Kinto Bridger Security Analysis

by Pessimistic

This report is public

March 18, 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

The audit does not give any warranties on the security of the code. A single audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, a security audit is not investment advice.

Summary

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

The initial audit showed two critical issues:
The bought amount does not change after the stake, Arbitrary call. The audit also revealed several issues of medium severity: CEI violation, Chain ID is cached, No documentation, Discrepancy with documentation, Not all the data is signed, Project roles, Wrong check leads to swapping from any asset, Unchecked usage of approve and transferFrom for arbitrary tokens. Moreover, several low-severity issues were found.

After the initial audit, the codebase was updated.

The developers fixed all critical issues: Arbitrary call and The bought amount does not change after the stake. Also, they provided comment or fixed issues of medium-severity: CEI violation, Chain ID is cached, No documentation, Discrepancy with documentation, Not all the data is signed, Project roles, Wrong check leads to swapping from any asset, Unchecked usage of approve and transferFrom for arbitrary tokens. All low-severity issues were fixed.

During the recheck #1, we discovered one medium-severity issue: Approval failure, added point to the Project roles issue and found two notes.

The developers increased the number of tests and improved the code coverage.

The project integrates with multiple tokens, including USDe, sUSDe, weETH, sDAI, and wstETH. Additionally, an Arbitrum stack-based bridge is used for bridging purposes and there is the integration with the 0x protocol for swapping.

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 recommend fixing the mentioned issues and adding documentation.

Project overview

Project description

For the audit, we were provided with Kinto Bridger project on a public GitHub repository, commit ff95ae67109980b57d473de307adbd8af3bcb629.

The scope of the audit included:

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

The developers did not provide the documentation for the codebase.

379 tests out of 379 pass successfully. The code coverage is 83.11%. The coverage was calculated for the scope of the audit.

The total LOC of audited sources is 368.

Codebase update #1

After the initial audit, the codebase was updated, and we were provided with commit 9fde6b40011f4562a02e28f35176c1af595bcc84.

The developers fixed all critical issues and fixed or provided the comment on all medium-severity issues. We found one issue of medium-severity. The number of tests increased to 398. All of them passed. The code coverage was 96.42%.

Audit process

We started the audit on March 4, 2024 and finished on March 6, 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.

During the work, we stayed in touch with the developers and discussed confusing or suspicious parts of the code.

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 #1 on March 13-18, 2024. We checked the new functionality and whether the previous issues were fixed. Also, we re-ran the tests and asked the developers to provide the code coverage as we could not calculate it.

The integrations with 0x and L1GatewayRouter protocols were out of scope. However, we asked the developers for additional time to be sure that the integrations were safe. During this time, we checked for the most common issues when integrating with 0x and L1GatewayRouter by using Solodit and whether these issues can be applied to the **Bridger** contract.

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.

C01. Arbitrary call (fixed)

The user can make an arbitrary call from the contract that stores some tokens. Imagine some deposits are not yet bridged. An attacker can call `depositETH` function with some `swapData`, bypass all the checks before the `_fillQuote` call, and then transfer the balance of the contract or abuse given approves using the `swapTarget` and `swapCalldata` provided by the sender. The balance of the `buyToken` can be left unchanged, resulting in `boughtAmount` being equal to `0` at line 386, which will successfully pass the check at line 382 with the `minReceive` parameter equal to `0` in `_fillQuote` function of the **Bridger** contract.

The `minReceive` parameter has been added to the signature, and the check that the swap target address should be equal to the `exchangeProxy` from the 0x has appeared. We consider this issue as fixed. However, there are concerns that any functions can be called from the 0x protocol by passing any `swapCalldata`.

C02. The bought amount does not change after the stake (fixed)

The user can specify **sUSDe** as a final token for bridging. In this case, the `inputAsset` is converted to **USDe** and then **USDe** is staked using `_stakeAssets` function. However, **sUSDe** is minted as a share of the existing token supply, meaning that the `amountBought` value used in the `Deposit` event at line 279 is not correct. This issue allows the redemption of less/more tokens (depending on the conversion rate) on the destination chain. The issue can be found at lines 274 and 279 in `_swap` function of the **Bridger** contract.

The issue has been fixed and is not present in the latest version of the code.

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. CEI violation (fixed)

The code violates the CEI (checks-effects-interactions) patter, allowing the caller of `depositBySig` function to use the same `_signatureData` several times, as the nonce is updated only after the external call inside the `_swap` function at line 169. If the caller could obtain several permit signatures from the same user, they would need only one signature for `_signatureData` to consume all the permit signatures, making re-entrant calls to `depositBySig`. The issue can be found at line 177 of the **Bridger** contract.

The issue has been fixed and is not present in the latest version of the code.

M02. Chain ID is cached (commented)

The value of `domainSeparator` is cached in the constructor, including the `block.chainid` value in the separator. If there is a fork of the network, the provided signature will be redeemable on both chains. Consider taking the `block.chainid` value in runtime instead of caching it. The issue is at line 425 in `_domainSeparatorV4` function of the **Bridger** contract.

> *Comment from the developers: We do not consider it as an issue. We prefer to get the gas savings by caching it.*

M03. No documentation (commented)

The project has no proper documentation for the scope of the audit.

The documentation is a critical part that helps to improve security and reduce risks. It should explicitly explain the purpose and behavior of the contracts, their interactions, and key design choices. It is also essential for any further integrations.

> *Comment from the developers: Documentation for the Bridger.sol contract will be added on release and available on docs.kinto.xyz.*

M04. Discrepancy with comments (fixed)

The codebase of the project is well-commented. However, there were several places where the code was distinct from the comments:

- It is said that `ETH when swaps are disabled is just switched to wstETH.` at line 26, but the direct conversion is made only when the `inputAsset` is `ETH` and the `finalAsset` is `wstETH`. Otherwise, if swaps are disabled, the deposit will revert;
- It is said that `Only allows assets for now which do not require swap.` at line 154. However, if `_signatureData.inputAsset != _signatureData.finalAsset`, and the `inputAsset` is allowed, then the condition will be false, and there will be a swap later in the code.

Consider rewriting the comments or changing the logic to make them consistent with each other.

The issues have been fixed and are not present in the latest version of the code.

M05. Not all the data is signed (fixed)

The user who wants to bridge their tokens does not sign the `_kintoWallet` and `_swapData` arguments of the `depositBySig` function. It allows privileged users to specify these arguments however they want, e.g., any address for the `_kintoWallet` argument in `depositBySig` function of the **Bridger** contract.

Only the `_kintoWallet` parameter has been added to the signature, and the check that the swap target address (the `_swapData` field) should be equal to the `exchangeProxy` from the 0x has appeared. We consider this issue as fixed. However, there are concerns that any functions can be called from the 0x protocol by passing any `swapCalldata` from the `_swapData`.

M06. 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` can add/remove assets from allowed assets using `whitelistAssets` function;
- The `owner` can enable/disable swaps using `setSwapsEnabled` function;
- The `owner` and the `senderAccount` can call `depositBySig` to deposit tokens whenever they want;
- **(new)** The privileged roles can call the `bridgeDeposits` with any parameters. Since the L1 gateway router also belongs to the project, privileged roles can make a bridge to a less valuable L2 token or take more gas commissions. Also, they can estimate gas costs incorrectly and the user's deposits will be stuck or lost.

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

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

M07. Wrong check leads to swapping from any asset (fixed)

If an attacker passes any not allowed `inputAsset` token and **sUSDe** as `finalAsset` at lines 155–160 in `depositBySig` function of **Bridger** contract, it will allow bypassing checks as `if` condition at line 157 will be equal to `false` and will not revert as `&&` requires both expressions to be `true`. Consider changing `&&` to `||`.

The issue has been fixed and is not present in the latest version of the code.

M08. Unchecked usage of approve and transferFrom for arbitrary tokens (fixed)

There are several places where `approve`/`transferFrom` is used for unknown tokens. `approve` function definition requires `bool` to be returned; however, it is not the case for some tokens, e.g., **USDT**. Moreover, the result of `transferFrom` at line 348 is not checked. It could be the source of the problems, even if there is enough balance and allowance, e.g., if the user is blacklisted. Consider checking the results of these calls. The OpenZeppelins's SafeERC20 library and its safe functions `forceApprove` and `safeTransferFrom` can be used to cover most of the cases. The issues can be found at lines 211, 312, 348, and 379 of the **Bridger** contract.

The issues have been fixed and are not present in the latest version of the code.

M09. Approval failure (new)

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. Consider fixing it in the context of the `bridgeDeposits` and `_fillQuote` functions.

Note: in the `_fillQuote` function, this case only applies when the exact number of received tokens (not the input tokens) is set in the `swapCallData`. However, since the `minReceive` value is passed to the function, the `swapCallData` seems to have the exact number of input tokens.

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. Possible frontrunning (fixed)

It is possible for anyone to frontrun the `depositBySig` call and consume the `permitSignature` that was specified in the `depositBySig` call. Then, the second try to consume the signature at line 344 will revert as the nonce will be invalid, but in fact, the allowance is already given. The issue is located at line 334 in `_permit` function of the **Bridger** contract.

The issue has been fixed and is not present in the latest version of the code.

L02. Slither config is not working (fixed)

It seems that **slither.config.json** contains an incorrect filter for the `filter_paths` parameter, resulting in filtering all the contracts when the `slither` is run in the project.

The issue has been fixed and is not present in the latest version of the code.

L03. Usage of expires for two different signatures (fixed)

The `signature.expiresAt` is used for validation of the `_signature.signature` and of the `permitSignature,` however, these values can be distinct. The check at line 406 reverts if `>=` condition is met, but some tokens (including the ERC20Permit implementation by `OpenZeppelin`) counts permits made at `expiresAt == block.timestamp` as valid. In this case, the check at line 406 in `onlySignerVerified` function of the **Bridger** contract will fail, but the inner `permit` check would succeed.

The issue has been fixed and is not present in the latest version of the code.

Notes

N01. Intrinsic gas optimization

Since `bridgeDeposit()` is expected to be invoked every hour on Ethereum, it is important to make it gas-efficient. We provided findings of the `semgrep` tool that contains many gas optimization advices to the developers, and in addition, here are some more tips on reducing gas consumption in the **Bridger** contract:

- **(fixed)** Consider changing the location of the `signature` variable from `memory` to `calldata` as only `calldata` argument is passed to the `_permit` function. It will reduce gas costs and remove unnecessary copying;
- **(fixed)** Only `approve` if the `allowance` is 0;
- **(fixed)** Reduce the calldata size by using `bytes calldata data` with a further decoding;
- Possible to mine a signature for `bridgeDeposit()` function so that the binary search used in a dispatcher provides the signature on a first iteration.

N02. sUSDe contains blacklist mechanism (commented)

It is possible to be blacklisted in **sUSDe** token, which will result in an inability to bridge or transfer tokens from the **Bridger** contract.

> *Comment from the developers: Acknowledged. We have a relationship with the team.*

N03. The code cannot be used from smart wallets (fixed)

The explicit check `_signature.signer.code.length > 0` at line 410 will not allow smart wallets to perform deposits. At the same time, `isValidSignatureNow`, which supports verification using contracts, is used. Consider removing the check to allow contracts to interact with the project or use `ECDSA.tryRecover` to support only EOA signatures at line 413 in `onlySignerVerified` function of the **Bridger** contract.

The issue has been fixed and is not present in the latest version of the code.

N04. The depositCount is not emitted (fixed)

The `depositCount` state variable is not used anywhere except for incrementing at line 278. It seems that it should be emitted with the `Deposit` event at line 270 in `_swap` function of the **Bridger** contract.

The issue has been fixed and is not present in the latest version of the code.

N05. Intrinsic gas optimization - 2 (new)

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.

N06. Deposit by signature failure (new)

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.

This analysis was performed by Pessimistic:

Oleg Bobrov, Security Engineer
Daria Korepanova, Senior Security Engineer
Rasul Yunusov, Security Engineer
Irina Vikhareva, Project Manager

March 18, 2024