PUBLIC

# Code Assessment

## of the Enzyme 31Third Adapter Smart Contracts

November 22, 2024

Produced for

by

# Contents

# 1 Executive Summary

Dear Enzyme Team,

Thank you for trusting us to help 31Third with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Enzyme 31Third Adapter according to Scope to support you in forming an opinion on their security risks.

31Third implements an adapter for Enzyme, which allows batch trades using the 31Third protocol. The adapter was made possible through an Enzyme grant to 31Third.

The most critical subjects covered in our audit are functional correctness, the isolation of the adapter from the rest of the system, and the correct integration with the external system. The isolation of the adapter was improved in response to BatchTrade Should Revert On Error. Security regarding all aforementioned subjects is high.

Some notes on the external system's behavior can be found in Replayable TradeSigner Signature and Rebasing Tokens with Transfer Loss are Not Supported.

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

## 1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

| | |
|---|---|
| Critical-Severity Findings | 0 |
| High-Severity Findings | 0 |
| Medium-Severity Findings | 0 |
| Low-Severity Findings | 2 |
| • Code Corrected | 2 |

# 2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

## 2.1 Scope

The assessment was performed on the source code files inside the Enzyme 31Third Adapter repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

| V | Date | Commit Hash | Note |
|---|---|---|---|
| 1 | 29 Mar 2024 | 6a152226acb68a420af5db02feffee2fc2162fff | Initial Version |
| 2 | 15 Apr 2024 | 5dc7840703af4b8a48a9a8f081a957df7ba935a5 | After Intermediate Report |
| 3 | 19 Nov 2024 | edfd4bddaa4b32c8e69a57a890a9df6143152022 | Retrieve Dust Asset |

The following files are in scope of this review:

```
contracts/external-interfaces/IThreeOneThird.sol
contracts/release/extensions/integration-manager/integrations/
    adapters/ThreeOneThirdAdapter.sol
contracts/release/extensions/integration-manager/integrations/
    utils/0.8.19/actions/ThreeOneThirdActionsMixin.sol
contracts/utils/0.8.19/AddressArrayLib.sol
```

For the solidity smart contracts, the compiler version `0.8.19` was chosen.

### 2.1.1 Excluded from scope

All the contracts that are not explicitly mentioned in the Scope section are excluded from scope. The external systems, with which the adapter interacts, are assumed to work correctly. Finally, attack vectors such as trades unfavorable for the fund by the fund manager have not been considered, as the managers are considered trusted by the system.

## 2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

At the end of this report section we have added subsections for each of the changes according to the versions.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Avantgarde Finance offers a 31Third adapter to integrate with Enzyme, which enables batch trading with two or more assets in a single transaction via the 31Third BatchTrade contract.

Enzyme is an investment management system that allows one to buy shares of certain vaults. Vaults act like funds and each vault has a vault manager, who is in charge of investing the funds of the vault into other DeFi protocols. The manager can set certain policies that limit the actions of the users including the manager's. Furthermore, the manager can only invest in third-party protocols for which an adapter exists.

### 2.2.1 31Third BatchTrade

31Third is a system designed to facilitate batch trading operations on decentralized liquidity sources such as DEXes, DEX aggregators, market makers and staking protocols.

Its core feature is batch trading (`batchTrade()`): The protocol allows users to execute multiple trades in a single transaction, potentially reducing gas fees and streamlining trading operations. Through a flexible adapter system, it supports various DEXes, allowing trades across multiple platforms within one batch operation. Trades are detailed in structured data including the exchange name, token addresses (from and to), amounts, and minimum receipt requirements. A fee will be charged after each swap, which may be waived if the `msg.sender` is a `feelessWallet`.

A `batchTradeConfig` will also be passed as input for a batch trade. User may choose to revert the batch execution in case any swap failed or return without revert by setting the flag `batchTradeConfig.revertOnError`.

### 2.2.2 31Third Adapter

The 31Third Adapter is a permissionless adapter which is designed to facilitate batched swaps from Enzyme vaults. The fund manager can initiate a call from the Comptroller to trigger the execution on the IntegrationManager, which will eventually call the 31Third adapter to batch swap on assets the vault proxy transfers.

The adapter implements `parseAssetsForAction()` to compute the net change of assets that is expected. It distinguishes the spending and incoming assets depending on the sign of their net change represented as `int256`. The vault proxy will transfer the spending assets to the adapter, before calling the `takeOrder()` function on the adapter.

`takeOrder()` will grant allowance to the `BatchTrade` contract and invoke `batchTrade()`. After the batch swap, all the incoming assets on the adapter will be sent back to the vault proxy.

### 2.2.3 Roles and Trust Model

Please refer to the main audit report and the extension audit reports for a general trust model of Enzyme protocol and its extensions.

In general, we assume Enzyme only interacts with normal ERC-20 tokens that do not have multiple entry points, callbacks, fees-on-transfer, or other special behaviors.

Fund owners and asset managers are generally fully trusted for a fund. However, their powers can be limited through the fund's settings. The funds' settings/policies are assumed to be set up correctly for the intended configuration/usage (e.g. a policy to limit the slippage in 31Third batch trade).

The managers are expected to regularly claim the fees and to pause the position if under-/overvaluation of the fund occurs.

Governance is fully trusted and expected to not only behave honestly but also to fully understand the systems they are interacting which includes choosing appropriate parameters.

All external systems are expected to be non-malicious and work correctly as documented.

The owner of 31Third protocol is partially trusted. They can configure feeless wallets, adjust fees, and pause or unpause the protocol. If 31Third is paused, the adapter cannot be used. The tradeSigner of 31Third protocol is partially trusted. The adapter and 31Third protocol cannot be used if the signer stops signing trade data.

### 2.2.4 Changes in Version 2

In Version 2 of the contracts, the call to 31Third `batchTrade()` is now always made with `revertOnError` set to true.

# 3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

# 4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

| Likelihood | Impact | | |
|---|---|---|---|
| | High | Medium | Low |
| High | Critical | High | Medium |
| Medium | High | Medium | Low |
| Low | Medium | Low | Low |

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

# 5 Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Design: Architectural shortcomings and design inefficiencies

Below we provide a numerical overview of the identified findings, split up by their severity.

| | |
|---|---|
| Critical-Severity Findings | 0 |
| High-Severity Findings | 0 |
| Medium-Severity Findings | 0 |
| Low-Severity Findings | 0 |

# 6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

| | |
|---|---|
| Critical-Severity Findings | 0 |
| High-Severity Findings | 0 |
| Medium-Severity Findings | 0 |
| Low-Severity Findings | 2 |

- BatchTrade Should Revert on Error Code Corrected
- Dust of Spending Asset May Be Left on the Adapter Code Corrected

| | |
|---|---|
| Informational Findings | 2 |

- Inaccurate minToReceive Computation Code Corrected
- Redundant Pragma Code Corrected

## 6.1 BatchTrade Should Revert on Error

Design Low Version 1 Code Corrected

*CS-EZTOT-001*

The `batchTrade` function takes a `revertOnError` config parameter. When it is set to false, the contract will not always revert on error. Instead, it will send the input tokens back to the caller. This could result in tokens remaining on the ThreeOneThirdAdapter.

In most cases, the failed BatchTrade will revert anyway, due to Enzyme's minIncoming check.

To ensure no funds can be left on the adapter, it should only be possible to call `batchTrade()` with `revertOnError` set to true.

---

**Code corrected:**

The `revertOnError` parameter has been removed from the enzyme input list. `batchTrade()` is now always called with `revertOnError` set to true.

## 6.2 Dust of Spending Asset May Be Left on the Adapter

Design Low Version 1 Code Corrected

*CS-EZTOT-006*

**Note:** This issue was reported by Avantgarde Finance.

The adapter will distinguish a token as a spending or incoming asset based on the expected net balance changes (based on the outgoing and min incoming amounts) over all the trades in

`parseAssetsForAction()`. Hence, for an expected net-outgoing token, it will not be regarded as an incoming asset even if the actual trades lead to a net income on it. Consequently, there might be left over of a token which is used as both trade input and output.

---

**Code corrected:**

A modifier, `postActionSpendAssetsTransferHandler` has been added to `takeOrder()`, which will push the remaining spending assets back to the vault proxy.

## 6.3 Inaccurate minToReceive Computation

**Informational** **Version 1** **Code Corrected**

*CS-EZTOT-002*

The fee charged in 31Third protocol is computed and rounded down as:

```
feeAmount = (_receivedAmount * feeBasisPoints) / 10000;
```

As a result, the minimum amount to receive after fees will be rounded up. However, in `parseAssetsForAction()` the minimum amount to receive after fees is rounded down:

```
assetChanges = assetChanges.addItem(int256(trades[i].minToReceiveBeforeFees *
    (10000 - feeBasisPoints) / (10000)));
```

As a result, the amount to receive after fees will be slightly inaccurate.

---

**Code corrected:**

The `assetChanges` calculation now correctly rounds up instead of down.

## 6.4 Redundant Pragma

**Informational** **Version 1** **Code Corrected**

*CS-EZTOT-004*

`ABIEncoderV2` is explicitly declared in `ThreeOneThirdAdapter` and `ThreeOneThirdActionMixin`. After solidity `v0.8.0`, ABI encoder v2 is activated by default, thus the following pragma has no effect.

```
pragma experimental ABIEncoderV2;
```

---

**Code corrected:**

The redundant pragma was removed.

# 7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

## 7.1 Rebasing Tokens With Transfer Loss Are Not Supported

**Informational** **Version 1** **Acknowledged**

*CS-EZTOT-003*

The `BatchTrade` contract does pre- and post-balance checks when transferring tokens. This ensures at least `fromAmount` will be received.

```
function _claimAndApproveFromToken(IExchangeAdapter, _exchangeAdapter,
    Trade memory _trade) private {
    if (_trade.from != ETH_ADDRESS) {
        uint256 fromBalanceBefore = _getBalance(_trade.from);
        IERC20(_trade.from).safeTransferFrom(
            msg.sender,
            address(this),
            _trade.fromAmount
        );
        uint256 fromBalanceAfter = _getBalance(_trade.from);
        if (fromBalanceAfter < fromBalanceBefore + _trade.fromAmount) {
            revert NotEnoughClaimed(_trade, _trade.fromAmount,
                fromBalanceAfter - fromBalanceBefore);
        }

        IERC20(_trade.from).safeIncreaseAllowance(
            _exchangeAdapter.getSpender(),
            _trade.fromAmount
        );
    }
}
```

However, this check may revert on transferring rebasing tokens such as stETH. Due to the shares to underlying conversion, there may be 1-2 wei loss during the transfer and the receiver will receive less than what is passed to `safeTransferFrom()`. A similar check is also performed in `_callExchange`.

This means that tokens such as stETH cannot always be traded using `BatchTrade`.

The 31Third docs explicitly state that fee-on-transfer tokens are not supported, but does not mention rebasing tokens.

---

31Third responded:

```
The 31Third backend is aware of this. This will also be adressed in v2 of the 31Third protocol.
```

## 7.2 Replayable TradeSigner Signature

Informational | Version 1 | Risk Accepted

*CS-EZTOT-005*

The `BatchTrade` contract (out of scope) has a `_preTradeCheck`, which verifies that the `tradeSigner` has signed the trade fields: `spender`, `from`, `fromAmount`, `to`, `minToReceiveBeforeFees`, and `data`.

The signed fields:

- Do not contain the domain separator.
- Are not bound to a `msg.sender`.
- Do not contain a nonce or expiration timestamp.

As a result, a signature can be reused in multiple ways:

- One user can reuse the signed trade data from another user.
- A trade data signed for Ethereum contracts can be replayed and accepted by contracts on another chain (e.g. Polygon) if they share the same tradeSigner.

Note that the same address may contain different code on different chains.

These scenarios should be carefully analyzed on 31Third protocol to ensure no unintended behavior is possible.

---

**Risk accepted**:

31Third responded:

```
Since only fund managers can rebalance
their Enzyme funds this issue can be neglected.
```

Additionally, 31Third stated that they would change the signer on Polygon to be a different address than the one on Ethereum. This mitigates the cross-chain signature replay.

# 8 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

## 8.1 The Adapter Should Not Be a Feeless Wallet on 31Third BatchTrade

Note Version 1

The owner of the 31Third BatchTrade contract has the privilege to set feeless wallets. An address registered as a feeless wallet has their fees waived in batch swaps.

In addition to the Enzyme Integration Manager, anyone can use the adapter for batch swaps, as it is permissionless.

31Third should not set the adapter as a feeless wallet, otherwise anyone can use the adapter to trade without fees.

## 8.2 Unsupported Trades

Note Version 1

Some batch trades are not supported by the adapter due to the way `parseAssetsForAction()` works: it goes through all the trades in a batch, and compute the net balance change given the exact outgoing and min expected incoming amount. An asset is regarded as an incoming asset if it has net incoming amount, and vise versa. Only the net outgoing balance is approved to the adapter.

As a result, if an asset is used as both input and output in a batch, there may be insufficient funds for the trade where it is input, hence the batch trade may revert. For instance, assume there is a batch of two trades between token A, B, C:

1. Trade1: *x* A -> *y* B
2. Trade2: *z* C -> *x* A
3. Net spending asset would be *z* C and *0* A.
4. As trades are executed in order, Trade1 will revert due to insufficient A token.