·Decurity·

Smart Contract Security Audit Report

Yearn No Hedge Joint Strategy

Decurity, 2022

Contents

1. General Information

This report contains information about the results of the security audit of the Yearn Finance (hereafter referred to as “Customer”) smart contracts, conducted by Decurity in the period from 07/04/2022 to 07/17/2022.

1.1. Introduction

Tasks solved during the work are:

- Review the protocol design and the usage of 3rd party dependencies,
- 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 contracts in the following repository: https://github.com/fp-crypto/joint-strategy. Initial review was done for the commit 82bd9cda6b60ffd9d61f7c57a89b865d21e124bc.

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 centralization risks have not been considered upon the request of the Customer.

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. 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 *Exploiting the weaknesses in the compiler or the runtime of the smart contracts*	Anyone

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 haven't discovered any critical exploitable vulnerabilities. However, the report includes suggestions about fixing the low-risk issues and implementing some best practices (see 3.1).

The Yearn Finance 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 Jul 24, 2022.

Table. Discovered weaknesses

Issue	Contract	Risk Level	Status
Migration permissions are loose	contracts/Joint.sol	Medium	Acknowledged
Insufficient Uniswap v3 callbacks access control	contracts/DEXes/UniV3StablesJoint.sol	Low	Acknowledged
Multiple "sandwiching" front-running vectors	contracts/DEXes/UniV3StablesJoint.sol	Low	Acknowledged
Missing check that position is open	contracts/DEXes/UniV3StablesJoint.sol	Low	Acknowledged
Role model inconsistency	@yearnvaults/contracts/BaseStrategy.sol	Low	Acknowledged
Unused dependencies	contracts/Joint.sol	Low	Acknowledged

2.2. Role Model

We've been able to identify the following key roles in the protocol:

- Vault: Only user touch-point, manages funds,
- Keeper: a bot which maintains the strategy, by ensuring it regularly generates returns for the Vault during pre-defined intervals or events,
- Governance: YFI token governance system (multisig),
- Management: Trusted with privileged access for limited operations (ensuring performance of Vaults),
- Strategist: original creator of strategy, is in charge of monitoring its position for adverse effects,
- Guardian: Trusted with privileged access for limited operations (ensuring safety of Vaults),
- Pool: Uniswap/Curve pool,
- Provider: ProviderStrategy contract.

The table below lists the roles implemented using the modifiers.

FileName	Modifier	Requirement	Roles
Joint.sol	onlyGovernance	msg.sender == providerA.vault().governance() msg.sender == providerB.vault().governance()	governance
Joint.sol	onlyVaultManagers	msg.sender == providerA.vault().governance() msg.sender == providerB.vault().governance() msg.sender == providerA.vault().management() msg.sender == providerB.vault().management()	governance management
Joint.sol	onlyProviders	msg.sender == address(providerA) msg.sender == address(providerB)	provider
Joint.sol	onlyKeepers	msg.sender == providerA.keeper()	keeper

		msg.sender == providerB.keeper() msg.sender == providerA.vault().governance() msg.sender == providerB.vault().governance() msg.sender == providerA.vault().management() msg.sender == providerB.vault().management()	governance management
@yearnvaults/contracts/BaseStrategy.sol	onlyAuthorized	msg.sender == strategist msg.sender == governance()	strategist governance
@yearnvaults/contracts/BaseStrategy.sol	onlyEmergencyAuthorized	msg.sender == strategist msg.sender == governance() msg.sender == vault.guardian() msg.sender == vault.management()	strategist governance guardian management
@yearnvaults/contracts/BaseStrategy.sol	onlyStrategist	msg.sender == strategist	strategist
@yearnvaults/contracts/BaseStrategy.sol	onlyGovernance	msg.sender == governance()	governance
@yearnvaults/contracts/BaseStrategy.sol	onlyKeepers	msg.sender == keeper msg.sender == strategist msg.sender == governance() msg.sender == vault.guardian() msg.sender == vault.management()	keeper strategist governance guardian management
@yearnvaults/contracts/BaseStrategy.sol	onlyVaultManagers	msg.sender == vault.management() msg.sender == governance()	management governance

The next table lists roles implemented within the functions (using the require statements).

FileName	Modifier	Requirement	Roles
UniV3StablesJoint.sol	uniswapV3MintCallback	msg.sender == address(_pool)	pool
UniV3StablesJoint.sol	uniswapV3SwapCallback	msg.sender == address(_pool)	pool
ProviderStrategy.sol	setJoint	JointAPI(_joint).providerA() == address(this)	provider

		JointAPI(_joint).providerB() == address(this)	
BaseStrategy.sol	withdraw	msg.sender == address(vault)	vault
BaseStrategy.sol	migrate	msg.sender == address(vault)	vault

3. General Recommendations

This section contains general recommendations on how to fix discovered 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

Follow the recommendations in section 4.

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,
- Perform regular audits for all the new contracts and updates,
- Ensure the secure off-chain storage and processing of the credentials (e.g. the privileged private keys),
- Launch a public bug bounty campaign for the contracts.

4. Findings

4.1. Migration permissions are too loose

Risk Level: Medium

Status: The vulnerability has been acknowledged by the customer, no fixes will be deployed.

Contracts:

- contracts/Joint.sol

References:

- https://github.com/yearn/yearn-security/blob/master/audits/20220409_Mixbytes_Yearn_Joint_Strategy/Mixbytes_-_Yearn_Joint_Strategy_Security_Audit_Report.pdf

Description:

The function *migrateProvider* in the contract *Joint* has a modifier *migrateProvider* that allows both *providerA* and *providerB* to migrate each other's strategy. A malicious provider can change the *StrategyProvider* of the other provider and steal their funds.

Remediation:

As mentioned in the previous report, *migrateProvider* should differentiate between *providerA* and *providerB* when updating strategy addresses.

4.2. Insufficient Uniswap v3 callbacks access control

Risk Level: Low

Status: The vulnerability has been acknowledged by the customer, no fixes will be deployed.

Contracts:

- contracts/DEXes/UniV3StablesJoint.sol

References:

- https://consensys.net/diligence/audits/2022/02/gamma/#uniswap-v3-callbacks-access-control-should-be-hardened

Description:

UniV3StablesJoint has the *uniswapV3MintCallback* and *uniswapV3SwapCallback* callbacks that are called by a Uniswap v3 pool when a new position is created and a swap occurs respectively. Both callbacks are protected with a require statement that checks that the caller is a Uniswap pool. Although it is an adequate protection in most cases, this check might not be sufficient if a pool is tricked into making malicious calls.

Remediation:

Consider adding state variables that ensure that a call to the callback was preceded by a corresponding operation, i.e. *uniswapV3MintCallback* should be called only after *createLP* or *harvest*, and *uniswapV3SwapCallback* only after *closePositionReturnFunds*.

4.3. Multiple “sandwiching” front running vectors

Risk Level: Low

Status: The vulnerability has been acknowledged by the customer, no fixes will be deployed.

Contracts:

- contracts/DEXes/UniV3StablesJoint.sol

References:

- https://consensys.net/diligence/audits/2022/02/gamma/#hypervisor---multiple-sandwiching-front-running-vectors

Description:

Calls to the Uniswap v3 functions swap, mint, and burn are susceptible to “sandwiching” vectors. While some of the calls are protected with minimum amount checks, the following ones are not:

- pool.mint in the function createLP on line 441 of contracts/DEXes/UniV3StablesJoint.sol
- pool.mint in the function harvest on line 798 of contracts/DEXes/UniV3StablesJoint.sol
- pool.burn via _closePosition on line 751 in contracts/Joint.sol
- pool.swap via closePositionReturnFunds on line 314 in contracts/Joint.sol (_minOutAmount is passed as zero)

Remediation:

Perform a check to ensure that token amount after the operation satisfies your slippage requirements.

4.4. Missing check that position is opened

Risk Level: Low

Status: The vulnerability has been acknowledged by the customer, no fixes will be deployed.

Contracts:

- contracts/DEXes/UniV3StablesJoint.sol

Description:

The function harvest in the contract *UniV3StablesJoint* can be called by keepers to compound generated fees into the existing UniV3 position. However, the function does not ensure that the position is actually opened.

Remediation:

In case of an out-of-order call to harvest before openPosition the transaction will be reverted as *UniswapV3Pool* requires *maxTick* to be greater than *minTick* (`require(tickLower < tickUpper, 'TLU');`). We still suggest to explicitly ensure in harvest that a position is already opened, e.g. by checking minTick:

`require(minTick > 0);`

4.5. Role description inconsistency

Risk Level: Low

Status: The vulnerability has been acknowledged by the customer, no fixes will be deployed.

Contracts:

- @yearnvaults/contracts/BaseStrategy.sol

References:

- https://docs.yearn.finance/developers/v2/SPECIFICATION

Description:

The Strategy Specification of the Yearn and comments of the *onlyEmergencyAuthorized* modifier say that only Governance or the Strategist can trigger the Strategy to enter into Emergency Exit Mode. However, this modifier also grants permission to the Guardian and Management.

Remediation:

The intended privileges of the roles should be clarified.

4.6. Unused dependencies

Risk Level: Low

Status: The vulnerability has been acknowledged by the customer, no fixes will be deployed.

Contracts:

- contracts/Joint.sol

Description:

There are two unused dependencies in the *Joint.sol* contract:

- Math.sol
- ySwapper.sol

Remediation: Remove unused dependencies.

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 expertise in the 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.