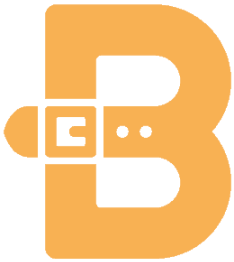
Belt Crosschain AMM Audit (Pt. 2)

Smart Contract Security Assessment

Dec 1, 2021



ABSTRACT

Dedaub was commissioned to perform a security audit on the Belt Finance Crosschain AMM contracts. We received the code base in the form of an archive file on Nov. 1, 2021. The primary contracts (non-test, non-interface, non-mock) are listed below:

```
crosschain/SwapReceiver.sol
crosschain/Router.sol
crosschain/FeeStore.sol
crosschain/Escrow.sol
crosschain/exchanges/ExchangeProxy.sol
```

SETTING & CAVEATS

The audited code base is of small size, at nearly 1KLoC. About half of it is completely newly audited code (new contracts FeeStore.sol, Escrow.sol).

VULNERABILITIES & FUNCTIONAL ISSUES

This section details issues that affect the functionality of the contract. Dedaub generally categorizes issues according to the following severities, but may also take other considerations into account such as impact or difficulty in exploitation:

Category	Description
CRITICAL	Can be profitably exploited by any knowledgeable third party attacker to drain a portion of the system's or users' funds OR the contract does not function as intended and severe loss of funds may result.
HIGH	Third party attackers or faulty functionality may block the system or cause the system or users to lose funds. Important system invariants can be violated.

MEDIUM	Examples: 01) User or system funds can be lost when third party systems misbehave. 02) DoS, under specific conditions. 03) Part of the functionality becomes unusable due to programming error.
LOW	Examples: 01) Breaking important system invariants, but without apparent consequences. 02) Buggy functionality for trusted users where a workaround exists. 03) Security issues which may manifest when the system evolves.

Issue resolution includes “dismissed”, by the client, or “resolved”, per the auditors.

CRITICAL SEVERITY

[No critical severity issues]

HIGH SEVERITY:

ID	Description	STATUS
H1	Wrong accounting for `AddLiquidity` transaction type	**RESOLVED**

In `SwapReceiver::onTokenBridgeReceived`, value `amountOut` results in wrong calculation in case of `AddLiquidity` transaction type:

```
address tokenOut = request.tokenOut;
 /// Dedaub: amountOut should be calculated after the next if statement
uint amountOut = _getTokenBalance(tokenOut);

if (txType == TransactionType.AddLiquidity) {
     tokenOut = address(ISwap(stableSwap).lpToken());
}

 /// Dedaub: requestLiquidity or requestSwap called
 amountOut = _getTokenBalance(tokenOut).sub(amountOut);
```

This can be exploited as follows: an attacker can initiate a valid `AddLiquidity` transaction and provide request.tokenOut data with any token of no or really low balance for the `SwapReceiver` contract. This fact will pass unnoticed, since nowhere in the code is `reqest.tokenOut` actually checked against the LP token. Because of the initial wrong balance accounting (`amountOut` calculated on `request.tokenOut` instead of `lpToken`) the attacker will get the whole lpToken balance of the contract.

MEDIUM SEVERITY:

[No medium severity issues]

LOW SEVERITY:

[No low severity issues]

OTHER/ ADVISORY ISSUES:

This section details issues that are not thought to directly affect the functionality of the project, but we recommend addressing them.

ID	Description	STATUS
A1	Redundant cast	**RESOLVED**

In `SwapReceiver::_requestLiquidity,` parameter txType is of TransactionType so the following cast is redundant:

```
if (TransactionType(txType) == TransactionType.AddLiquidity) {
```

ID	Description	STATUS
A2	Dead code	**RESOLVED**

In `Router.sol` modifier `onlyOperator` is defined but never used. More generally, the role of an operator is defined but essentially never used in the contract.

A3	Inconsistent refund-related checks within SwapReceiver	**RESOLVED**

When `SwapReceiver::_requestBridge` fails to successfully complete all internal transactions after fees have been purchased, `FeeStore::refund` is called so as to sell feeAmount back.

```
if (!succeed) {
    // refund Fee
    /// Dedaub: function returns `false` and a refund() is also called.
    _approve(FeeStore(feeStore).feeToken(), feeStore, feeAmount);
    FeeStore(feeStore).refund(token, feeAmount);
}
```

However, there is another case where `refund()` should also be called:

```
/// Dedaub: refund() is not called in this case
/// Dedaub: !feeBought doesn't necessarily mean that no fee was bought -
/// could also be that some was bought but not enough according to return
/// value of _buyFeeWithToken()
 if (!feeBought) {
     return false;
 }
```

A4	Zero initialization	**INFO**

There are several cases where variables are explicitly zero-initialized, when this is the default behavior.

`SwapReceiver::_requestSwap`:

```
uint received = 0;
```

`SwapReceiver::_requestLiquidity`:

```
uint tokenIndex = 0;
uint received = 0;
bool succeed = false;
```

`SwapReceiver::_requestBridge`, `SwapReceiver::onTokenBridgeReceived`, `SwapReceiver::_requestLiquidity`:

```
bool succeed = false;
```

A4	Compiler known issues	**INFO**

Most of the contracts were compiled with the Solidity compiler v.0.6.12 while some of them with v.0.5.0. Both versions have some known bugs at the time of writing. We have inspected the list of bugs and believe that they do not affect the audited contracts.

DISCLAIMER

The audited contracts have been analyzed using automated techniques and extensive human inspection in accordance with state-of-the-art practices as of the date of this report. The audit makes no statements or warranties on the security of the code. On its own, it cannot be considered a sufficient assessment of the correctness of the contract. While we have conducted an analysis to the best of our ability, it is our recommendation for high-value contracts to commission several independent audits, as well as a public bug bounty program.

ABOUT DEDAUB

Dedaub offers technology and auditing services for smart contract security. The founders, Neville Grech and Yannis Smaragdakis, are top researchers in program analysis. Dedaub's smart contract technology is demonstrated in the

contract-library.com service, which decompiles and performs security analyses on the full Ethereum blockchain.