ENJOYOORS EVM VAULTS SECURITY AUDIT REPORT

March 11, 2025

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 Enjoyoors EVM Vaults protocol is an ERC20 token vault system that features whitelisted tokens, configurable deposit limits, and a two-step withdrawal process with mandatory time delays. The protocol employs role-based access control and incorporates safety mechanisms, such as emergency pauses, while tracking user balances and requiring withdrawal approvals through a dedicated contract.

1.4 Project Dashboard

Project Summary

Title	**Description**
Client	Enjoyoors
Project name	EVM Vaults
Timeline	16.12.2024 - 11.03.2025
Number of Auditors	3

Project Log

Date	**Commit Hash**	**Note**
16.12.2024	efbc1b269e0f84c631437156a6fa878d51935822	Commit for the audit
23.01.2025	a72a9bf7f514d0947107d60c080971a81fe2c405	Commit for the re-audit

Project Scope

The audit covered the following files:

File name	**Link**
contracts/vault/EnjoyoorsWithdrawalApprover.sol	EnjoyoorsWithdrawalApprover.sol
contracts/vault/EnjoyoorsVault.sol	EnjoyoorsVault.sol
contracts/vault/access/Access.sol	Access.sol
contracts/vault/access/Pauses.sol	Pauses.sol

File name	Link
contracts/vault/access/Setup.sol	Setup.sol
contracts/vault/access/TokenListing.sol	TokenListing.sol
contracts/vault/actions/EnjoyoorsVaultWithdrawals.sol	EnjoyoorsVaultWithdrawals.sol
contracts/vault/actions/EnjoyoorsVaultDeposits.sol	EnjoyoorsVaultDeposits.sol
contracts/vault/EnjoyoorsVaultBase.sol	EnjoyoorsVaultBase.sol
contracts/libraries/Asserts.sol	Asserts.sol

Deployments

File name	Contract deployed on mainnet	Comment
EnjoyoorsWithdrawalApprover.sol	0x8A4E79...e98B7c17	Ethereum Mainnet
EnjoyoorsVault.sol	0x59660c...27046409	Ethereum Mainnet
EnjoyoorsWithdrawalApprover.sol	0x8A4E79...e98B7c17	Base
EnjoyoorsVault.sol	0x59660c...27046409	Base
EnjoyoorsWithdrawalApprover.sol	0x8A4E79...e98B7c17	Arbitrum
EnjoyoorsVault.sol	0x59660c...27046409	Arbitrum
EnjoyoorsWithdrawalApprover.sol	0x8A4E79...e98B7c17	Avalanche
EnjoyoorsVault.sol	0x59660c...27046409	Avalanche

1.5 Summary of findings

Severity	# of Findings
Critical	2
High	0
Medium	1
Low	4

ID	Name	Severity	Status
C-1	Reentrancy Vulnerability in `_claimWithdrawal` Function	Critical	Fixed
C-2	Stuck Rebase Rewards Due to Internal Balance Tracking	Critical	Fixed
M-1	Absence of Fee-on-transfer Protection in `_deposit` Function	Medium	Fixed
L-1	Need for ERC165 Check in the `EnjoyoorsWithdrawalApprover`	Low	Fixed
L-2	Lack of Return Value Verification in `_claimWithdrawal` Function	Low	Fixed
L-3	Misleading Variable Name in Withdrawal Approver Contract	Low	Acknowledged
L-4	`vaultWithdrawalClaimable()` Does Not Check `WithdrawalRequest.claimed` Flag	Low	Fixed

1.6 Conclusion

The primary objective of the audit was to verify that the deposit functionality and subsequent token claims operate correctly and securely. It was necessary to ensure that users are guaranteed to withdraw their tokens after the owner-specified time period and cannot withdraw more tokens than intended. Additionally, a comprehensive review was conducted based on our internal checklist and potential attack scenarios.

The audit considered the following key aspects of the system, along with potential attack vectors and points of failure:

Project Architecture.
Inheritance for the contracts is correctly implemented, with no unused imports or unimplemented functionality. However, there is a complex inheritance structure within the `EnjoyoorsVault` contract. It may be beneficial to simplify the project structure in future versions, given its relatively straightforward functionality.

Support for Different Token Types.
All issues classified as Medium severity and higher are related to the lack of support for certain specific token types, such as tokens with callbacks, fee-on-transfer tokens, and rebasable tokens. Recommendations for addressing these concerns are provided in the corresponding findings within the report.

Access Control.
The `AccessControl` library is used correctly, with the `DEFAULT_ADMIN` role defined and separate roles assigned for each configurable protocol parameter. Every function that modifies the system's configuration is properly restricted to the corresponding role.

System Configuration.
Each configurable parameter can be modified in both directions: deposit limits can be increased or decreased, both deposits and withdrawals can be paused or unpaused, among other actions.
The current protocol implementation does not allow removing tokens from the whitelist, which mitigates the risk of funds becoming stuck if a token with a non-zero balance is accidentally removed. However, if such functionality is introduced in the future, it will be crucial to ensure that tokens with non-zero balances cannot be removed from the whitelist.

Restrictions on User-Supplied Addresses.
Each function that interacts with user-supplied tokens verifies them against the list of whitelisted tokens. The `EnjoyoorsWithdrawalApprover.vaultWithdrawalClaimable()` function accepts any address as the `enjoyoorsVault` parameter, allowing a malicious user to supply a poisoned vault. However, since this is a `view` function that is not used within the current protocol's scope, it does not pose any risks to the system.

Handling of Invalid Withdrawal Request IDs.
The `EnjoyoorsVault.getWithdrawalRequestById()` function returns an empty `WithdrawalRequest` for non-existent `requestId` values, which should be considered by calling functions. The withdrawal request with an index of `0` exists within the system, but it cannot be exploited to manipulate the protocol, as all relevant functions are protected with appropriate checks.

Despite the overall architectural soundness, greater care should be taken when implementing support for specific token types, or restrictions should be introduced to limit the types of tokens supported by the protocol. Addressing other issues and incorporating the recommended improvements will further enhance the protocol's efficiency and reliability.

2. FINDINGS REPORT

2.1 Critical

C-1	Reentrancy Vulnerability in `_claimWithdrawal` Function
Severity	Critical
Status	Fixed in a72a9bf7

Description

A significant security issue has been located in the `_claimWithdrawal` function of the `EnjoyoorsVaultWithdrawals` contract.

The vulnerability is embedded within the potential for a reentrancy attack. This could occur with ERC777 tokens, due to their callback on transfer. In this scenario, a hacker could repetitively call the `claimWithdrawal` function until the contract is fully drained of its assets.

This issue deems **Critical** severity because a reentrancy attack can end up causing serious damage by draining funds from the contract. Particularly for a contract handling withdrawal functions, this vulnerability exposes the contract to a substantial risk of loss.

Recommendation

We recommend adding the `nonReentrant` modifier to the `_claimWithdrawal` function as a precautionary measure. This will prevent recursive calls and therefore protect against potential reentrancy attacks.

Client's Commentary

> The corresponding modifier has been added to the all user functions at the following commit: 9383c157e4d8b5beab852ae241cecc9fe1a02940

C-2	Stuck Rebase Rewards Due to Internal Balance Tracking
Severity	Critical
Status	Fixed in a72a9bf7

Description

The issue is identified within the function `_deposit` of contract `EnjoyoorsVaultDeposits`.

The contract implements internal balance tracking through `userSupply[token][msg.sender] += amount` which does not account for rebasing tokens. Rebasing tokens (like stETH, aTokens) can increase their balance over time through protocol rewards or interest accrual. Since the contract tracks user balances internally and limits withdrawals to the tracked deposit amount, any additional tokens received from rebasing will become permanently locked in the contract.

The issue is classified as **Critical** severity because:

1. It leads to permanent loss of user rewards from rebasing tokens
2. There is no mechanism to recover these stuck tokens
3. The issue affects all users of rebasing tokens systematically
4. The locked rewards could potentially accumulate to significant amounts over time

Recommendation

We recommend implementing one of the following solutions:

1. Track user shares instead of absolute amounts:

    - Convert deposit amounts to shares based on the current total supply
    - Calculate withdrawal amounts based on the share of total balance
    - This automatically distributes rebases proportionally to all users

2. As a minimal solution:

    - Explicitly prohibit rebasing tokens if the contract cannot properly handle them
    - Use wrapped alternative of a rebasing token if exists (wstETH for stETH as an example)

Client's commentary

> The protocol won't support such tokens. The corresponding note will be added to the technical documentation.

2.2 High

Not Found

2.3 Medium

M-1	Absence of Fee-on-transfer Protection in `_deposit` Function
Severity	Medium
Status	Fixed in a72a9bf7

Description

An oversight has been detected within the `_deposit` function of the `EnjoyoorsVaultDeposits` contract.

The issue arises from the absence of a fee-on-transfer protection mechanism. This could result in incorrect setting of user amounts. Consequently, users might be in a position to claim more than they are entitled to. It gravely threatens the contract's integrity since this scenario inevitably leads to a situation where the last user cannot claim any funds, given there are insufficient funds on the contract due to the over-claims made by the preceding users.

This issue is classified as **Medium** severity due to its potential to distort the distribution of deposits and consequently funds, which could lead to the breakdown of the contract's operation.

Recommendation

We strongly suggest the addition of a fee-on-transfer protection within the `_deposit` function. This would ensure the correct amount is allocated to the user upon executing a deposit transaction, especially for tokens with transfer fees. This can be done by adding balance checks for the token before and after the `transferFrom` call. Note that it is crucial to add a `nonReentrant` check to protect the function from the reentrancy attack.

Client's Commentary

> The protocol won't support such tokens. The note will be added to the docs.
> The nonReentrant modifier has been added.
> In case any of the whitelisted tokens enables fee-on-transfer the deposits will be paused. The "last user withdrawal" problem will be solved by transferring the corresponding amount of tokens to the Vault.

2.4 Low

L-1	Need for ERC165 Check in the `EnjoyoorsWithdrawalApprover`
Severity	Low
Status	Fixed in a72a9bf7

Description

An issue has been detected within the `_changeWithdrawalApprover` function of the `Setup` contract. The call to change the `EnjoyoorsWithdrawalApprover` contract address does not currently have an ERC165 check to ensure that the new address is implementing the required interface.

Recommendation

We recommend introducing an ERC165 check within the `_changeWithdrawalApprover` function to ensure that the new contract address is implementing the required interface. Incorporating such a check would help in enhancing the integrity and stability of the protocol by ensuring that the new address can respond to interface-specific functions correctly.

Client's Commentary

> ERC165 check has been added at the following commit: 35804cccb61a13c64579541153ca28efbb91ea3b

L-2	Lack of Return Value Verification in `_claimWithdrawal` Function
Severity	Low
Status	Fixed in a72a9bf7

Description

An issue has been discovered in the `_claimWithdrawal` function of the `EnjoyoorsVaultWithdrawals` contract.

The function is currently not verifying the return value of the `canClaimWithdrawal()` function. As a result, the function could proceed even when the `canClaimWithdrawal()` method returns `false`.

This issue is identified as low severity since it might lead to potential bugs in the codebase but is not directly exploitable in the current state.

Recommendation

It is recommended to implement a check for the `canClaimWithdrawal()` function return value and handle any unexpected outcomes appropriately.

Client's Commentary

> canClaimWithdrawal has ben renamed to the more appropriate vefiryWithdrawal at the following commit: 1bdf6b3b17f735aa15cdc677e9a9929f926f8572

L-3	Misleading Variable Name in Withdrawal Approver Contract
Severity	Low
Status	Acknowledged

Description

The issue is identified within the state variable declaration of contract `EnjoyoorsWithdrawalApprover`.

The contract uses the variable name `withdrawalPeriodSeconds` which does not accurately reflect its actual purpose. This variable represents a timeout or waiting period that must elapse before a withdrawal can be claimed, rather than a general "period" which could be interpreted differently.

The issue is classified as **Low** severity because it's purely a code clarity concern that doesn't affect the contract's functionality or security, and the variable's purpose is still documented in the comments.

Recommendation

We recommend renaming the variable to `withdrawalUnlockingSeconds` to more accurately reflect its purpose as a mandatory waiting time between withdrawal request submission and claim.

Client's commentary

> We use such naming for consistency between other Vaults in other blockchains.

L-4	`vaultWithdrawalClaimable()` Does Not Check `WithdrawalRequest.claimed` Flag
Severity	Low
Status	Fixed in a72a9bf7

Description

This issue has been identified within the `vaultWithdrawalClaimable` function of the `EnjoyoorsWithdrawalApprover` contract.
Currently, the function returns `true` even if the withdrawal request has already been executed and `WithdrawalRequest.claimed` flag is `true`.

The issue is classified as **Low** severity because it may lead to incorrect or unintended behavior in certain scenarios, such as allowing a user to mistakenly believe they can claim a withdrawal.

Recommendation

We recommend checking the `claimed` flag within the `vaultWithdrawalClaimable` function and returning `false` if this flag is set to `true`.

Client's Commentary

> We decided to rename the vaultWithdrawalClaimable to isWithdrawalApproved at the following commit: a72a9bf7f514d0947107d60c080971a81fe2c405

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