·Decurity·

THALES

# Smart Contract Security Audit Report

---

## Thales Market LiquidityPool

Decurity, 2023

# Contents

# 1. General Information

This report contains information about the results of the security audit of the Thales Market (hereafter referred to as “Customer”) LiquidityPool smart contracts, conducted by Decurity in the period from 04/17/2023 to 04/25/2023.

## 1.1. 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.

## 1.2. Scope of Work

The audit scope included the contracts in the following repository subdirectory: https://github.com/thales-markets/contracts/tree/main/contracts/SportMarkets/LiquidityPool.

The initial review was done for the commit 147548f81ef273988fb278d11c4288a5051944fe. The final review was done for the commit edc677cb74635cf22f43244234f103eeef3084aa.

## 1.3. Threat Model

The assessment presumes actions of an intruder who might have capabilities of any role. The main possible external threat actors are protocol users, oracles, and privileged addresses, such as owner or AMM.

It’s worth noting that the protocol is prone to the centralization risks by design because of the nature of the business logic. However, the primary goal of this review was

the vulnerability assessment in terms of the protocol implementation rather than the risk assessment.

## 1.4. 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).

## 1.5. 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.

# 2. Summary

As a result of this work, we have discovered a single high-risk security issue that could lead to an unintentional loss of funds.

The other suggestions included fixing the low-risk issues and some best practices (see 3.1).

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

## 2.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of Apr 28, 2023.

*Table. Discovered weaknesses*

| Issue | Contracts | Risk Level | Status |
|---|---|---|---|
| needsTransformingCollateral can be set during an active round | SportsAMMLiquidityPool.sol | High | Fixed |
| Arbitrary token transfer (centralization risks) | SportAMMLiquidityPool.sol | Medium | Fixed |
| Functions lack address(0) check | SportAMMLiquidityPool.sol | Low | Fixed |
| getRoundEndTime and getRoundStartTime may return incorrect time | SportAMMLiquidityPool.sol | Low | Won't fix |

| | | | |
|---|---|---|---|
| Round pools may be created with incorrect times | SportAMMLiquidityPool.sol | **Low** | Fixed |
| No events on state changes | SportAMMLiquidityPool.sol | **Low** | Fixed |
| Implementations are not initialized at the deployment | DefaultLiquidityProvider.sol, SportAMMLiquidityPool.sol | **Info** | Won't fix |
| Unnecessary variables initialization with default values | SportAMMLiquidityPoolRound.sol | **Info** | Fixed |
| Suboptimal if statements | SportAMMLiquidityPool.sol | **Info** | Won't fix |
| Unnecessary checked arithmetic in loops | SportAMMLiquidityPool.sol | **Info** | Won't fix |
| Suboptimal state variable reads and writes | (Multiple) | **Info** | Won't fix |
| Incorrect comment in getOrCreateMarketPool | SportAMMLiquidityPool.sol | **Info** | Fixed |
| Incorrect require error in transferTokens | SportAMMLiquidityPool.sol | **Info** | Fixed |
| Potential transfer of 0 tokens in commitTrade | SportAMMLiquidityPool.sol | **Info** | Fixed |

| | | | |
|---|---|---|---|
| Lack of zero address check when calling stakingThales | SportsAMMLiquidityPool.sol | **Info** | Fixed |

# 3. General Recommendations

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

To mitigate the current weaknesses, follow the technical recommendations for each finding in section 4.

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.

# 4. Findings

## 4.1. needsTransformingCollateral can be set during an active round

**Risk Level**: High

**Status**: Fixed in the commit edc677cb. Method to set the variable was removed and the variable was added to the initializer logic.

**Contracts**:

- SportsAMMLiquidityPool.sol

**Location**: Lines: 589. Function: `setNeedsTransformingCollateral(bool _needsTransformingCollateral)`.

**Description**:

The following function can be called during an active round.

```
function setNeedsTransformingCollateral(bool _needsTransformingCollateral)
external onlyOwner {
    needsTransformingCollateral = _needsTransformingCollateral;
}
```

This will affect logic in every place where `_transformCollateral` is called. Also, `needsTransformingCollateral` is not set in the `initialize` function, so there is a probability that a malicious user may front-run this function call and call deposit before it was not applied. They will be able to gain an extra amount available for deposit than others in case `needsTransformingCollateral` was going to be set to true.

**Remediation**:

Consider setting `needsTransformingCollateral` value in `initializer`.

Also consider removing the setter from the contract because calling it by mistake might have a very negative impact.

## 4.2. Arbitrary token transfer (centralization risks)

**Risk Level**: Medium

**Status**: Fixed in the commit edc677cb. Admin transfer methods were added as a failsafe in case ownership of the proxy is lost by mistake or that an urgent action needs be taken to prevent further damage in case an exploit is suspected. Those methods are removed now.

**Contracts**:

- SportsAMMLiquidityPool.sol

**Location**: Functions: `transferTokens` and `transferTokensFromLiquidityPool`.

**Description**:

The pool contract contains functions `transferTokens` and `transferTokensFromLiquidityPool` that allow an owner to withdraw any tokens from the contract.

These functions are intended for emergency situations as a failsafe.

However, they pose a risk if there are no limitations on the owner's actions. Precisely, the current owner of the liquidity pool on Arbitrum (https://arbiscan.io/address/0x8e9018b48456202aa9bb3e485192b8475822b874) is not a timelock contract but rather a gnosis safe (https://arbiscan.io/address/0x0089282ac624bbbda7ec472d63ee99643d7edf4a).

**Remediation**:

Consider creating a separate role for the critical actions to provide better transparency and decentralization.

## 4.3. Functions lack address(0) check

**Risk Level**: Low

**Status**: Fixed in the commit edc677cb. Checks have been added.

**Contracts**:

- SportAMMLiquidityPool.sol

**Location**: Lines: `594`, `636`. Function: `setPoolRoundMastercopy(address _poolRoundMastercopy) & setSportAmm(ISportsAMM _sportAMM)`.

**Description**:

The following functions lack the `address(0)` check:

```
setPoolRoundMastercopy(address _poolRoundMastercopy) - line 594
setSportAmm(ISportsAMM _sportAMM) - line 636
```

However their arguments are not supposed to equal `address(0)`.

**Remediation**:

Consider adding the `address(0)` check.

## 4.4. getRoundEndTime and getRoundStartTime may return incorrect time

**Risk Level**: Low

**Status**: Won't fix. The Customer believes that this finding is incorrect. The round times are statically defined as soon as `firstRoundStartTime` is set, and trading in a futuristic round can take place before that round has begun. Closing the round does not have a relation to `RoundEndTime`, although a different naming of variables and methods could alleviate this semantic misunderstanding.

**Contracts**:

- SportAMMLiquidityPool.sol

**Location**: Lines: `499`. Function: `getRoundEndTime & getRoundStartTime`.

**Description**:

The following functions rely on a fact that every round has a constant `roundLength`:

```
function getRoundStartTime(uint _round) public view returns (uint) {
    return firstRoundStartTime + (_round - 1) * roundLength;
}
```

```
function getRoundEndTime(uint _round) public view returns (uint) {
    return firstRoundStartTime + _round * roundLength;
}
```

However, that is actually not true because round is incremented only when it has been closed via the `closeRound()` function on line 345. This means that the actual time when the round will end may not equal to `roundLength`.

This results in the fact that these 2 functions may return incorrect values in case if `closeRound()` was called some time after `roundLength` has passed. As a result, these two functions will return incorrect values, which will also affect the round pool's `roundStartTime` and `roundEndTime`.

**Remediation**:

Probably a new variable which will contain the actual round end time should be introduced and these two functions should rely on it.

## 4.5. Round pools may be created with incorrect times

**Risk Level**: Low

**Status**: Fixed in the commit edc677cb. Fixed by adding the method to update `roundTimes` and using in the `start` method.

**Contracts**:

- SportAMMLiquidityPool.sol

**Location**: Lines: `567`. Function: `_getOrCreateRoundPool(uint _round)`.

**Description**:

Pool for the first round may be created before the start. As a result, the `getRoundEndTime` calls on the line 567 will return an incorrect value because `firstRoundStartTime` is not yet set:

```
newRoundPool.initialize(address(this), sUSD, _round,
getRoundEndTime(_round), getRoundEndTime(_round + 1));
```

As a result, the created round pool will have incorrect `roundStartTime` and `roundEndTime` values.

**Remediation**:

The first round pool can be created with `block.timestamp`. Another option would be to introduce a setter on a round pool contract which will update times and will be called in the start function.

## 4.6. No events on state changes

**Risk Level**: Low

**Status**: Fixed in the commit edc677cb. All events have been added.

**Contracts**:

- SportAMMLiquidityPool.sol

**Description**:

The following functions perform state changes and are important parts of logic, however, they don't emit an event:

```
prepareRoundClosing()
setOnlyWhitelistedStakersAllowed(bool flagToSet)
setNeedsTransformingCollateral(bool _needsTransformingCollateral)
```

**Remediation**:

Consider adding events to the listed functions.

## 4.7. Implementations are not initialized at the deployment

**Risk Level**: Info

**Status**: Won't fix. As upgrading the dependency affects other audited contracts too, the Customer chooses to not add it immediately. As they use transparent proxies, they don't need to worry about initializing implementation for any other reason but optics.

**Contracts:**

- DefaultLiquidityProvider.sol
- SportAMMLiquidityPool.sol

**Description**:

After the deployment of `DefaultLiquidityProvider` and `SportAMMLiquidityPool` contracts via proxies, their implementations are still available for initialization. In some cases, such uninitialized implementations can be taken over by an attacker for the purpose of calling `selfdestruct` on them.

**Remediation**:

It is recommended to automatically lock the implementation contracts by calling the function `_disableInitializers()` in the constructor:

```
/// @custom:oz-upgrades-unsafe-allow constructor
constructor() {
    _disableInitializers();
}
```

## 4.8. Unnecessary variables initialization with default values

**Risk Level**: Info

**Status**: Fixed in the commit edc677cb.

**Contracts**:

- SportAMMLiquidityPoolRound.sol

**Location**: Lines: 25.

**Description**:

Uninitialized variables are assigned with the type's default value. Explicitly initializing a variable with its default value costs unnecessary gas

Proof (SportAMMLiquidityPoolRound.sol:25):

```
bool public initialized = false;
```

**Remediation**:

Do not initialize variables with default values.

## 4.9. Suboptimal if statements

**Risk Level**: Info

**Status**: Won't fix. Noted, but won't fix immediately as these gas savings are non-material on Optimistic Rollups.

**Contracts**:

- SportAMMLiquidityPool.sol

**Description**:

Using nested `if` statements is cheaper than using `&&` multiple check combinations. There are more advantages, such as easier to read code and better coverage reports.

Examples of such suboptimal code locations:

- SportAMMLiquidityPool.sol:138-142
- SportAMMLiquidityPool.sol:212-214
- SportAMMLiquidityPool.sol:327-333

**Remediation**:

Use nested `if`.

## 4.10. Unnecessary checked arithmetic in loops

**Risk Level**: Info

**Status**: Won't fix. Noted, but won't fix immediately as these gas savings are non-material on Optimistic Rollups.

**Contracts**:

- SportAMMLiquidityPool.sol

**Description**:

A lot of times there is no risk that the loop counter can overflow. Using javascript's unchecked block saves the overflow checks.

**Remediation**:

Replace `i++` with `unchecked { ++i; }` in the following functions in `SportAMMLiquidityPool` to optimize the gas usage:

- `processRoundClosingBatch`
- `exerciseMarketsReadyToExercised`

Also, replace `index++` in the following functions:

- `transferTokensFromLiquidityPool`
- `transferTokens`
- `setWhitelistedAddresses`
- `setWhitelistedStakerAddresses`

## 4.11. Suboptimal state variable reads and writes

**Risk Level**: Info

**Status**: Won't fix. Noted, but won't fix immediately as these gas savings are non-material on Optimistic Rollups.

**Description**:

The state variable `round` is accessed multiple times without caching in following functions in `SportAMMLiquidityPool`:

- `exerciseMarketsReadyToExercised`
- `processRoundClosingBatch`
- `closeRound`
- `withdrawalRequest`.

**Remediation**:

In each of the mentioned functions, create a local variable `round` and assign it the value of the corresponding state variable `round`. Replace all subsequent reads and writes of `round` with the local variable. This will reduce the number of expensive SSTORE operations and improve gas efficiency.

## 4.12. Incorrect comment in getOrCreateMarketPool

**Risk Level**: Info

**Status**: Fixed in the commit edc677cb.

**Contracts**:

- SportAMMLiquidityPool.sol

**Location**: Lines: 247.

**Description**:

Comment on the line 247 before the function `getOrCreateMarketPool` is incorrect and was copy-pasted from another function:

```
/// @notice request withdrawal from the LP
```

**Remediation**:

Rewrite the comment.

## 4.13. Incorrect require error in transferTokens

**Risk Level**: Info

**Status**: Fixed in the commit edc677cb. The method was removed altogether in 4.2.

**Contracts**:

- SportAMMLiquidityPool.sol

**Location**: Lines: 695. Function: `transferTokens(address[], address, uint, bool)`.

**Description**:

The following assertion error message on line 695 in the `transferTokens` function in `SportAMMLiquidityPool` is incorrect:

```
require(tokens.length > 0, "Whitelisted addresses cannot be empty");
```

**Remediation**:

Specify a correct error message.

## 4.14. Potential transfer of 0 tokens in commitTrade

**Risk Level**: Info

**Status**: Fixed in the commit edc677cb.

**Contracts**:

- SportAMMLiquidityPool.sol

**Location**: Lines: `180`. Function: `commitTrade`.

**Description**:

Function performs check that `poolBalance > amountToMint` on line 180. In case `amountToMint` equals `poolBalance`, the function will call `_depositAsDefault` with 0 value:

```
uint poolBalance = sUSD.balanceOf(liquidityPoolRound);
if (poolBalance > amountToMint) {
    sUSD.safeTransferFrom(liquidityPoolRound, address(sportsAMM),
amountToMint);
} else {
    uint differenceToLPAsDefault = amountToMint - poolBalance;
    _depositAsDefault(differenceToLPAsDefault, liquidityPoolRound,
marketRound);
    sUSD.safeTransferFrom(liquidityPoolRound, address(sportsAMM),
amountToMint);
}
```

**Remediation**:

The check on line 180 should be ≥ instead of >.

## 4.15. Lack of zero address check when calling stakingThales

**Risk Level**: Info

**Status**: Fixed in the commit edc677cb.

**Contracts**:

- SportsAMMLiquidityPool.sol

**Location**: Lines: 130. Function: `deposit(uint amount)`.

**Description**:

Deposit function performs a call to `stakingThales` contract on the line 130, however it checks that `stakingThales` is not `address(0)` only on the line 149. This may lead to the situation when non-whitelisted users will try to call the deposit function and catch a revert:

```
require(
    (balancesPerRound[round][msg.sender] + amount +
balancesPerRound[nextRound][msg.sender]) <=
      _transformCollateral((stakingThales.stakedBalanceOf(msg.sender) *
stakedThalesMultiplier) / ONE),
  "Not enough staked THALES"
);
```

```
if (address(stakingThales) != address(0)) {
    stakingThales.updateVolume(msg.sender, amount);
}
```

**Remediation**:

Zero address check should be one of the requirements on line 128.

# 5. Appendix

## 5.1. 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.