Kinto Bridger V2 Security Analysis

by Pessimistic

This report is public

June 6, 2024

Abstract

In this report, we consider the security of smart contracts of Kinto Bridger 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 Bridger V2 smart contracts (audit results of the V1 version are available here). We described the audit process in the section below.

The audit showed several issues of medium severity: Project roles, Unsigned addresses and Insufficient code coverage. Note that the M01 issue was also present in the old report. The "Approval failure" issue had medium severity in the previous report, and it is included as a note in this report. Moreover, several low severity issues were found.

After the initial audit, the codebase was updated.

The developers fixed the Insufficient code coverage issue of medium severity. They provided the comment on the Unsigned addresses issue of medium severity. Also, the developers added more comments to the code.

The number of tests and the code coverage increased.

The project integrates with multiple tokens, including USDe, sUSDe, and wstETH. And any other ERC20 tokens can be bridged. Additionally, there is an integration with the 0x protocol for swapping and with the Superbridge.

While these contracts were not audited by us directly, we tried to verify their integration and identify potential integration vulnerabilities. It is crucial to recognize that the project's exposure to risks escalates with the addition of external protocols.

The overall code quality of the project is good.

General recommendations

We have no further recommendations.

Project overview

Project description

For the audit, we were provided with Kinto Bridger project on a public GitHub repository, commit 28bc80b0c03be42d99584fb06ed3a91714063e26.

The scope of the audit included:

- **src/bridger/Bridger.sol**;
- **src/interfaces/IBridger.sol**.

The documentation included:

- The description of the project ("bridger-spec.md", 711be4e3ad97c94edbad0d708e8bd24ce3d72cfe sha1sum);
- The description of the integration with the Superbridge.

479 tests out of 479 pass successfully. The code coverage is 51.56%. The coverage was calculated for the scope of the audit.

The total LOC of audited sources is 398.

Codebase update

After the initial audit, the codebase was updated again, and we were provided with commit 60d7bcef5176b3815bd5edc86612a955f8d61e94.

The developers fixed or commented on all medium severity issues.

All 493 tests passed successfully. The code coverage increased to 97.47%.

Audit process

This is the v2 version of the Bridger. The report for the first version is available here. In this recheck, we checked fixes for the old issues and updated functionality. The issues that were not fixed or were fixed in the current version, were moved to this report.

We started the audit on May 28, 2024, and finished on May 30, 2024.

We did the preliminary research and started the review. In progress, we contacted the developers to get more information about the project and to ask some questions that occurred during the review and preliminary research.

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:

- Whether different types of tokens will be processed correctly;
- Whether nonces are updated properly;
- Whether the code follows Check-Effect-Interactions pattern;
- Whether there are frontrunning possibility;
- Whether there are no arbitrary calls that the unprivileged caller can fully control.

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 a private report all the verified issues we found during the manual audit or discovered by automated tools.

We made the recheck on June 6, 2024. We checked whether the previous issues were fixed and the new comments.

We re-ran the tests and calculated the code coverage. Finally, we updated the report.

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 (commented)

The project contains some functions that can be called only by the `owner` or `senderAccount` addresses:

- The `owner` can `pause`/`unpause` the deposits to the contract;
- The `owner` can upgrade the contract;
- The `owner` can change `senderAccount` using `setSenderAccount` function;
- The `owner` and `senderAccount` can call the `depositBySig` function with the signed data by user.

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

> *Note that some functions that were present on the old report were removed, as they are not relevant in the new version.*

> *Comment from the developers: The contract is owned by a 3 out of 5 Multisig using Gnosis Safe.*

M02. Unsigned addresses (commented)

The `bridgeData` parameter is not included in the signature in the deposits functionality. It is crucial for the `depositBySig` function, as a sender can put malicious addresses of the `vault` or `connector` and steal tokens.

> *Comment from the developers: Depositors who require a higher level of security or deposit larger amounts should use the `depositERC20` or `depositETH` function. These functions are called directly by the user, giving the user full control over the supplied arguments.*

M03. Insufficient code coverage (fixed)

The code coverage is low and equal to 51.56%. The important scenarios can be uncovered with the tests.

We always note the availability of tests as well as code coverage. We highly recommend improving the code coverage to avoid the appearance of sudden bugs.

The issue has been fixed and is not present in the latest version of the code.The code coverage increased to 97.47%.

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. Outdated contract version

Consider updating the contract version in the `_domainSeparatorV4` function to invalidate old signatures.

L02. Unavailable permit functionality

For some tokens, the `depositBySig` functionality can be unavailable in different chains. For instance, the Base version of the `DAI` token does not have the permit functionality.

L03. Unchecked amount of gas fee

There is no check of gas fee sent to the Superbridge (whether this gas is enough for transaction execution) in the `depositETH` and `_deposit` functions at lines 231 and 267. It can be checked by using the getMinFees function.

In this case, it does not lead to any problems and can be used as an additional check, as it is always possible to call retry in the Suprebridge to resend the transaction.

Notes

N01. Intrinsic gas optimization - 2 (fixed)

An allowance check performed in the `_permit` function may be called before `assembly` block at lines 345-348 to decrease gas consumption in case when allowance is already set.

The issue was discovered in the audit of the v1 version and was fixed on the current version of the codebase.

N02. Deposit by signature failure

After including `minReceive` into a `_hashSignatureData`, an owner or the privileged role can no longer manipulate `minReceive` when performing the swap. It means that the signer has to specify `minReceive`, which, on the other hand, forces the relayer (owner or privileged role) to execute the transaction with an already specified slippage value. In order to grief the owner out of gas, a signer can specify the slippage, which is close to zero.

Though the relayer will not attempt to execute the swap (through the `depositBySig` function), which fails during a simulation phase, it does not fully guarantee successful on-chain execution, especially for the low liquidity pools.

N03. Approval failure

In order to save gas, a lazy approval approach has been introduced. It relies on the fact that most tokens do not deduct an amount transferred from an allowance if the allowance has been assigned to `type(uint256).max`. Unfortunately, this is not the case with wstETH. Because of it, after a wstETH transfer, the allowance is no longer assigned to `type(uint256).max`.

Therefore, the `safeApprove` function will fail upon further transfers since `safeApprove` does not allow updating the allowance from non-zero to non-zero (it reverts if the approved amount and allowance are not equal to zero). It is possible when allowance is less than `amountBought` but greater than zero. Consider using the forceApprove function instead of `safeApprove` in the `depositETH` and `_deposit` functions.

> *The issue was discovered in the audit of the v1 version. In the current version, the severity was changed from medium to note since this case is more likely to be theoretically possible.*

This analysis was performed by Pessimistic:

Oleg Bobrov, Security Engineer
Daria Korepanova, Senior Security Engineer
Yhtyyar Sahatov, Security Engineer
Konstantin Zherebtsov, Business Development Lead
Irina Vikhareva, Project Manager
Alexander Seleznev, CEO

June 6, 2024