Via protocol Security Analysis

by Pessimistic

This report is public

October 27, 2022

Abstract

In this report, we consider the security of smart contracts of Via Protocol 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 Via Protocol smart contracts. We performed our audit according to the procedure described below.

The audit showed four issues of medium severity: Return value is always zero, Overpowered roles, Deleting the ViaRouter contract through delegatecall, Variables mixed up. Moreover, several low-severity issues were found.

After the initial audit the codebase was updated. Most of the issues have been fixed. One medium severity issue and one low severity issue were commented on.

General recommendations

We recommend using a whitelist of function signatures to be called by the validator. We recommend fixing the mentioned issues and improving NatSpec coverage. We also recommend implementing CI to run tests, calculate code coverage, and analyze code with linters and security tools.

Project overview

Project description

For the audit, we were provided with Via protocol project on a private GitHub repository, commit 4c094b9191239a063741b674a76db9c09112fbf4.

The scope of the audit includes everything inside the **contracts/** folder.

The documentation for the project includes **README.md** (sha1sum 197f220ab1a9dc909012f81b5b777e517b041d83) of the private repository. There is no public documentation available.

All 3 tests pass successfully. The code coverage is 70.27%.

The total LOC of audited sources is 1030.

Codebase update

After the initial audit, the codebase was updated. For the recheck, we were provided with commit ded3978b63dc93cfa4f2aced7cf55fb1494905b0. This update includes new functionality and fixes for most of the issues mentioned in the initial audit.

Local tests cannot check the work with the transfer of tokens through bridges. However, other tests work well. All 24 tests pass. The code coverage is 97.09%.

Procedure

In our audit, we consider the following crucial features of the code:

1. Whether the code is secure.
2. Whether the code corresponds to the documentation (including whitepaper).
3. Whether the code meets best practices.

We perform our audit according to the following procedure:

- Automated analysis
  - We scan the project's codebase with the automated tool Slither.
  - We manually verify (reject or confirm) all the issues found by the tool.
- Manual audit
  - We manually analyze the codebase for security vulnerabilities.
  - We assess the overall project structure and quality.
- Report
  - We reflect all the gathered information in the report.

Inter alia, we verify that:

- Upgradeable proxy pattern is implemented correctly.
- Router contracts cannot be compromised when interacting with them, bypassing a proxy.
- Adapters correctly integrate with corresponding bridges.
- The project token whitelist does not include tokens with features that can lead to security vulnerabilities or break important functionality.
- No standard Solidity issues are present in the codebase.

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 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. Return value is always zero (fixed)

The expression at line 206 in the `_swap` function of the **ViaRouter** contract returns `assetOut`, which is always a zero address. Consider replacing `assetOut` with `swapData.assetOut`.

The issue has been fixed with commit 6c1061771dad9cf451b53aadf7529144b84dfa0c.

M02. Overpowered roles (commented)

The system depends heavily on the roles listed below in the current implementation. Thus, some scenarios can lead to undesirable consequences for the project and its users, e.g., if one of the private keys becomes compromised.

- It is planned that the user will give unlimited approval for the **GaslessRelay** contract. Next, the validator will use the `GaslessRelay.execute` function. The validator can steal all of his tokens without the user's permission.
- The validator of a router can delete it via a delegatecall inside the `ViaRouter._bridge()` function. He can also transfer out all the fees by passing a malicious `viaData.assetIn` address to `ViaRouter.execute()` function and exploit a call from a router contract on line 160.
- The validator of a router can actually steal the funds from users who have given approval to the router.
- The owner can set the validator, and thus he has access to all the actions listed above.

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

> *Comment from the developers:*
> *a) We are aware of risks related to owner role privileges. To provide additional security, owner role will be transferred to a Multisig account. In future, we'll introduce Via Protocol token and DAO of token holders. After that, contract ownership will be transferred to that account.*
> *b) Regarding validator role, if we exclude selfdestruct opportunity (prevented by adapters whitelist, see L05 response), validator's opportunities are limited in our perspective. Stealing fees is not possible, because everything that happens inside `execute` function can only be done with funds that msg.sender has passed to contract (so any asset/amount that validator gives in calldata, will be transferred from his account to router before any other actions)*

M03. Deleting the ViaRouter contract through delegatecall (fixed)

Any **ViaRouter** implementation contract can be deleted by any user using the following exploit:

1. Call the `initialize` function of a logic contract directly, bypassing a proxy. Caller X becomes the validator.
2. X signs a message with `bridgeData.target` set to a malicious contract.
3. X calls `execute()`, passes the validator check, and executes the `_bridge()` function with a `delegatecall` to the malicious contract, which, in turn, deletes the **ViaRouter** contract via `selfdestruct()`.

The issue has been fixed with commit bdb9dd5ed2c270ad302122484b6aa25ff9d37a60.

M04. Variables mixed up (fixed)

In the **ViaRouter** contract, line 155 will correctly be

```
collectedFees[token] -= balanceBefore - balanceAfter;
```

The issue has been fixed with commit 2287082dbaa57540073721c10d622a8f66585ea2.

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. Unsafe usage of safeApprove (fixed)

A contracts uses a `safeApprove` function of the OpenZeppelin SafeERC20 library to approve incoming tokens. If the entire value of the variable was not used, then the next run is reverted. We recommend using the `Transfers.approve()` function instead.

- In the **MultichainAdapter** contract is not handled at line 35. The adapter will revert the next call if the bridge does not transfer the entire approved amount.
- In the **GaslessRelay** contract, line 146 uses the `safeApprove` function.

The issue has been fixed with commit 2287082dbaa57540073721c10d622a8f66585ea2.

L02. Unused constants (fixed)

Constants `NONCONITNUOUS_SWAP_PATH` and `INVALID_INCOMING_AMOUNT` of **Errors** library are never used.

The issue has been fixed with commit 6c1061771dad9cf451b53aadf7529144b84dfa0c.

L03. Typo in constant name (fixed)

The constant `NONCONITNUOUS_SWAP_PATH` of the **Errors** library is misspelled - consider renaming in to `NONCONTINUOUS_SWAP_PATH`.

The issue has been fixed with commit 6c1061771dad9cf451b53aadf7529144b84dfa0c.

L04. Wormhole integration issue (commented)

WormholeAdapter contract intergrates with the Wormhole bridge and calls its `wrapAndTransferETH` and `transferTokens` functions. Both of these functions return the `uint64` value. We recommend handling these return values to avoid unexpected issues.

> *Comment from the developers: In case if call does not fail, these functions only return sequence - basically, wormhole request ID. This ID isn't used in our contracts, therefore we don't see handling of this value necessary.*

L05. Redundant storage variable (fixed)

The `adapters` mapping at line 49 of the **ViaRouter** contract is supposed to be a whitelist of adapters addresses set by the owner of a ViaRouter. However, an adapter passed to `execute()` the function as `bridgeData.target` is never checked, and `adapters` mapping remains unused. Consider including the adapter address check or removing the redundant code.

The issue has been fixed with commit 6c1061771dad9cf451b53aadf7529144b84dfa0c.

L06. Unused imports (fixed)

The imported functionality of `SafeERC20Upgradeable` and `SafeMathUpgradeable` is never used. Consider removing the imports at lines 6-7 in the **ViaRouter** contract.

The issue has been fixed with commit 6c1061771dad9cf451b53aadf7529144b84dfa0c.

Notes

N01. Native currency is not transferred back to user (addressed)

The `extraNativeValue` variable in `execute` the function of the **ViaRouter** contract represents an amount of native tokens that a user sends with the call but does not include in `viaData.amountIn`. Then, depending on the adapter, `extraNativeValue` is either passed forward to the bridge or remains on the **ViaRouter** contract. The user sometimes loses his money if he passes extra native currency. Consider transferring `extraNativeValue` back to the user if it is not passed to the bridge.

N02. The protocol allows for rebase tokens (addressed)

Some ERC20 tokens control their price by minting or burning users' tokens during rebasing. According to this whitelist, some rebase tokens can be used in the protocol (Ampleforth, Olympus, TempleDAO, YAM, OUSD, Basis Cash, etc.). Assume some rebase tokens remain in the **ViaRouter** contract as a fee. In case the balance of the contract decreases by `x` due to rebasing, `x` fee tokens get impermanently stuck in the contract as `collectedFees[token]` does not change with the balance.

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Vladimir Pomogalov, Security Engineer
Ivan Gladkikh, Junior Security Engineer
Irina Vikhareva, Project Manager

October 27, 2022