Smart Contract Security Audit Report

Gearbox Pendle&Mellow Integrations

1. Contents

2. General Information

This report contains information about the results of the security audit of the Gearbox (hereafter referred to as "Customer") smart contracts, conducted by Decurity in the period from 18/08/2024 to 20/08/2024.

2.1. Introduction

Tasks solved during the work are:

- Review the protocol design and the usage of 3rd party dependencies,
- Audit the contracts implementation,
- Develop the recommendations and suggestions to improve the security of the contracts.

2.2. Scope of Work

The audit scope included the contracts in the following repositories: integrations-v3 and oracles-v3. Initial review was done for the commits e620d8 and fba278. Retesting was performed for commits e0134f and e23574. The testing for version 3.1 was completed for commits d877d1 and a2c632. After the re-testing Gearbox has introduced minor changes, suggested and audited by Watchpug; the corresponding commit hashes are fc8d3a and 9e56cb.

The following contracts have been tested:

- integrations-v3/contracts/adapters/pendle/PendleRouterAdapter.sol
- integrations-v3/contracts/adapters/mellow/MellowVaultAdapter.sol
- integrations-v3/contracts/adapters/mellow/Mellow4626VaultAdapter.sol
- oracles-v3/contracts/oracles/pendle/PendleTWAPPTPriceFeed.sol

2.3. Threat Model

The assessment presumes actions of an intruder who might have capabilities of any role (an external user, token owner, token service owner, a contract). The centralization risks have not been considered upon the request of the Customer.

The main possible threat actors are:

- User,
- Protocol owner,
- Liquidity Token owner/contract.

The table below contains sample attacks that malicious attackers might carry out.

Table. Theoretically possible attacks

Attack	Actor
Contract code or data hijacking *Deploying a malicious contract or submitting malicious data*	Contract owner Token owner
Financial fraud *A malicious manipulation of the business logic and balances, such as a re-entrancy attack or a flash loan attack*	Anyone
Attacks on implementation *Exploiting the weaknesses in the compiler or the runtime of the smart contracts*	Anyone

2.4. Weakness Scoring

An expert evaluation scores the findings in this report, an impact of each vulnerability is calculated based on its ease of exploitation (based on the industry practice and our experience) and severity (for the considered threats).

2.5. Disclaimer

Due to the intrinsic nature of the software and vulnerabilities and the changing threat landscape, it cannot be generally guaranteed that a certain security property of a program holds.

Therefore, this report is provided “as is” and is not a guarantee that the analyzed system does not contain any other security weaknesses or vulnerabilities. Furthermore, this report is not an endorsement of the Customer’s project, nor is it an investment advice.

That being said, Decurity exercises best effort to perform their contractual obligations and follow the industry methodologies to discover as many weaknesses as possible and maximize the audit coverage using the limited resources.

3. Summary

As a result of this work, we have discovered several Medium-level issues.

The other suggestions included fixing the low-risk issues and some best practices (see Security Process Improvement).

The Gearbox team has given the feedback for the suggested changes and explanation for the underlying code.

3.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of August 28, 2024.

Table. Discovered weaknesses

Issue	Contract	Risk Level	Status
Missing YT token validation	integrations-v3/contracts/adapters/pendle/PendleRouterAdapter.sol	Medium	Fixed

Issue	Contract	Risk Level	Status
Lack of deadline check	integrations-v3/contracts/adapters/pendle/PendleRouterAdapter.sol	**Medium**	Acknowledged
Potential overflow	integrations-v3/contracts/adapters/mellow/MellowVaultAdapter.sol	**Medium**	Fixed
Underlying tokens are not validated	integrations-v3/contracts/adapters/mellow/MellowVaultAdapter.sol	**Low**	Fixed
Call can be executed without the found token	integrations-v3/contracts/adapters/mellow/MellowVaultAdapter.sol	**Low**	Fixed
Lack of array length check	integrations-v3/contracts/adapters/mellow/MellowVaultAdapter.sol	**Low**	Fixed
Unused variable	integrations-v3/contracts/adapters/pendle/PendleRouterAdapter.sol	**Info**	Fixed
Wrong comments	integrations-v3/contracts/adapters/pendle/PendleRouterAdapter.sol oracles-v3/contracts/oracles/pendle/PendleTWAPPTPriceFeed.sol:	**Info**	Fixed

4. General Recommendations

This section contains general recommendations on how to improve overall security level.

The Findings section contains technical recommendations for each discovered issue.

4.1. Security Process Improvement

The following is a brief long-term action plan to mitigate further weaknesses and bring the product security to a higher level:

- Keep the whitepaper and documentation updated to make it consistent with the implementation and the intended use cases of the system,
- Perform regular audits for all the new contracts and updates,
- Ensure the secure off-chain storage and processing of the credentials (e.g. the privileged private keys),
- Launch a public bug bounty campaign for the contracts.

5. Findings

5.1. Missing YT token validation

Risk Level: Medium

Status: Fixed in the commit 6c746c.

Contracts:

- integrations-v3/contracts/adapters/pendle/PendleRouterAdapter.sol

Description:

The `redeemPyToToken()` and `redeemDiffPyToToken()` functions use the YT (Yield Token) but do not verify that it was created by Pendle Finance. This oversight allows an attacker to pass a fake YT token that returns the correct PT (Principal Token) address and successfully passes all checks on the Pendle Finance side, leading to potential unexpected behavior within the system.

Additionally, there is a second issue related to how the Pendle Finance `redeemPyToToken()` call is executed. During this call, PT tokens may be transferred to the YT contract, which controls the amount of SY (Standardized Yield) token that should be received. This could create an opportunity for a malicious YT token to steal PT tokens by not sending the correct amount of tokens to the receiver. If an attacker specifies a malicious YT token and an incorrect expected balance during the call to the CreditFacade, there is a potential risk of losing tokens.

Remediation:

Consider implementing a check to ensure that the YT token passed to the `redeemPyToToken()` and `redeemDiffPyToToken()` functions was indeed created by Pendle Finance.

5.2. Lack of deadline check

Risk Level: Medium

Status: Acknowledged

Contracts:

- integrations-v3/contracts/adapters/pendle/PendleRouterAdapter.sol

Description:

The contract does not allow users to submit a deadline for their actions. This missing feature enables pending transactions to be maliciously executed later. Swap should provide their users with an option to limit the execution of their pending actions(swaps). The most common solution is to include a deadline timestamp as a parameter (for example see Uniswap V2). If such an option is not present, users can unknowingly perform bad trades. Since it is not built into the `PendleRouter` it can be built on top of it. Introducing it in the `PendleRouterAdapter` will secure users from performing bad trades.

Remediation:

Consider adding the deadline to the `swapExactTokenForPt()`:

```
function swapExactTokenForPt(
        address,
        address market,
        uint256 minPtOut,
        ApproxParams calldata guessPtOut,
        TokenInput calldata input,
        LimitOrderData calldata,
+        uint256 deadline
    ) external creditFacadeOnly returns (uint256 tokensToEnable, uint256
tokensToDisable) {
+       require(deadline <= block.timestamp);
        (, address pt,) = IPendleMarket(market).readTokens();
        //@audit-issue add deadline for a case when tx got stuck

        if (isPairAllowed[market][input.tokenIn][pt] != PendleStatus.ALLOWED)
revert PairNotAllowedException();

        address creditAccount = _creditAccount();

        LimitOrderData memory limit;

        TokenInput memory input_m;

        {
            input_m.tokenIn = input.tokenIn;
            input_m.netTokenIn = input.netTokenIn;
            input_m.tokenMintSy = input.tokenIn;
        }

        (tokensToEnable, tokensToDisable,) = _executeSwapSafeApprove(
            input_m.tokenIn,
            pt,
```

```
            abi.encodeCall(
                IPendleRouter.swapExactTokenForPt, (creditAccount, market,
minPtOut, guessPtOut, input_m, limit)
            ),
            false
        );
    }
```

5.3. Potential overflow

Risk Level: Medium

Status: Fixed in the commit e1ccfb.

Contracts:

- integrations-v3/contracts/adapters/mellow/MellowVaultAdapter.sol

Description:

The `amount * rateMinRAY` can be overflowed.

```
function depositOneAssetDiff(address asset, uint256 leftoverAmount, uint256
rateMinRAY, uint256 deadline)
        external
        creditFacadeOnly
        returns (uint256 tokensToEnable, uint256 tokensToDisable)
    {
        if (!isUnderlyingAllowed[asset]) revert
UnderlyingNotAllowedException(asset);

        address creditAccount = _creditAccount();

        uint256 balance = IERC20(asset).balanceOf(creditAccount);
        if (balance > leftoverAmount) {
            unchecked {
                uint256 amount = balance - leftoverAmount;

                // @audit should be exclude from unchecked
                (tokensToEnable, tokensToDisable) = _depositOneAsset(
                    creditAccount, asset, amount, amount * rateMinRAY / RAY,
deadline, leftoverAmount <= 1
                );
            }
        }
    }
```

Remediation:

Consider excluding the operation from `unchecked` block.

5.4. Underlying tokens are not validated

Risk Level: Low

Status: Fixed in the commit 247e7b.

Contracts:

- integrations-v3/contracts/adapters/mellow/MellowVaultAdapter.sol

Description:

The `deposit()` function performs sanity checks on the underlyings tokens and their corresponding amounts to ensure that only allowed tokens are processed. However, the current implementation includes a condition that allows tokens with an amount of zero to bypass the `isUnderlyingAllowed` check:

```
address[] memory underlyings =
IMellowVault(targetContract).underlyingTokens();
uint256 len = underlyings.length;
if (amounts.length != len) revert IncorrectArrayLengthException(); // U:[MEL-
3]
for (uint256 i = 0; i < len;) {
    if (amounts[i] > 0 && !isUnderlyingAllowed[underlyings[i]]) {
        revert UnderlyingNotAllowedException(underlyings[i]); // U:[MEL-3]
    }
    unchecked {
        ++i;
    }
}
_approveAssets(underlyings, amounts, type(uint256).max);
```

This condition (`amounts[i] > 0`) could allow a malicious actor to bypass the `isUnderlyingAllowed` check by passing a zero amount for a token, potentially leading to unintended behavior within the system during the approve call of a non-permitted underlying token.

While it's true that the underlying tokens can only be set by the admin of the Mellow Vault and that a zero amount might only pass without issue if the oracle ratios are zero, this still presents a potential vulnerability.

Remediation:

To ensure that all underlying tokens are properly validated, consider removing the `amounts[i] > 0` check. This will enforce the `isUnderlyingAllowed` check for all tokens, regardless of the amount specified.

5.5. Call can be executed without the found token

Risk Level: Low

Status: Fixed in the commit 6c746c.

Contracts:

- integrations-v3/contracts/adapters/mellow/MellowVaultAdapter.sol

Description:

If the token was not found among the underlyings, the call will still be executed.

```
function _depositOneAsset(
        address creditAccount,
        address asset,
        uint256 amount,
        uint256 minLpAmount,
        uint256 deadline,
        bool disableTokenIn
    ) internal returns (uint256 tokensToEnable, uint256 tokensToDisable) {
        address[] memory underlyings =
IMellowVault(targetContract).underlyingTokens();
        uint256 len = underlyings.length;

        uint256[] memory amounts = new uint256[](len);

        for (uint256 i = 0; i < len;) {
            if (underlyings[i] == asset) {
                amounts[i] = amount;
                break;
            }

            unchecked {
                ++i;
            }
        }
        // @audit Lack of check for not found token
        _approveToken(asset, type(uint256).max);
        _execute(abi.encodeCall(IMellowVault.deposit, (creditAccount, amounts,
```

```
minLpAmount, deadline)));
        _approveToken(asset, 1);
```

Remediation:

Consider reverting a transaction if the token was not found.

5.6. Lack of array length check

Risk Level: Low

Status: Fixed in the commit 6c746c.

Contracts:

- integrations-v3/contracts/adapters/mellow/MellowVaultAdapter.sol

Description:

It is possible to call the function by presenting an array of a longer length than the vault returns.

```
function deposit(address, uint256[] memory amounts, uint256 minLpAmount,
uint256 deadline)
        external
        creditFacadeOnly
        returns (uint256 tokensToEnable, uint256 tokensToDisable)
    {
        address creditAccount = _creditAccount();

        address[] memory underlyings =
IMellowVault(targetContract).underlyingTokens();

        uint256 len = underlyings.length;
        // @audit Lack of array length check

        for (uint256 i = 0; i < len;) {
            if (amounts[i] > 0 && !isUnderlyingAllowed[underlyings[i]]) {
                revert UnderlyingNotAllowedException(underlyings[i]);
            }

            unchecked {
                ++i;
            }
        }

        _approveAssets(underlyings, amounts, type(uint256).max);
        _execute(abi.encodeCall(IMellowVault.deposit, (creditAccount, amounts,
minLpAmount, deadline)));
        _approveAssets(underlyings, amounts, 1);
```

Remediation:

Consider adding an additional array length check.

```
function deposit(address, uint256[] memory amounts, uint256 minLpAmount,
uint256 deadline)
        external
        creditFacadeOnly
        returns (uint256 tokensToEnable, uint256 tokensToDisable)
    {
        address creditAccount = _creditAccount();

        address[] memory underlyings =
IMellowVault(targetContract).underlyingTokens();

        uint256 len = underlyings.length;
+               require(len == amounts.length, "the lengths do not match");

        for (uint256 i = 0; i < len;) {
            if (amounts[i] > 0 && !isUnderlyingAllowed[underlyings[i]]) {
                revert UnderlyingNotAllowedException(underlyings[i]);
            }

            unchecked {
                ++i;
            }
        }

        _approveAssets(underlyings, amounts, type(uint256).max);
        _execute(abi.encodeCall(IMellowVault.deposit, (creditAccount, amounts,
minLpAmount, deadline)));
        _approveAssets(underlyings, amounts, 1);
```

5.7. Unused variable

Risk Level: Info

Status: Fixed in the commit 6c746c.

Contracts:

- integrations-v3/contracts/adapters/pendle/PendleRouterAdapter.sol

Description:

The `limit` variable is never used:

```
integrations-v3/contracts/adapters/pendle/PendleRouterAdapter.sol:
  316:         LimitOrderData memory limit;
```

Remediation:

Consider removing it.

5.8. Wrong comments

Risk Level: Info

Status: Fixed in the commits 6c746cand e23574.

Contracts:

- integrations-v3/contracts/adapters/pendle/PendleRouterAdapter.sol
- oracles-v3/contracts/oracles/pendle/PendleTWAPPTPriceFeed.sol

Description:

The comment associated with the `isPairAllowed` mapping is incorrect. The comment suggests that the mapping uses `tokenIn`, but in practice, it may use either `tokenId` or `tokenOut`.

```
integrations-v3/contracts/adapters/pendle/PendleRouterAdapter.sol:
  40:     /// @notice Mapping from (market, tokenIn, pendleToken) to whether
swaps are allowed, and which directions
  41:     mapping(address => mapping(address => mapping(address =>
PendleStatus))) public isPairAllowed;
```

The comment states that the `latestRoundData()` function returns the last update timestamp, but the implementation does not return this value.

```
oracles-v3/contracts/oracles/pendle/PendleTWAPPTPriceFeed.sol:
    81:     /// @notice Returns the USD price of the PT token with 8 decimals
and the last update timestamp
  82:     function latestRoundData() external view override returns (uint80,
int256, uint256, uint256, uint80) {
  83:         int256 answer = _getValidatedPrice(priceFeed, stalenessPeriod,
skipCheck);
  84:
  85:         if (expiry > block.timestamp) {
  86:             answer = int256(FixedPoint.mulDown(uint256(answer),
_getPTToAssetRate()));
  87:         }
  88:
```

```
89:         return (0, answer, 0, 0, 0);
90:     }
```

Remediation:

Update the comments to correctly reflect the purpose and usage of the `isPairAllowed` mapping and `latestRoundData()` function. This will improve code clarity and accuracy.

6. Appendix

6.1. About us

The Decurity team consists of experienced hackers who have been doing application security assessments and penetration testing for over a decade.

During the recent years, we've gained expertise in the blockchain field and have conducted numerous audits for both centralized and decentralized projects: exchanges, protocols, and blockchain nodes.

Our efforts have helped to protect hundreds of millions of dollars and make web3 a safer place.