YEARN FINANCE VESTING ESCROW SECURITY AUDIT REPORT

October 13, 2023

MixBytes()

TABLE OF 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 the Client. 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. The security engineers check the provided source code independently of each other in accordance with the methodology described below:

1. Project architecture review:

- Project documentation review.
- General code review.
- Reverse research and study of the project architecture on the source code alone.

Stage goals

- Build an independent view of the project's architecture.
- Identifying logical flaws.

2. Checking the code in accordance with the vulnerabilities checklist:

- Manual code check for vulnerabilities listed on the Contractor's internal checklist. The Contractor's checklist is constantly updated based on the analysis of hacks, research, and audit of the clients' codes.
- Code check with the use of static analyzers (i.e Slither, Mythril, etc).

Stage goal

Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flash loan attacks etc.).

3. Checking the code for compliance with the desired security model:

- Detailed study of the project documentation.
- Examination of contracts tests.
- Examination of comments in code.
- Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit.
- Exploits PoC development with the use of such programs as Brownie and Hardhat.

Stage goal

Detect inconsistencies with the desired model.

4. Consolidation of the auditors' interim reports into one:

- Cross check: each auditor reviews the reports of the others.
- Discussion of the issues found by the auditors.
- Issuance of an interim audit report.

Stage goals

- Double-check all the found issues to make sure they are relevant and the determined threat level is correct.
- Provide the Client with an interim report.

5. Bug fixing & re-audit:

- The Client either fixes the issues or provides comments on the issues found by the auditors. Feedback from the Customer must be received on every issue/bug so that the Contractor can assign them a status (either "fixed" or "acknowledged").
- Upon completion of the bug fixing, the auditors double-check each fix and assign it a specific status, providing a proof link to the fix.
- A re-audited report is issued.

Stage goals

- Verify the fixed code version with all the recommendations and its statuses.
- Provide the Client with a re-audited report.

6. Final code verification and issuance of a public audit report:

- The Customer deploys the re-audited source code on the mainnet.
- The Contractor verifies the deployed code with the re-audited version and checks them for compliance.
- If the versions of the code match, the Contractor issues a public audit report.

Stage goals

- Conduct the final check of the code deployed on the mainnet.
- Provide the Customer with a public audit report.

Finding Severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss funds.
Low	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Customer 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 Customer is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

1.3 Project Overview

The audited project consists of two Vyper smart-contracts, implementing the token vesting. The enhanched functionality are:

- support for the vesting_start in the past
- support for the cliff period
- ability to claim partial amounts
- option to claim to a different account
- option to claim from a different account (if enabled)
- emergency drawback feature for unvested amount (can be disabled by invoking `disown`).

1.4 Project Dashboard

Project Summary

Title	Description
Client	Yearn Finance
Project name	Vesting Escrow
Timeline	5 Sep 2023 - 13 Oct 2023
Number of Auditors	3

Project Log

Date	Commit Hash	Note
05.09.2023	945b5ca09d8bb2d2ad9132df7368cb4992496f6a	Initial commit for the audit
06.09.2023	1664e9fc720a3ff80ce3780bdaaaf0e2c9bc8ebe	Commit for the re-audit
12.10.2023	4d051eee3a754564cd3d3ece5de5dc8b890ba147	Commit with the updated Vyper compiler version

Project Scope

The audit covered the following files:

File name	Link
VestingEscrowSimple	VestingEscrowSimple.vy
VestingEscrowFactory	VestingEscrowFactory.vy

Deployments

File name	Contract deployed on mainnet	Comment
VestingEscrowSimple.vy	0x9692f652a3048eb7f5074e12b907f20d33f37a01	
VestingEscrowFactory.vy	0x200c92dd85730872ab6a1e7d5e40a067066257cf	

1.5 Summary of findings

Severity	# of Findings
Critical	0
High	0
Medium	1
Low	4

ID	Name	Severity	Status
M-1	The `VestingEscrowSimple` instance can be initialized without supplying a sufficient amount of vesting tokens	Medium	Acknowledged
L-1	Inaccurate value representation by the `locked` function after the `owner` revocation	Low	Fixed
L-2	Malfunction of the `collect_dust` function after the `owner` revocation	Low	Fixed
L-3	An unintentionally large vesting period	Low	Fixed
L-4	Lack of the `target` parameter checks	Low	Acknowledged

1.6 Conclusion

The audited scope includes well-written smart contracts. Test coverage is sufficient. After the audit 1 Medium and 4 Low severity findings have been discovered. All the findings have been confirmed and acknowledged or fixed by the client.

2. FINDINGS REPORT

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

M-1	The `VestingEscrowSimple` instance can be initialized without supplying a sufficient amount of vesting tokens
Severity	Medium
Status	Acknowledged

Description

The `VestingEscrowSimple` contract assumes that the contract holds a sufficient amount of vesting tokens. However, the vesting contract can be initialized without supplying the required tokens if the instance is created without involving the original `VestingEscrowFactory` contract VestingEscrowSimple.vy#L54.

This may mislead the recipient regarding the actual amount to be distributed. Additionally, it may cause the `claim()` function with default arguments to be reverted VestingEscrowSimple.vy#L142.

Recommendation

It is recommended to perform a check of the vesting token amount during the contract initialization. If support for fee-on-transfer tokens is planned, the `initialize` function can store the actual amount of tokens present on the contract, instead of relying on its argument value.

Client's commentary

> **wontfix:**
>
> - the contract doesn't support fee-on-transfer tokens
> - only intended behavior is Factory > Escrow creation path. Direct Escrow deployments are out of scope

2.4 Low

L-1	Inaccurate value representation by the `locked` function after the `owner` revocation
Severity	Low
Status	Fixed in 1664e9fc

Description

VestingEscrowSimple.vy#L127
A vulnerability was discovered in the internal `_locked` function. If the `owner` revokes access and subsequently someone sends tokens to the contract, then the balance of the contract no longer reflects the remaining tokens available for claim. As a result, the value returned by the `locked` function can be larger than expected.

Recommendation

We recommend modifying the internal `_locked` function as follows:

```
return self._total_vested_at(self.disabled_at) - self._total_vested_at(time)
```

L-2	Malfunction of the `collect_dust` function after the `owner` revocation
Severity	Low
Status	Fixed in 1664e9fc

Description

VestingEscrowSimple.vy#L216
VestingEscrowSimple.vy#L178
The issue was identified in the `collect_dust` function. When a `recipient` attempts to withdraw an amount of `self.token` that exceeds the total value, they can claim via the `collect_dust` call after the `self.owner` has invoked the `revoke` function, so the issue arises. In such a scenario, the `self.token` balance of the contract diminishes below the `self.total_locked` value. Consequently, the residual funds or "dust" transferred to the contract cannot be collected.

Recommendation

We recommend modifying the expression located here VestingEscrowSimple.vy#L216 to:

```
amount = amount +self.total_claimed -self._total_vested_at(self.disabled_at)
```

L-3	An unintentionally large vesting period
Severity	Low
Status	Fixed in 1664e9fc

Description

The `VestingEscrowSimple` contract can be initialized with an unintentionally large vesting period, i.e., by supplying a value denoted in milliseconds instead of seconds VestingEscrowFactory.vy#L92.

Recommendation

We recommend performing a range check of the vesting period or denoting it in days in order to prevent issues related to specifying time in seconds or milliseconds.

Client's commentary

> Put comments that all durations are in seconds.

L-4	Lack of the `target` parameter checks
Severity	Low
Status	Acknowledged

Description

The vesting factory constructor doesn't have necessary checks for the `target` parameter: VestingEscrowFactory.vy#L51.
This may lead to the loss of vesting tokens, transferred to an uncontrolled address or the wrong smart contract.

Recommendation

We recommend adding a zero address check and a bytecode length check or performing EIP-165 checks.

Client's commentary

> **wontfix:** the parameters set on factory init have public getters, inviting the user to verify the correctness of deployment

3. ABOUT MIXBYTES

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

Contacts

https://github.com/mixbytes/audits_public

https://mixbytes.io/

hello@mixbytes.io

https://twitter.com/mixbytes