be at the end of the withdraw queue and have a limited debt ratio.

2.4 COMMENT

CMT-1	Sandwich attacks to adding and removing liquidity
File	Joint.sol
Severity	Comment
Status	Acknowledged

DESCRIPTION

We just want to highlight that if add/remove liquidity transaction will be processed using mempool the sandwich attacks can be exploted.

RECOMMENDATION

We recommend to use private relay or implement slippage checks.

CLIENT'S COMMENTARY

This is known and slippage checks offer additional protection.

CMT-2	getCurrentPrice() can be view
File	LPHedgingLib.sol
Severity	Comment
Status	Acknowledged

DESCRIPTION

Function getCurrentPrice() doesn't change the state.
So, for safety and to save the gas we should make it as 'view'.
At line:
LPHedgingLib.sol#L68

RECOMMENDATION

We recommend to add modifier `view`.

CLIENT'S COMMENTARY

Next version will implement this recommendation.

CMT-3	Unused internal method `checkProviders()`
File	Joint.sol
Severity	Comment
Status	Acknowledged

DESCRIPTION

At line:
Joint.sol#L87

RECOMMENDATION

We recommend to use `checkProviders()` in modifier `onlyProviders` which duplicates its functionality.

CLIENT'S COMMENTARY

Next version will implement this recommendation.

CMT-4	Ineffective `adjustPosition()`
File	Joint.sol
Severity	Comment
Status	Acknowledged

DESCRIPTION

No need to call `openPosition()` if `wantBalance=0`

RECOMMENDATION

We recommend to place `openPosition()` into `if`. For example:

```
function adjustPosition(uint256 _debtOutstanding) internal override {
        if (emergencyExit || dontInvestWant()) {
           return;
        }

        // Using a push approach (instead of pull)
        uint256 wantBalance = balanceOfWant();
        if (wantBalance > 0) {
            want.transfer(joint, wantBalance);
            JointAPI(joint).openPosition();
        }
    }
```

CLIENT'S COMMENTARY

Next version will implement this recommendation.

CMT-5	Unused import `Math.sol`
File	HegicJoint.sol Joint.sol ProviderStrategy.sol LPHedgingLib.sol
Severity	Comment
Status	Acknowledged

DESCRIPTION

At lines:
HegicJoint.sol#L11
Joint.sol#L11
ProviderStrategy.sol#L13
LPHedgingLib.sol#L9

RECOMMENDATION

We recommend to remove unused import `Math.sol`

CLIENT'S COMMENTARY

Next version will implement this recommendation.

CMT-6	Unused import `Address`
File	ProviderStrategy.sol HegicJoint.sol Joint.sol LPHedgingLib.sol
Severity	Comment
Status	Acknowledged

DESCRIPTION

At lines:
ProviderStrategy.sol#L9
HegicJoint.sol#L9
Joint.sol#L9
LPHedgingLib.sol#L7

RECOMMENDATION

We recommend to remove unused import `Address`.

CLIENT'S COMMENTARY

Next version will implement this recommendation.

CMT-7	Incorrect range for `period` in `setHedgingPeriod()`
File	HegicJoint.sol
Severity	Comment
Status	Acknowledged

DESCRIPTION

`require()` in `setHedgingPeriod()` is set as `period < 90`:
HegicJoint.sol#L141

However, `period` can be equal to 90:
HegicPool.sol#L233

RECOMMENDATION

We recommend to change condition to `period <= 90` in `setHedgingPeriod()`.

CLIENT'S COMMENTARY

Next version will implement this recommendation. This issue only prevents 1s of the range from being used.

CMT-8	getReward is not used and has incorrect implementation
File	SushiJoint.sol
Severity	Comment
Status	Acknowledged

DESCRIPTION

At the line 142
SushiJoint.sol#L142
there is an incorrect deposit call of masterchef with zero argument.

RECOMMENDATION

We recommend to remove unused code.

CLIENT'S COMMENTARY

Next version will implement this recommendation

3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build open-source solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

BLOCKCHAINS

- Ethereum
- Cosmos
- EOS
- Substrate

TECH STACK

- Python
- Solidity
- Rust
- C++

CONTACTS

https://github.com/mixbytes/audits_public

https://mixbytes.io/

hello@mixbytes.io

https://t.me/MixBytes

https://twitter.com/mixbytes