TapiocaDAO StargateV2 Audit Report

Version 1.0

Windhustler

TapiocaDAO StargateV2 Audit Report

Windhustler

August 15, 2022

Introduction

A time-boxed security review of the **TapiocaDAO StargateV2** integration was done by **Windhustler**, focusing on the security aspects of the smart contracts.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource, and expertise-bound effort where I try to find as many vulnerabilities as possible. I can not guarantee 100% security after the review or even if the review will find any vulnerabilities. Subsequent security reviews, bug bounty programs, and on-chain monitoring are recommended.

About Windhustler

Windhustler is an independent smart contract security researcher. Having extensive experience in developing and managing DeFi projects holding millions in TVL, he is putting his best efforts into security research & reviews. Check his previous work here or reach out on X @windhustler.

About TapiocaDAO

TapiocaDAO is a decentralized autonomous organization (DAO) which created a decentralized Omnichain stablecoin ecosystem, comprised of multiple sub-protocols, which includes; Singularity, the

first-ever Omnichain isolated money market, Big Bang, an Omnichain CDP Stablecoin Creation Engine, Yieldbox, the most powerful token vault ever created, tOFT (Tapioca Omnichain Wrapper[s]) which transforms any fragmented asset into a unified Omnichain asset, twAML, an economic incentive consensus mechanism, and Pearlnet, the self-sovereign Omnichain verifier network.

Severity classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact - The technical, economic, and reputation damage from a successful attack

Likelihood - The chance that a particular vulnerability gets discovered and exploited

Severity - The overall criticality of the risk

Informational - Findings in this category are recommended changes for improving the structure, usability, and overall effectiveness of the system.

Security Assessment Summary

***review commit hash* - 84bd33b12140398c7f23d08b08cfdba7b1fc3421**

Scope

The following smart contracts were in the scope of the audit:

- `contracts/StargateV2Strategy/StargateV2Strategy.sol`

Findings Summary

ID	Title	Severity	Status
H-1	`_withdraw` can be permanently DoS-ed for pools with `convertRate != 1`	High	-
M-1	Setting new `StargatePool` address makes strategy unusable	Medium	Fixed
M-2	`pendingRewards` assumes that ARB/STG are always present as rewards	Medium	Fixed
M-3	`ITapiocaOracle.peek` call should return manipulation resistant prices and check staleness	Medium	Fixed
M-4	`_currentBalance` doesn't consider harvested rewards	Medium	Ack
L-1	`setFarm` doesn't collect the rewards and withdraw staked LPTokens	Low	Fixed
L-2	reward tokens should be swapped into `inputToken` during invest	Low	Fixed
L-3	`_currentBalance` function returns an approximate amount of token in the strategy	Low	-
L-4	`clearAllowance` call is redundant	Low	Fixed

Detailed Findings

H-1 `_withdraw` can be permanently DoS-ed for pools with `convertRate != 1`

Context

- StargateV2Strategy.sol#L353-L392

Description

`StargateV2Strategy.withdraw()` is used to withdraw tokens from the farm, and wrap them into TOFT so the user withdrawing from the `YieldBox` can get his tokens back.

It does so by first checking the available balance of TOFT in the strategy, and then it only pulls the difference needed.

```
1          uint256 assetInContract = IERC20(contractAddress).balanceOf(
               address(this));
2
3          //  check first if `contractAddress` is already available
               without performing any withdrawal action
4          uint256 toWithdrawFromPool;
5          unchecked {
6              toWithdrawFromPool = amount > assetInContract ? amount -
                   assetInContract : 0; // Asset to withdraw from the pool
                   if not enough available in the contract
7          }
```

After it determines the `toWithdrawFromPool` amount it makes the assumption that this exact amount can be withdrawn from the farm and redeemed from the pool.

By taking a look at the `StargatePool.redeem()` contract logic:

```
1      /// @notice Redeem the LP token of the sender and return the
           underlying token to receiver
2      /// @dev Emits Redeemed when the LP tokens are redeemed
           successfully.
3      /// @dev Reverts if the sender does not hold enough LP tokens or if
            the pool does not have enough credit.
4      /// @param _amountLD The amount of LP token to redeem in LD
5      /// @param _receiver The account to which to return the underlying
           tokens
6      /// @return amountLD The amount of LP token burned and the amount
           of underlying token sent to the receiver
7      function redeem(uint256 _amountLD, address _receiver) external
           nonReentrantAndNotPaused returns (uint256 amountLD) {
8          uint64 amountSD = _ld2sd(_amountLD);
9          paths[localEid].decreaseCredit(amountSD);
10
11         // de-dust LP token
12         amountLD = _sd2ld(amountSD);
13         // burn LP token. Will revert if the sender doesn't have enough
               LP token
14         lp.burnFrom(msg.sender, amountLD);
15         tvlSD -= amountSD;
16
17         // send the underlying token from the pool to the receiver
18         _safeOutflow(_receiver, amountLD);
19         _postOutflow(amountSD); // decrease the pool balance
20
21         emit Redeemed(msg.sender, _receiver, amountLD);
22     }
```

It can be observed that `_amountLD` passed as parameter is not always equal to the amount of underlying token transferred out. A small difference can occur due to conversion to `sharedDecimals`.

This property opens up a DoS attack vector on the withdraw function.

Consider the following example:

- There is 1e18 TOFT sitting in the `StargateV2Strategy` contract.
- A user initiates a withdrawal for an amount equal to `2e18` inside the YieldBox.
- A griever notices this and frontruns his transaction donating 1 wei of TOFT to the `StargateV2Strategy` contract.
- As his transaction gets executed toWithdrawFromPool = 2e18 - 1e18 - 1 wei ~ 999999999999999999.
- If `convertRate != 1` for the `StargatePool` it's not possible to redeem this exact amount as dust gets removed from the input amount. The amount of redeemed underlying token will be equal to 999999000000000000, i.e. if convertRate is 10**12 the last 12 digits are cleaned.
- Following the rest of the execution in the `_withdraw` function, the total balance of TOFT would be `1999999000000000000` while the `safeTransfer` function at the end expects to transfer the input amount, i.e. `2e18`.

```
1          // send `contractAddress`
2          IERC20(contractAddress).safeTransfer(to, amount);
3          emit AmountWithdrawn(to, amount);
```

- This would cause the withdraw to un-expectedly revert and this attack can be performed repeatedly disabling withdrawals.

Recommendation

If `convertRate` for the StargatePool is different than 1 you should check the `toWithdrawFromPool` value. If it contains any dust you should withdraw slightly more so `toWithdrawFromPool` is a multiple of `convertRate`. Make sure to handle the edge case when the user is the last one withdrawing, as `toWithdrawFromPool` can have a maximum value equal to the balance of LPTokens in the farm.

Resolution

No resolution yet.

M-1 Setting new `StargatePool` address makes strategy unusable

Context

- StargateV2Strategy.sol#L206-L210

Description

During `StargateV2Strategy` contract deployment `pool` and corresponding `lpToken` variables get initialized. Each `StargatePool` is deployed with a corresponding `LPToken` that can't change.

`StargateV2Strategy.setPool()` allows to change the address of the StargatePool but it doesn't change the underlying LP token. There is in fact no option to even change the LPToken address in the `StargateV2Strategy` contract.

If the owner sets a new pool the old LPToken would still remain making all the functions in the contract reverting.

Recommendation

There are a few recommendations here:

- As LPToken is an immutable variable inside the StargatePool there is no need to save it as a storage variable in the `StargateV2Strategy` contract. When needed it can be read from the pool contract directly, e.g. `pool.lpToken()`.
- Consider removing `setPool` function altogether. In its current state, the `StargateV2Strategy` is supposed to work exclusively with USDC pool on Artbitrum and it's not clear why would a change of pool be needed in the first place. If it does occur it would break other functionality. If you wish to keep the `setPool` function then all the LPTokens should be redeemed into the underlying tokens and all the rewards collected before the new pool gets set.
- Consider adding an admin-controlled function to withdraw any tokens that might be sitting in the `StargateV2Strategy` contract.

Resolution

Fixed by removing the possibility of setting a new pool in #78.

M-2 `pendingRewards` assumes that ARB/STG are always present as rewards

Context

- StargateV2Strategy.sol#L311-L315

Description

The farm contract used in the `StargateV2Strategy` contract can add and remove rewards. It's not guaranteed that ARB/STG will always be present as rewards and it's best to not rely on its existence.

If ARB or STG gets removed as a reward `_currentBalance` will permanently revert.

Recommendation

If STG/ARB are not present as a reward consider simply skipping them instead of reverting.

Resolution

Fixed by skipping the reward if it's not found in #81.

M-3 `ITapiocaOracle.peek` call should return manipulation resistant prices and check staleness

Context

- StargateV2Strategy.sol#L317-L318

Description

`ITapiocaOracle.peek` function call is used to get the latest prices for ARB and STG. Considering the `StargateV2Strategy` is intended to be used for USDC it is important that the prices returned are in ARB/USDC and STG/USDC. Respecting the difference in the decimal configuration of USDC, e.g. 6 decimals. Prices should also be manipulation resistant, e.g. you shouldn't rely on a spot price from a DEX.

By checking the interface description for the `ITapiocaOracle.peek` function return data, the first value is a bool. If no valid (recent) rate is available, it should return false otherwise true.

```
1     /// @notice Check the last exchange rate without any state changes.
2     /// @param data Usually abi encoded, implementation-specific data
          that contains information and arguments to & about the oracle.
3     /// For example:
4     /// (string memory collateralSymbol, string memory assetSymbol,
          uint256 division) = abi.decode(data, (string, string, uint256));
```

```
>>>     /// @return success if no valid (recent) rate is available,
    return false else true.
     /// @return rate The rate of the requested asset / pair / pool.
     function peek(bytes calldata data) external view returns (bool
        success, uint256 rate);
```

Since this bool value is not checked at all, the prices of ARB/STG can be stale and higher than the spot price leading to the perception of a much higher withdrawable balance.

Recommendation

There is no immediate security impact within the StargateV2Strategy contract, however other parts of the system relying on this value might be affected.

Resolution

Fixed in #82, if the price fetched from the oracle is stale the reward token is not accounted for. The price returned from the oracle is expected to respect the decimal difference between the reward tokens and input token, i.e. STG/USDC and ARB/USDC.

M-4 `_currentBalance` doesn't consider harvested rewards

Context

- StargateV2Strategy.sol#L339

Description

`_currentBalance` only considers pending rewards and not the rewards that were already claimed and sitting on the balance of the `StargateV2Strategy` contract. The rewards can be claimed by anyone by either calling the `claim` function or `StargateStaking.depositTo()` and specifying the `to == StargateV2Strategy`. This is because a deposit triggers the transfer of rewards.

```
## StargateMultiRewarder.sol

function onUpdate(
        IERC20 stakingToken,
        address user,
        uint256 oldStake,
        uint256 oldSupply,
```

```
8           uint256 /*newStake*/
9       ) external onlyStaking {
10          uint256[] memory ids = registry.byStake[stakingToken].values();
11          address[] memory tokens = new address[](ids.length);
12          uint256[] memory amounts = new uint256[](ids.length);
13
14          for (uint256 i = 0; i < ids.length; i++) {
15              RewardPool storage pool = registry.pools[ids[i]];
16              address rewardToken = pool.rewardToken;
17              tokens[i] = rewardToken;
18              amounts[i] = pool.indexAndUpdate(registry.rewardDetails[
                    rewardToken], user, oldStake, oldSupply);
19          }
20
21          emit RewardsClaimed(user, tokens, amounts);
22
23          for (uint256 i = 0; i < ids.length; i++) {
24              if (amounts[i] > 0) {
25                  _transferToken(user, tokens[i], amounts[i]);
26              }
27          }
28      }
```

Recommendation

Consider including harvested rewards into the `_currentBalance` calculation.

Resolution

Acknowledged.

L-1 `setFarm` doesn't collect the rewards and withdraw staked LPTokens

Context

- StargateV2Strategy.sol#L217-L221

Description

`StargateV2Strategy.setFarm()` function is used to set a new Staking contract where LPTokens can be deposited to collect rewards. The function doesn't check if there are any LPTokens staked in the old farm or if there are some pending rewards. If a change of farm does occur there is no way of retrieving

the staked amounts from the old farm without re-setting it again and calling `emergencyWithdraw`.

Recommendation

The `setFarm` function should withdraw all the LPTokens staked before setting up a new farm.

Resolution

Fixed in #80, `setFarm` now withdraws lpTokens from the old farm and deposits them in the new one.

L-2 reward tokens should be swapped into `inputToken` during invest

Context

- StargateV2Strategy.sol#L261-L282

Description

The `invest` function is used to swap STG and ARB into USDC and deposit/stake the output amount. There is no check that the `SZeroXSwapData.buyToken` is `inputToken`(USDC). The owner might accidentally swap tokens into something else than `inputToken` and there seems to be no way of retrieving tokens from the `StargateV2Strategy` contract.

Recommendation

Add a check that the ARB/STG are swapped into `inputToken` of the `StargateV2Strategy` contract.

Resolution

Fixed in #84.

L-3 `_currentBalance` function returns an approximate amount of token in the strategy

Context

- StargateV2Strategy.sol#L337

Description

For `StargatePools` that have the `convertRate != 1` it's not possible to redeem dust amounts of LPTokens. For example, if `convertRate == 1e12` an amount of LPTokens less than this value is not possible to convert to the underlying tokens.

These are dust amounts and someone can intentionally donate them to the `StargateV2Strategy` to just slightly inflate the value of `_currentBalance`. In other words, its value can be higher than what is actually withdrawable from the strategy.

Recommendation

Estimate the potential impact on other parts of the system that rely on this value.

Resolution

No resolution yet.

L-4 `clearAllowance` call is redundant

Context

- StargateV2Strategy.sol#L176, StargateV2Strategy.sol#L387

Description

`clearAllowance` gets called to wipe the allowance inside the Pearlmit after wrapping the `inputToken` into its corresponding TOFT.

By checking the `Pearlmit.clearAllowance()` function it seems that this call has no purpose:

```
## Pearlmit.sol

    /**
     * @notice Clear the allowance of an owner if it is called by the approved operator
     */
    function clearAllowance(address owner, uint256 tokenType, address token, uint256 id) external {
>>>         (uint256 allowedAmount,) = _allowance(_transferApprovals, owner, msg.sender, tokenType, token, id, ZERO_BYTES32);
        if (allowedAmount > 0) {
            _clearAllowance(owner, tokenType, token, msg.sender, id);
        }
    }
```

`owner = StargateV2Strategy` and `msg.sender = StargateV2Strategy` which will always have an allowance equal to 0.

Recommendation

If there is a need to clear the allowance it should be done inside the TOFT contract since during this flow TOFT is the operator that gets granted the allowance during the `pearlmit.approve()` call.

Resolution

`clearAllowance` has been removed in #83.