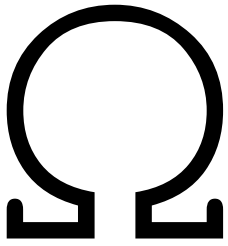
Opus-Edu

Final Audit Report

February 14, 2024



Team Omega

Teamomega.eth.limo

Summary

Kois Center has asked Team Omega to audit the contracts that define the behavior of their Opus Edu contracts.

We found **no high severity issues** - these are issues that can lead to a loss of funds, and are essential to fix. We classified **2** issues as “medium” - these are issues we believe you should definitely address. We classified **3** issues as “low”, and an additional **12** issues were classified as “info” - we believe the code would improve if these issues were addressed as well.

Severity	**Number of issues**	**Number of resolved issues**
High	0	0
Medium	2	1
Low	4	4
Info	11	10

Scope of the Audit

The audit concerns files developed in the following repository under the contracts folder:

- `https://github.com/LimeChain/opus-edu-contracts`

The audit report was based on the following commit from January 22, 2024

```
518ca8cb45df465625e7d51faa637402db9c8c86
```

Methods Used

The contracts were compiled, deployed, and tested in a test environment. Two auditors independently reviewed the code - the current report is the result of combining our findings.

Code Review

We manually inspected the source code to identify potential security flaws.

Automatic analysis

We have used static analysis tools to detect common potential vulnerabilities. No high severity issues were identified with the automated processes. Some low severity issues were found, and we have included them below in the appropriate parts of the report.

Resolution

The team at Kois Center has addressed almost all issues in the following commit on February 5, 2024. We checked the changes and updated the issues below:

`c306135c1a0c434854381ed9f6fa45040bd1dea6`

Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

About Team Omega

Team Omega (teamomega.eth.limo) is a small team that consists of Jelle Gerbrandy and Ben Kaufman. They have been working on blockchain projects, and Solidity in particular, since 2017, and have been doing smart contract audits since 2021.

Severity definitions

High	Vulnerabilities that can lead to loss of assets or data manipulations.
Medium	Vulnerabilities that are essential to fix, but that do not lead to assets loss or data manipulations
Low	Issues that do not represent direct exploit, such as poor implementations, deviations from best practice, high gas costs, etc
Info	Matters of opinion

Findings

General

G1. Missing test coverage [low] [resolved]

The test coverage is high, covering 96% of lines, and a bit under 85% of branches (excluding the mock files), but still does not cover all parts of the code. Also coverage includes the mock file which should be ignored in the configurations.
Recommendation: It is recommended to reach a 100% test coverage. In particular, we urge you to also include a test case that covers the call to `swapStrategy.swapExactOut` which is currently not tested. Also, you should configure the coverage script to ignore the mock folder.
Resolution: This issue was resolved as recommended. Test coverage is now at 100%.

G2. Pin versions of solidity dependencies [info] [resolved]

In `package.json`, a possible range of versions of the OpenZeppelin and Uniswap dependencies is specified rather than a single one:

```
"@openzeppelin/contracts": "^4.7.3",
"@uniswap/v3-periphery": "^1.4.4",
```

This can lead to unexpected problems when a new version of OpenZeppelin or Uniswap is released and other developers (or the continuous integration process, or yourself at a later date) will recompile the contracts with this new version (for example to verify the compiled code on etherscan).
Recommendation: Specify fixed versions of smart contract dependencies in `package.json`, instead of ranges, so that the solidity code can be verified and there is no ambiguity about the actual code you are to deploy or already have deployed.
Severity: Info
Resolution: This issue was resolved as recommended.

G3. Inaccurate error messages [info] [resolved]

- When `takeOut` is called, and the user does have the token, but it was already activated, the error message received will be `TakeOutInsufficientBalance`, which is inaccurate, since it is not the balance that is insufficient, but the error is that the token was already activated. The

error message should be more reflective of the error, like the similar error on transfer saying `CannotTransferATokenThatIsActivated`.

- Calls to `safeTransferFrom` or `safeBatchTransferFrom` will revert with a `TransfersAreDisabled` error message. However, the code defines a `transfer` method (which calls the internal function `_safeTransferFrom`) with which it is possible - in limited cases - to transfer the tokens - so transfers are not disabled. The error message is confusing. Moreover, it is not clear why you break with the ERC1155 standard and define a new `transfer` function, rather than use `safeTransferFrom`.
- When `updateSlippageCoverage` is called, in case the new slippageCoverage is higher than the purchaseFeePercentage, the call will revert with the error `SlippageFeePercentageTooLow`. This error message is misleading, since the issue is that the slippage is too high, or that relatively, the fee percentage is too low.

Recommendation: Update the error messages as described. Also consider staying closer to the ERC1155 standard and implement `safeTransferFrom`.
Severity: Info
Resolution: This issue was resolved as recommended.

G4. Move dependencies to devDependencies and remove unused dependencies [info] [resolved]

In `package.json`, many dependencies which are specified in the `dependencies` section could be moved to the `devDependencies` section. Also various dependencies, such as the `@uniswap/sdk-core`, `@uniswap/v3-core`, and `@uniswap/v3-sdk` are unused and could be removed to minimize dependencies.
Recommendation: Move non-Solidity dependencies to the devDependencies and remove unused dependencies.
Severity: Info
Resolution: This issue was resolved as recommended.

G5. Index relevant event parameters [info] [resolved]

In `ICourseAccess`, `IPaymentProviderConfig`, and `IPaymentProviderEIP712` none of the event parameters are marked as `indexed`. It makes sense to mark event parameters, such as caller addresses and accounts involved, as indexed to allow for efficient filtering when querying the events.
Recommendation: Add indexing of relevant event parameters.
Severity: Info

Resolution: This issue was resolved as recommended.

CourseAccess.sol

CA1. hasBeenActivated can be manipulated with reentrancy in transfer [medium] [resolved]

The `transfer` function performs the transfer of the token before updating the `hasBeenActivated` variable to `true`.
Since the token transfer has a call to the receiver's `onERC1155Received`, the function is susceptible to reentrancy. An example scenario would be user A has the tokens, then uses transfer to send it to a malicious contract B, which reenters the transfer and sends the token back to user A. The result would be that the `hasBeenActivated` for the contract address B will be set to true, even though it no longer has the token. User A could also re-enter by sending the token to themselves, and calling the refund to get the money back, but still have `hasBeenActivated` as true.
Recommendation: Do state changes before external calls. In this case, flip lines 161 and 162. You could also use a reentrancy guard, but that might not be necessary for this contract.
Severity: Medium
Resolution: This issue was resolved as recommended.

PaymentProviderConfig.sol

PPC1. Unnecessary use of override keyword [info] [resolved]

The `override` keyword is used in the definition of all state variables in the contract, but it is not necessary to use and could be removed to reduce clutter of the code.
Recommendation: Remove the use of `override` to declare state variables in the contract.
Severity: Info
Resolution: This issue was resolved as recommended.

PPC2. Remove unused import [info] [resolved]

The contracts contains the following unused import:

```
import {ISwapRouter} from
"@uniswap/v3-periphery/contracts/interfaces/ISwapRouter.sol";
```

Since it is not used anywhere in the contract or its inheritances it can be removed.
Recommendation: Remove the unused import.
Severity: Info
Resolution: This issue was resolved as recommended.

PPC3. staffOnly check not written efficiently [info] [resolved]

The `staffOnly` modifier checks that the caller of a function is either the `owner` or the `purchaseFeeReceiver`. However, it does that by doing the checks and saving the result in memory, which is unnecessary. It could instead be written like the check in line 131 of the `CourseAccess` contract.
Recommendation: Write the check like so:

```
modifier staffOnly() {
      if (msg.sender != owner() && msg.sender != purchaseFeeReceiver) {
            revert NotAuthorized();
      }
      _;
}
```

Severity: Info
Resolution: This issue was resolved as recommended.

SwapStrategy.sol

SS1. Wrong variable name amountReceived in swapExactOut [low] [resolved]

The `swapExactOut` function returns a variable called `amountReceived`, but its value is not of the amount received in the swap, but of the amount used in swapped tokens.
Recommendation: Rename the return variable to `amountUsed` to reflect its actual value.
Severity: Low
Resolution: This issue was resolved as recommended.

PaymentProvider.sol

PP1. Slippage coverage can be used to obtain an unfair discount [medium] [not resolved]

The slippage coverage is configured as the amount which the platform is willing to cover for slippage in the swap of tokens into the desired token, and comes at the expense of platform fees.
This slippage coverage can be abused to obtain an unfair discount in payment. Since the platform is willing to accept any amount which after the conversion will leave the price of the product minus the slippage covered percentage, the buyer can use a token which they can swap for less than the covered slippage. For example, if the desired token is USDC, and covered slippage is 3%, the buyer can use USDT or another stable coin and only send 97.097 USDT, assuming the fee and slippage don't pass the 0.1%, the user will then obtain a discount of nearly the entire slippage coverage of the platform. This incentivizes buyers to use a stablecoin different from the desired currency, making the system unfair and inefficient in incentivizing unnecessary swaps.
Recommendation: As long as the platform is willing to accept less when making a swap, it will be difficult to avoid this kind of unfair discount. One option would be to give users who pay in the desired currency the entire slippage coverage discount, to avoid incentivizing unnecessary swaps. Another option would be to remove the slippage coverage entirely, and have users assume the responsibility for fees and slippage.
Severity: Medium
Resolution: This issue was not resolved.

PP2. Allowed slippage is not enforced [low] [resolved]

Every function that may perform a swap of tokens in the system accepts an `allowedSlippage` variable of type `uint256` as part of the `swapData` parameter. However, the `allowedSlippage` is only used to determine whether the code executes a swap of exact amount in, or a swap with exact amount out. Although the name suggests otherwise, `allowedSlippage` is not used to enforce a maximum slippage for the swap. This could lead users to believe their swap has a slippage protection when doing swap of exact amount in, while it doesn't have any such protection.
Recommendation: Either enforce a slippage protection based on a value given either by the user or an oracle, or remove the parameter. For example, you could remove it and have a simple boolean parameter of `isSlippageAllowed` in the `buy` function.
Severity: Low
Resolution: This issue was resolved as recommended. The `allowedSlippage` parameter was replaced with an `isCustomSlippage` boolean.

PP3. If user sends more tokens then needed when buying a course, these tokens will be lost forever [low] [resolved]

In `_handlePurchasePayment`, the payment for buying tokens is handled. The function will transfer `swapData.amount` tokens of the `swapData.token` contract to the PaymentProvider contract, and, if necessary, swap these tokens to the `desiredCurrency`.
If the `swapData.token == desiredCurrency` and `amount > totalPurchasePrice`, then amount tokens are transferred to the PaymentProvider contract, but only `totalPurchasePrice` tokens are accounted for in l. 357ff. Any surplus tokens will be stuck in the contract.
Recommendation: Either send back the surplus tokens, or account for the tokens in listerRevenue and platformRevenue.
Severity: Low
Resolution: This issue was resolved as recommended. In case more desired tokens were specified by the user in the amount, the extra tokens will now go to the platform fees.

PP4. Update item signature does not commit to specific item [info] [resolved]

The `updateItem` function requires a valid signature from a whitelisted signer to allow a lister to update an item.
However the signature does not commit to a specific `itemId`, which means the lister can update any item of which they are the lister.
Recommendation: If this is intended the code can be left as it is. Yet if the signature is meant to be valid for a specific item, it is important to include the `itemId` in the signature.
Severity: Info
Resolution: This issue was resolved as recommended. The signature now commits to an `itemId`.

PP5. Unnecessary duplication of itemId in memory [info] [resolved]

In line 146 the `itemId` parameter of the `buy` function is saved into another `_itemId` variable. The comment says this is used to avoid a stack too deep error, but that is not the case, and the code works just fine without this unnecessary saving of the variable into a new one.
Recommendation: Remove line 146 and the comment in 145.
Severity: Info
Resolution: This issue was resolved as recommended.

PP6. Unnecessary property “lister” in list function [info] [resolved]

The `ListInputItemData` used only in the `list` function has a `lister` property, which is only allowed to be the `msg.sender`. This means the property could just be removed, and the code could use the `msg.sender` instead.
Recommendation: Remove `lister` from the `ListInputItemData` struct and use `msg.sender` instead.
Severity: Info
Resolution: This issue was resolved as recommended.

PP7. Remove unused imports [info] [partially resolved]

The contracts contains the following unused imports:

```
import {ISwapRouter} from
"@uniswap/v3-periphery/contracts/interfaces/ISwapRouter.sol";
import {IPeripheryPayments} from
"@uniswap/v3-periphery/contracts/interfaces/IPeripheryPayments.sol";
import "./interfaces/ISwapStrategy.sol";
```

Since these are not used anywhere in the contract they can be removed.
Recommendation: Remove the unused imports.
Severity: Info
Resolution: This issue was partially resolved. The imports for `ISwapRouter` and `IPeripheryPayments` were removed, but the import of `ISwapStrategy` remains.