KINTO × PESSIMISTIC

SECURITY ANALYSIS

by Pessimistic

This report is public

December 2, 2024

ABSTRACT

In this report, we consider the security of smart contracts of Kinto Access Registry project. Our task is to find and describe security issues in the smart contracts of the platform.

DISCLAIMER

SUMMARY

In this report, we considered the security of Kinto Access Registry smart contracts. We described the audit process in the section below.

The audit showed two issues of medium severity: Project roles, and Possible bridging DoS. Additionally, several low-severity issues were found.

The overall code quality is good.

GENERAL RECOMMENDATIONS

We recommend fixing the M02 issue and reviewing all low-severity issues.

PROJECT OVERVIEW

Project description

For the audit, we were provided with Kinto Access Registry project on a public GitHub repository, commit 6b9fc53015eefbb57e2b30b3d40734f08765ab9b.

The scope of the audit included the following contracts:

- **src/access/AccessRegistry.sol**;
- **src/access/AccessPoint.sol**;
- **src/access/workflows/AaveBorrowWorkflow.sol**;
- **src/access/workflows/AaveLendWorkflow.sol**;
- **src/access/workflows/AaveRepayWorkflow.sol**;
- **src/access/workflows/AaveWithdrawWorkflow.sol**;
- **src/access/workflows/BridgeWorkflow.sol**;
- **src/access/workflows/SwapWorkflow.sol**.

The documentation for the project included markdown file in the **src/access** folder.

The **kinto-core** project includes 699 tests, of which 694 pass successfully, while the remaining five are skipped. The code coverage for the files within the scope of the audit is 88.13%.

The total LOC of audited sources is 382.

AUDIT PROCESS

We started the audit on November 20, 2024 and finished on November 25, 2024.

We inspected the materials provided for the audit, conducted a call with the developers, and discussed confusing or suspicious parts of the code during our work.

We manually analyzed all the contracts within the scope of the audit and checked their logic. Among others, we verified the following properties of the contracts:

- The possibility of withdrawing funds from the **AccessPoint** to an arbitrary address in the event of a stolen private key;
- The correctness of integrations with Aave and 0x;
- Bridging scenarios (see M02).

We scanned the project with the following tools:

- Static analyzer Slither;
- Our plugin Slitherin with an extended set of rules;
- Semgrep rules for smart contracts. We also sent the results to the developers in the text file.

We ran tests and calculated the code coverage.

We combined in the report all the verified issues we found during the manual audit or discovered by automated tools.

MANUAL ANALYSIS

The contracts were completely manually analyzed, their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger project security. They can lead to loss of funds or other catastrophic consequences. The contracts should not be deployed before these issues are fixed.

The audit showed no critical issues.

Medium severity issues

Medium severity issues can influence project operation in the current implementation. Bugs, loss of potential income, and other non-critical failures fall into this category, as well as potential problems related to incorrect system management. We highly recommend addressing them.

M01. **Project roles**

The owner of the **AccessRegistry** contract has the following powers:

- Upgrade the implementation of **AccessPoint**;
- Upgrade the **AccessRegistry** contract implementation;
- Allow and disallow workflows. This could potentially block the option for the user to interact with Aave and prohibit bridging of the funds.

We recommend designing contracts in a trustless manner or implementing proper key management, e.g., setting up a multisig.

M02. **Possible bridging DoS**

In the current implementation of the **AaveBorrowWorkflow**, the user has the ability to provide arbitrary `BridgeData`. As a result, they can specify a `BridgeData.gasFee` that is used in the `depositERC20` function of the **Bridger** contract. Consequently, this amount of currency will be sent to the vault, effectively blocking the bridging functionality for other users until someone sends this currency back from the vault to the **Bridger**. We recommend limiting this behavior.

Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in future versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

L01. Approves are left for the vault

In **BridgeWorkflow** contract, **AccessPoint** gives an infinite approval that cannot be revoked in the future. We recommend using the**Bridger** contract instead.

L02. Different functions for the same logic

The **AccessRegistry** contract uses two different functions for encoding calls: `encodeCall` and `encodeWithSignature`. We recommend using the same functions.

L03. Unused code

`AmountOutTooLow` error from the **SwapWorkflow** contract is not used in the code. We recommend removing it.

L04. Unused imports

These imports are not used in the contracts:

- **IERC20** in **AccessPoint.sol** and **AccessRegistry.sol**;
- **ECDSA** in **AccessRegistry.sol**;
- **BaseAccount** in **AccessRegistry.sol**;
- **TokenCallbackHandler** in **AccessRegistry.sol**;
- **ByteSignature** in **AccessRegistry.sol**;
- **IWETH** in all Aave workflows;
- **IAccessPoint** in **BridgeWorkflow.sol**.

L05. Withdrawal of all assets

According to the `withdraw` function of the **AaveWithdrawWorkflow** contract, an `amount` equal to `type(uint256).max` signals that the user intends to withdraw all assets from Aave. However, the `withdrawAndBridge` function does not support this argument value, as it does not adjust the `amount` like the `withdraw` function does. Therefore, it requires users to specify the exact `amount` that corresponds to their entire `aToken` balance.

Notes

N01. Minimum return check is reliant on the 0x protocol

The correct pattern for swaps is to include a check for the minimum amount received. In the current implementation, this check is implemented using the `swapCallData` from the 0x protocol. Due to the abscence of this check on the contract itself, it is necessary to closely monitor the integration with 0x protocol because if it breaks while a transaction goes through, the user may lose money.

N02. Possibility to exit the system

If a user's private key gets compromised, there is a possibility for a hacker to move tokens from the `AccessPoint` via `SwapWorkflow`. This can be done by either filling a malicious limit order or swapping tokens through Uniswap V3 with the recipient changed.

This analysis was performed by Pessimistic:

Pavel Kondratenkov, Senior Security Engineer
Oleg Bobrov, Security Engineer
Irina Vikhareva, Project Manager
Alexander Seleznev, CEO

December 2, 2024