Gains Network Security Review

Pashov Audit Group

Conducted by: Said, ast3ros, btk

July 5th 2024 - July 7th 2024

Contents

1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

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

3. Introduction

A time-boxed security review of the **GainsNetwork-org/gTrade-contracts** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Gains Network

Gains Network is a liquidity-efficient decentralized leveraged trading platform. Trades are opened with DAI, USDC or WETH collateral, regardless of the trading pair. The leverage is synthetic and backed by the respective gToken vault, and the GNS token. Trader profit is taken from the vaults to pay the traders PnL (if positive), or receives trader losses from trades if their PnL was negative.

5. Risk Classification

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

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

review commit hashes:

- 9847881d77ebacdf0824b3440a65e8ed1ac52cbc
- 3586456d5b11230365823f954b46e2e8dc7e886d

fixes review commit hashes:

- e0f440b2cdddcdbe9022df6414717700906de45c
- a1477f67c6125576cd55670a76e9bfd9889d65f9

Scope

The following smart contracts were in scope of the audit:

For `Scope v9.1`:

- `GNSStaking`
- `GNSDiamondStorage`
- `GNSOtc`
- `OtcUtils`
- `StorageUtils`
- `TradingCommonUtils`

For `Scope v8.0.1`:

- `GNSPriceAggregator`
- `LiquidityPoolUtils`
- `PriceAggregatorUtils`

7. Executive Summary

Over the course of the security review, Said, ast3ros, btk engaged with Gains Network to review Gains Network. In this period of time a total of **2** issues were uncovered.

Protocol Summary

Protocol Name	Gains Network
Repository	https://github.com/GainsNetwork-org/gTrade-contracts
Date	July 5th 2024 - July 7th 2024
Protocol Type	Leveraged trading platform

Findings Count

Severity	**Amount**
High	1
Medium	1
Total Findings	**2**

Summary of Findings

ID	Title	Severity	Status
[H-01]	Storage slot collision	High	Resolved
[M-01]	MIN_GNS_WEI_IN is too small leading to zero rewards	Medium	Resolved

8. Findings

8.1. High Findings

[H-01] Storage slot collision

Severity

Impact: High

Likelihood: Medium

Description

There is a miscalculation in the storage layout allocation for the `PriceAggregatorStorage` struct, which leads to a potential storage slot collision with `OtcStorage`.

The key issue is with the bytes32[2] jobIds field. It's incorrectly calculated as taking up 64 bits, but in reality, it occupies 64 bytes (512 bits), spanning 2 full storage slots.

```
struct PriceAggregatorStorage {
        IChainlinkFeed linkUsdPriceFeed; // 160 bits
        uint24 twapInterval; // 24 bits
        uint8 minAnswers; // 8 bits
        bytes32[2] jobIds; // 64 bits @audit 64 bytes instead of 64 bits
        // ... other fields
        uint256[41] __gap;
    }
```

This miscalculation causes the `PriceAggregatorStorage` to actually occupy 52 slots instead of the intended 50 slots. The storage slots are allocated as follows:

```
uint256 internal constant GLOBAL_PRICE_AGGREGATOR_SLOT = 551;
    uint256 internal constant GLOBAL_OTC_SLOT = 601;
```

With PriceAggregatorStorage occupying 52 slots starting from slot 551, it will overlap with the first two slots of OtcStorage, which starts at slot 601. This

overlap will cause data corruption, as the first two slots of OtcStorage will be overwritten by the last two slots of PriceAggregatorStorage.

Recommendations

Reduce the __gap array size to 39 in `PriceAggregatorStorage` or change `GLOBAL_OTC_SLOT` to start from slot 603 to ensure no overlap.

8.2. Medium Findings

[M-01] `MIN_GNS_WEI_IN` is too small leading to zero rewards

Severity

Impact: Medium

Likelihood: Medium

Description

When `sellGnsForCollateral` is executed, the caller can trade their gns for the provided collateral, as long as there is enough `collateralBalances` and the calculated `gnsAmount` is greater than `MIN_GNS_WEI_IN` (100).

```
function sellGnsForCollateral
   (uint8 _collateralIndex, uint256 _collateralAmount) internal {
 IOtc.OtcStorage storage s = _getStorage();

 uint256 availableCollateral = s.collateralBalances[_collateralIndex];
 uint256 gnsPriceCollateral = getOtcRate(_collateralIndex);
 uint256 gnsAmount = _calculateGnsAmount
   (_collateralIndex, _collateralAmount, gnsPriceCollateral);

 // 1. Validation
 if (_collateralAmount == 0) revert IGeneralErrors.ZeroValue();
 if
   (_collateralAmount > availableCollateral) revert IGeneralErrors.InsufficientBalance
>>>     if (gnsAmount < MIN_GNS_WEI_IN) revert IGeneralErrors.BelowMin();

 // 2. Receive GNS from caller
 TradingCommonUtils.transferGnsFrom(msg.sender, gnsAmount);

 // 3. Reduce available OTC balance for collateral
 uint256 newBalanceCollateral = availableCollateral - _collateralAmount;
 s.collateralBalances[_collateralIndex] = newBalanceCollateral;
 emit IOtcUtils.OtcBalanceUpdated(_collateralIndex, newBalanceCollateral);

 // 4. Distribute GNS
 (
   uint256treasuryAmountGns,
   uint256stakingAmountGns,
   uint256burnAmountGns
 ) = _calculateGnsDistribution(
 gnsAmount
 );

 if (treasuryAmountGns > 0) _distributeTreasuryGns(treasuryAmountGns);
 if (stakingAmountGns > 0) _distributeStakingGns(stakingAmountGns);
 if (burnAmountGns > 0) _burnGns(burnAmountGns);

 // 5. Transfer collateral to caller
 TradingCommonUtils.transferCollateralTo
   (_collateralIndex, msg.sender, _collateralAmount);

 emit IOtcUtils.OtcExecuted(
 _collateralIndex,
 _collateralAmount,
 gnsPriceCollateral,
 treasuryAmountGns,
 stakingAmountGns,
 burnAmountGns
 );
 }
```

The received `gnsAmount` will then be distributed to the treasury, GNS staking, and burn, as calculated by `_calculateGnsDistribution` using the configured percentages. However, when distributing to staking Gns, it is possible that the provided amount results in 0 additional `accRewardPerGns` due to its small value.

```
function distributeReward(
   address_token,
   uint256_amountToken
 ) external override onlyRewardToken(_token
 require(gnsBalance > 0, "NO_GNS_STAKED");

 IERC20(_token).safeTransferFrom(msg.sender, address(this), _amountToken);

 RewardState storage rewardState = rewardTokenState[_token];
>>>     rewardState.accRewardPerGns += uint128(
  (_amountToken * rewardState.precisionDelta * 1e18) / gnsBalance);

 emit RewardDistributed(_token, _amountToken);
 }
```

This will cause the transferred GNS tokens to the GNS staking contract to not be recorded as rewards.

Recommendations

Increase the value of `MIN_GNS_WEI_IN` to a safer and more reasonable amount.