Competitive Security Assessment

Gambit-P3

Nov 3rd, 2023

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

Project Detail

Project Name	Gambit-P3
Platform & Language	Solidity
Codebase	• https://github.com/changerio/changer-futures-contracts-public • audit commit - d81292137d4e638b8bdec8ea2d547fe7a1a15d1d • final commit - 440ee7769f0acc21efc55be63108a63a3984d029
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Audit Scope

File	SHA256 Hash
contracts/v1/callback/GambitTradingCallbacksV1.sol	586948d511c0fd30e93da258a064189730d70aed398b651a67196ef7d99fd428
contracts/v1/trading-storage/GambitTradingStorageV1.sol	6be3851bd88948857f617f9d46be0fcb5fec4052719d3f801a705ce25e487ee7
contracts/v1/trading/GambitTradingV1.sol	c07f03f811c879678979e2e540f28d7ae434fe42684f24d729e544691ed7b720
contracts/v1/pair-infos/GambitPairInfosV1.sol	d6af72aa0c77ca87bb9b797fb51dc71e0ae1b8dc245b3adf5023c5f37ac225fe
contracts/v1/callback/GambitPriceAggregatorV1.sol	ef63626ef012306e05375a17db812b1842473a807ea75d5c6c7154dc7df55db8
contracts/v1/pair-storage/GambitPairsStorageV1.sol	7ad597266fc16b0d2432881f01307a125ec22d5616aa5c508649e87b8827b9a0
contracts/v1/trading-storage/interfaces/IGambitTradingStorageV1.sol	f17124905f23b498d7873517ed8147318bc7e3a26fdff73e06927a7a2f80baad
contracts/v1/pair-infos/interfaces/IGambitPairInfosV1.sol	27784127073685a9d52d013a625280b7444f440003f96cf62297c75781531a81
contracts/v1/pair-storage/interfaces/IGambitPairsStorageV1.sol	0a07ed406a0729a1da352c61db1a6ba1169c68956029ec805f2d7bb2655a052c
contracts/v1/callback/interfaces/IGambitPriceAggregatorV1.sol	01dcf4475426bb6a4f7ea8f7418f04dd556ff1544a9a60dc06f7f8e12d8af455
contracts/v1/callback/interfaces/IGambitTradingCallbacksV1.sol	b46ecaac0377e5fae4bbdaa0c2a91ae50e6065e172fb70fa27e9fe754ede6fc8

Code Assessment Findings

ID	Name	Category	Severity	Client Response	Contributor
GAM-1	Users can increase `leverage` to exceed the max leverage	Logical	Critical	Mitigated	danielt
GAM-2	Potential logical flaw in `isWithinExposureLimits()`	Logical	Critical	Fixed	Hacker007
GAM-3	Incorrect price impact calculation	Logical	Medium	Fixed	biakia
GAM-4	Unnecessary fee charging	Logical	Medium	Mitigated	danielt
GAM-5	Logical risk in `GambitPairInfosV1::diff()` function	Logical	Medium	Fixed	newway55
GAM-6	Limit orders are not executed on TP/SL value update.	Logical	Low	Mitigated	n16h7m4r3
GAM-7	`tx.origin` == `msg.sender` check that ensures call is from EOA might not hold true in the future	Language Specific	Low	Acknowledged	Hacker007
GAM-8	Redundant checks	Logical	Informational	Fixed	biakia, Hacker007, danielt
GAM-9	Code Style in `IGambitTradingStorageV1::Trade()` struct	Code Style	Informational	Acknowledged	newway55

GAM-1:Users can increase `leverage` to exceed the max leverage

Category	Severity	Client Response	Contributor
Logical	Critical	Mitigated	danielt

Code Reference

- code/contracts/v1/callback/GambitTradingCallbacksV1.sol#L776-L778

```
776:uint newPositionSizeUsdc = t.positionSizeUsdc - amount; // underflow checked in trading contract
777:             uint newLeverage = (t.positionSizeUsdc * t.leverage) /
778:                 newPositionSizeUsdc;
```

Description

danielt : In the `GambitTradingCallbacksV1.sol` contract, the `removeCollateralCallback` function can remove collateral for a trade without checking if the new leverage is valid.

```
    function removeCollateralCallback(
        AggregatorAnswer calldata a
    ) external onlyPriceAggregator notDone {
        ...
        if (t.leverage > 0) {
            ...
            uint newPositionSizeUsdc = t.positionSizeUsdc - amount; // underflow checked in trading
contract
            uint newLeverage = (t.positionSizeUsdc * t.leverage) /
                newPositionSizeUsdc;

            // Charge in USDC if collateral in storage
            v.reward1 = storageT.handleDevGovFees(
                o.pairIndex,
                (t.positionSizeUsdc * t.leverage) / 1e18,
                false
            );
            ...
```

As a result, users can increase the leverage of a trade to exceed the max leverage by removing the collateral with the `removeCollateralCallback` function, which may harm the protocol.

Recommendation

danielt : Recommend limiting the `newLeverage`, as what did in the `openTrade` function of the `GambitTradingV1.sol` contract:

```
    function openTrade(
        IGambitTradingStorageV1.Trade memory t,
        NftRewardsInterfaceV6.OpenLimitOrderType orderType, // LEGACY => market
        uint spreadReductionId,
        uint slippageP, // for market orders only
        address referrer
    ) external notContract notDone {
    ...
        require(
            t.leverage > 0 &&
                t.leverage >= pairsStored.pairMinLeverage(t.pairIndex) &&
                t.leverage <= pairsStored.pairMaxLeverage(t.pairIndex),
            "LEVERAGE_INCORRECT"
        );
    ...
```

Client Response

Mitigated,The validity of removeColalteralOrder (e.g., new leverage range) is checked from GambitTradingV1's removeCollateral function.

GAM-2:Potential logical flaw in `isWithinExposureLimits()`

Category	Severity	Client Response	Contributor
Logical	Critical	Fixed	Hacker007

Code Reference

- code/contracts/v1/pair-infos/GambitPairInfosV1.sol#L854-L866
- code/contracts/v1/pair-infos/GambitPairInfosV1.sol#L883-L891

```
854:int newExp = int(
855:             (params.positionSizeUsdc *
856:                 params.leverage *
857:                 (params.openPrice -
858:                     (
859:                         params.buy
860:                             ? (params.openPrice - params.currentPrice)
861:                             : (params.currentPrice - params.openPrice)
862:                     ))) /
863:                 params.openPrice /
864:                 PRECISION /
865:                 1e18
866:         );

883:int longExp = int(storageT.openInterestUsdc(params.pairIndex, 0)) +
884:                 pairPnlLong;
885:             int shortExp = int(storageT.openInterestUsdc(params.pairIndex, 1)) +
886:                 pairPnlShort;
887:             newExpDiff = diff(
888:                 params.buy ? longExp + newExp : longExp,
889:                 params.buy ? shortExp : shortExp + newExp
890:             );
891:             expDiff = diff(longExp, shortExp);
```

Description

Hacker007 : The function `isWithinExposureLimits()` uses the following formula to calculate the `newExp`.

```
        // exposure = col * lev + pnl
        // new exposure of the trade
        int newExp = int(
            (params.positionSizeUsdc *
                params.leverage *
                (params.openPrice -
                    (
                        params.buy
                            ? (params.openPrice - params.currentPrice)
                            : (params.currentPrice - params.openPrice)
                    )) /
                params.openPrice /
                PRECISION /
                1e18
        );
```

And call the function `openInterestUsdc()` to get the current long/short exp.

```
            int longExp = int(storageT.openInterestUsdc(params.pairIndex, 0)) +
                pairPnlLong;
            int shortExp = int(storageT.openInterestUsdc(params.pairIndex, 1)) +
                pairPnlShort;
            newExpDiff = diff(
                params.buy ? longExp + newExp : longExp,
                params.buy ? shortExp : shortExp + newExp
            );
```

The function `openInterestUsdc()` references the state variable.

```
mapping(uint => uint[3]) public openInterestUsdc; // 1e6 (USDC) or 1e18 (DAI) [long,short,max]
```

Per the comments, the decimals of all the related variables/functions is as below:

- params.positionSizeUsdc, 1e6 (USDC) or 1e18 (DAI)
- params.leverage, 1e18
- params.openPrice, 1e18
- PRECISION, 1e10
- storageT.openInterestUsdc(), 1e6 (USDC) or 1e18 (DAI)

If the decimals of `params.positionSizeUsdc` is 1e6(USDC), the newExp may round down to zero while the decimals of longExp/shortExp is 1e6. On the other side, if the decimals of `params.positionSizeUsdc` is 1e18(DAI), the newEXP decimal is 1e8 while the decimals of longExp/shortExp is 1e18.

The decimals of newExp is incompatible with longExp/shortExp in both cases, which may bring unexpected errors.

Recommendation

Hacker007 : Avoid divide newExp by PRECISION.

```
        int newExp = int(
            (params.positionSizeUsdc *
                params.leverage *
                (params.openPrice -
                    (
                        params.buy
                            ? (params.openPrice - params.currentPrice)
                            : (params.currentPrice - params.openPrice)
                    ))) /
                params.openPrice /
                1e18
        );
```

Client Response

Fixed

GAM-3:Incorrect price impact calculation

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	biakia

Code Reference

- code/contracts/v1/pair-infos/GambitPairInfosV1.sol#L478-L508

```
478:function getTradePriceImpactPure(
479:        uint openPrice, // PRECISION
480:        bool long,
481:        uint startOpenInterest, // 1e6 (USDC) or 1e18 (DAI)
482:        uint tradeOpenInterest, // 1e6 (USDC) or 1e18 (DAI)
483:        uint onePercentDepth // 1e6 (USDC) or 1e18 (DAI)
484:    )
485:        public
486:        view
487:        returns (
488:            uint priceImpactP, // PRECISION (%)
489:            uint priceAfterImpact // PRECISION
490:        )
491:    {
492:        if (onePercentDepth == 0) {
493:            return (0, openPrice);
494:        }
495:
496:        // USDC: 1e10 = 1e6  * 1e10 / 1e6  / "1e6"
497:        // DAI:  1e10 = 1e18 * 1e10 / 1e18 / "1e0"
498:        priceImpactP =
499:            ((startOpenInterest + tradeOpenInterest / 2) * PRECISION) /
500:            onePercentDepth /
501:            (usdcDecimals() == 6 ? 1e6 : 1e0);
502:
503:        uint priceImpact = (priceImpactP * openPrice) / PRECISION / 100;
504:
505:        priceAfterImpact = long
506:            ? openPrice + priceImpact
507:            : openPrice - priceImpact;
508:    }
```

Description

biakia : In contract `GambitPairInfosV1`, the function `getTradePriceImpactPure` is used to calculate the price impact:

```
function getTradePriceImpactPure(
        uint openPrice, // PRECISION
        bool long,
        uint startOpenInterest, // 1e6 (USDC) or 1e18 (DAI)
        uint tradeOpenInterest, // 1e6 (USDC) or 1e18 (DAI)
        uint onePercentDepth // 1e6 (USDC) or 1e18 (DAI)
    )
        public
        view
        returns (
            uint priceImpactP, // PRECISION (%)
            uint priceAfterImpact // PRECISION
        )
    {
        if (onePercentDepth == 0) {
            return (0, openPrice);
        }

        // USDC: 1e10 = 1e6  * 1e10 / 1e6  / "1e6"
        // DAI:  1e10 = 1e18 * 1e10 / 1e18 / "1e0"
        priceImpactP =
            ((startOpenInterest + tradeOpenInterest / 2) * PRECISION) /
            onePercentDepth /
            (usdcDecimals() == 6 ? 1e6 : 1e0);

        uint priceImpact = (priceImpactP * openPrice) / PRECISION / 100;

        priceAfterImpact = long
            ? openPrice + priceImpact
            : openPrice - priceImpact;
    }
```

If the token is `USDC`, the decimal of `startOpenInterest` and `tradeOpenInterest` is 6, the decimal of `PRECISIO N` is 10. The decimal of `onePercentDepth` is 6 as per the following code:

```
struct PairParams {
        uint onePercentDepthAbove; // 1e6 (USDC) or 1e18 (DAI)
        uint onePercentDepthBelow; // 1e6 (USDC) or 1e18 (DAI)
        uint rolloverFeePerBlockP; // PRECISION (%)
        uint fundingFeePerBlockP; // PRECISION (%)
    }
```

So the decimal of the `priceImpactP` will be 4(1e4 = 1e6 * 1e10 / 1e6 / 1e6) instead of 10, which means the price impact will be much smaller than it should be.

Recommendation

biakia : Consider following fix:

```
priceImpactP =
            ((startOpenInterest + tradeOpenInterest / 2) * PRECISION) /
            onePercentDepth;
```

Client Response

Fixed

GAM-4:Unnecessary fee charging

Category	Severity	Client Response	Contributor
Logical	Medium	Mitigated	danielt

Code Reference

- code/contracts/v1/callback/GambitTradingCallbacksV1.sol#L757

```
757:function removeCollateralCallback(
```

Description

danielt : In the `GambitTradingCallbacksV1.sol`, the `removeCollateralCallback` function comes to two results, collateral successfully removed or collateral removal is canceled.

```
    function removeCollateralCallback(
        AggregatorAnswer calldata a
    ) external onlyPriceAggregator notDone {
        IGambitTradingStorageV1.PendingRemoveCollateralOrder memory o = storageT
            .reqID_pendingRemoveCollateralOrder(a.orderId);

        IGambitTradingStorageV1.Trade memory t = storageT.openTrades(
            o.trader,
            o.pairIndex,
            o.index
        );

        if (t.leverage > 0) {
            IGambitTradingStorageV1.TradeInfo memory i = storageT
                .openTradesInfo(o.trader, o.pairIndex, o.index);

            Values memory v;
            uint amount = o.amount;

            uint newPositionSizeUsdc = t.positionSizeUsdc - amount; // underflow checked in trading
contract
            uint newLeverage = (t.positionSizeUsdc * t.leverage) /
                newPositionSizeUsdc;

            // Charge in USDC if collateral in storage
            v.reward1 = storageT.handleDevGovFees(
                o.pairIndex,
                (t.positionSizeUsdc * t.leverage) / 1e18,
                false
            );

            t.initialPosToken -=
                (v.reward1 * (10 ** (18 - usdcDecimals())) * PRECISION) /
                i.tokenPriceUsdc;
            t.positionSizeUsdc -= v.reward1;
            storageT.updateTrade(t);

            emit DevGovFeeCharged(t.trader, v.reward1);
            ...
            if (
                a.price > 0 &&
                t.buy == o.buy &&
                t.openPrice == o.openPrice &&
```

```
  (t.buy ? v.liqPrice <= a.price : v.liqPrice >= a.price)
            ) {
                ...
                emit CollateralRemoved(
                    a.orderId,
                    o.trader,
                    o.pairIndex,
                    o.index,
                    amount,
                    newLeverage
                );
            } else {
                emit CollateralRemoveCanceled(  //@audit 揖让是cancel ,  为什么要在L791 updateTrade?
                    a.orderId,
                    o.trader,
                    o.pairIndex,
                    o.index
                );
            }
        }

        storageT.unregisterPendingRemoveCollateralOrder(a.orderId);
    }
```

In the case of the collateral removal being canceled, the fee is still charged by invoking the `handleDevGovFees` function and the `updateTrade` function, which looks unnecessary, especially for the high-leverage trade.

Recommendation

danielt : Recommend checking if the implementation meets the design and consider charging a trade fee only for the case of collateral successfully removed.

Client Response

Mitigated,The remove collateral order charges fee because bot is required to send transaction to update on-demand price. The fee charging policy is same as updateSlCallback function: 1) if trade that order is pointing is same, charge fee and take the order. 2) if trade that order is pointing has been closed , cannot charge fee and cannot take order. 3) if trade that order is pointing has been closed and another trade is opened, charge fee (from new open trade) and cancel the order. We will add a function to cancel open order (remove collateral order and update sl order) in the future.

GAM-5:Logical risk in `GambitPairInfosV1::diff()` function

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	newway55

Code Reference

- code/contracts/v1/pair-infos/GambitPairInfosV1.sol#L926-L931

```
926:function diff(int a, int b) internal pure returns (int) {
927:        int a_ = abs(a);
928:        int b_ = abs(b);
929:        return a_ > b_ ? a_ - b_ : b_ - a_;
930:    }
931:}
```

Description

newway55 : The diff function provided aims to return the absolute difference between two integers. This function will not always return the absolute difference because it disregards the original signs of a and b when determining which is larger after applying the abs function.

The mistake in the original logic arises from not taking the signs of a and b into account after calculating the absolute values.

POC :

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

import "forge-std/Test.sol";
import "../GambitPairInfosV1.sol";t

contract GambitPairInfosV1Test is Test {
    GambitPairInfosV1 internal gambitPairInfos;

    function setUp() public {
        gambitPairInfos = new GambitPairInfosV1();

    }

    function testDiffFunction() public {
        // Test with both positive numbers
        assertEq(gambitPairInfos.diff(10, 5), 5, "Diff should be 5");

        // Test with a positive and a negative number
        assertEq(gambitPairInfos.diff(-10, 5), 15, "Diff should be 15");

        // Test with both negative numbers
        assertEq(gambitPairInfos.diff(-10, -5), 5, "Diff should be 5");

        // Test with a negative and a positive number
        assertEq(gambitPairInfos.diff(10, -5), 15, "Diff should be 15");

        // Test with zeros
        assertEq(gambitPairInfos.diff(0, -5), 5, "Diff should be 5");
        assertEq(gambitPairInfos.diff(5, 0), 5, "Diff should be 5");
        assertEq(gambitPairInfos.diff(0, 0), 0, "Diff should be 0");

        // Test where the first number is smaller than the second
        assertEq(gambitPairInfos.diff(5, 10), 5, "Diff should be 5");

        // Test where the absolute values are the same but signs differ
        assertEq(gambitPairInfos.diff(-5, 5), 10, "Diff should be 10");
    }
}
```

Recommendation

newway55 : The comparison a_ > b_ evaluate to true (since 5 > 3), and the function would return a_ - b_, which is 5 - 3, resulting in 2.

But, this is not the correct absolute difference between -5 and 3. The absolute difference should be |-5 - 3| = | -8 | = 8

```
function diff(int a, int b) internal pure returns (int) {
    return abs(a - b);
}
```

Since diff is used in `isWithinExposureLimits` function it is completely giving wrong results and thus this is a critical issue.

Client Response

Fixed

GAM-6:Limit orders are not executed on TP/SL value update.

Category	Severity	Client Response	Contributor
Logical	Low	Mitigated	n16h7m4r3

Code Reference

- code/contracts/v1/trading/GambitTradingV1.sol#L499
- code/contracts/v1/trading/GambitTradingV1.sol#L529
- code/contracts/v1/trading-storage/GambitTradingStorageV1.sol#L764
- code/contracts/v1/trading-storage/GambitTradingStorageV1.sol#L779

```
499:function updateTp(

529:function updateSl(

764:function updateSl(

779:function updateTp(
```

Description

n16h7m4r3 : Functions `updateTp()` and `updateSl()` allows traders to update the `t.tp` (take profit) or `t.sl` (stop loss) target for the limit order's. The functions `updateTp()` and `updateSl()` do not check and execute limit order trade immediately if the current position of the asset has reached the target value (TP/SL).

In EVM compatible chains transactions wait in the mempool to get validated and finalized in the upcoming block. Also the trading pair could be highly volatile depending on market sentiment. For the trader's this could result in position not getting squared off in case if the TP target has already been reached or incur additional loss in an event when the SL is triggered below the target SL during the update of the respective values.

Recommendation

n16h7m4r3 : Consider executing limit order if the target TP or SL value is reached during update. Also consider adding a function to allow trader to update TP/SL simultaneously.

Client Response

Mitigated,bot close the trade with NFT order when the price is reached to trade’s TP or SL value. So, if trader can successfully execute updateTp or updateSl, this means that bot considered the trade as not a target trade to be closed.

We choose this design as updating pyth network's price requires fee and we want to restrict the permission to update price is restricted.

GAM-7: `tx.origin` == `msg.sender` check that ensures call is from EOA might not hold true in the future

Category	Severity	Client Response	Contributor
Language Specific	Low	Acknowledged	Hacker007

Code Reference

- code/contracts/v1/trading/GambitTradingV1.sol#L149-L152

```
149:modifier notContract() {
150:        require(tx.origin == msg.sender);
151:        _;
152:    }
```

Description

Hacker007 : The modifier `notContract` to check if msg.sender is a contract account.

```
    modifier notContract() {
        require(tx.origin == msg.sender);
        _;
    }
```

`tx.origin == msg.sender` check ensures calls are only made from EOA. However, EIP-3074 suggests that this condition to ensure calls are only from EOA might not hold true. According to EIP 3074, this EIP introduces two EVM instructions AUTH and AUTHCALL. The first sets a context variable authorized based on an ECDSA signature. The second sends a call as the authorized account. This essentially delegates control of the EOA to a smart contract. Therefore, using `tx.origin` to ensure `msg.sender` is an EOA will not hold true in the event EIP 3074 goes through. Moreover, Do NOT assume that `tx.origin` will continue to be usable or meaningful according to the answer of `Vitalik Buterin`.

Recommendation

Hacker007 : Recommend using the OpenZeppelin library `Address`. The function `isContract()` checks the code size of any address.

Client Response

Acknowledged,We acknowledge this issue but don’t change the codebase because we are plan to migrate this issue in the future with ERC-4337 migation.

GAM-8:Redundant checks

Category	Severity	Client Response	Contributor
Logical	Informational	Fixed	biakia, Hacker007, danielt

Code Reference

- code/contracts/v1/trading-storage/GambitTradingStorageV1.sol#L366-L372
- code/contracts/v1/pair-infos/GambitPairInfosV1.sol#L380-L408
- code/contracts/v1/trading/GambitTradingV1.sol#L641-L646
- code/contracts/v1/trading/GambitTradingV1.sol#L694-L699

```
366:function updateTimelockOwner(
367:        address _timelockOwner
368:    ) external nonZeroAddress(_timelockOwner) onlyTimelockOwner {
369:        require(_timelockOwner != address(0));
370:        timelockOwner = _timelockOwner;
371:        emit AddressUpdated("timelockOwner", _timelockOwner);
372:    }

380:uint openInterestToken = storageT.openInterestToken(pairIndex, index); // 1e15 (USDC or DAI)
381:        uint openInterestUsdc = storageT.openInterestUsdc(pairIndex, index); // 1e6 (USDC) or 1e
18 (DAI)
382:
383:        if (openInterestToken == 0) return 0;
384:
385:        // 1e19 (USDC) or 1e7 (DAI)
386:        uint d = 10 ** (25 - usdcDecimals());
387:
388:        // USDC: 1e10 = 1e6  * "1e19" / 1e15
389:        // DAI:  1e10 = 1e18 * "1e7" / 1e15
390:        uint avgOpenPrice = openInterestToken == 0
391:            ? 0
392:            : (openInterestUsdc * d) / openInterestToken;
393:
394:        // 1e10
395:        int priceDiff = avgOpenPrice == 0
396:            ? int(0)
397:            : (
398:                avgOpenPrice > currentPrice
399:                    ? -int(avgOpenPrice - currentPrice) // price drop => loss (long)
400:                    : int(currentPrice - avgOpenPrice) // price rise => profit (long)
401:            );
402:
403:        // reverse direction if short
404:        if (!buy) priceDiff *= -1;
405:
406:        // USDC: 1e6  = 1e10 * 1e15 / "1e19"
407:        // DAI:  1e18 = 1e10 * 1e15 / "1e7"
408:        pnl = (priceDiff * int(openInterestToken)) / int(d);

641:require(
642:            newLeverage > 0 &&
```

```
643:                newLeverage >= pairsStored.pairMinLeverage(t.pairIndex) &&
644:                newLeverage <= pairsStored.pairMaxLeverage(t.pairIndex),
645:            "LEVERAGE_INCORRECT"
646:        );

694:require(
695:            newLeverage > 0 &&
696:                newLeverage >= pairsStored.pairMinLeverage(t.pairIndex) &&
697:                newLeverage <= pairsStored.pairMaxLeverage(t.pairIndex),
698:            "LEVERAGE_INCORRECT"
699:        );
```

Description

biakia : In the function `updateTimelockOwner`, the modifier `nonZeroAddress` and the `require` statement do the same thing, which will check that `_timelockOwner` is not `address(0)`:

```
require(_timelockOwner != address(0));

 modifier nonZeroAddress(address a) {
        require(a != address(0), "ZERO_ADDRESS");
        _;
    }
```

These checks are redundant.

Hacker007 : The function `getOpenPnLSide()` calls `openInterestUsdc(pairIndex, index)` and assigns the return value to `openInterestUsdc`. Per the function logic of `getOpenPnLSide()`, the target pnl will be zero if `openInterestUsdc` is zero. Moreover, the if statement in L383 ensures that openInterestToken is greater than 0, the ternary operator in L390-L392 is redundant.

danielt : In the `GambitTradingV1.sol` contract, the `addCollateral()` function adds collaterals to an existing trade, which results in its leverage lower. Thus, there is no need to check if its `newLeverage` is less than or equal to `pairsStored.pairMaxLeverage(t.pairIndex)`:

```
    function addCollateral(
        uint pairIndex,
        uint index,
        uint amount
    ) external notContract notDone {
        ...
        uint newPositionSizeUsdc = t.positionSizeUsdc + amount;
        uint newLeverage = (t.positionSizeUsdc * t.leverage) /
            newPositionSizeUsdc;
        ...
        require(
            newLeverage > 0 &&
                newLeverage >= pairsStored.pairMinLeverage(t.pairIndex) &&
                newLeverage <= pairsStored.pairMaxLeverage(t.pairIndex),
            "LEVERAGE_INCORRECT"
        );
```

A similar scenario happens to the `removeCollateral()` function, there is no need to check if a trade's `newLeverage` is greater than or equal to `pairsStored.pairMinLeverage(t.pairIndex)` due to the removal of collateral will increase its `newLeverage`.

Recommendation

biakia : Consider removing the `require` statement:

```
  function updateTimelockOwner(
        address _timelockOwner
    ) external nonZeroAddress(_timelockOwner) onlyTimelockOwner {
        timelockOwner = _timelockOwner;
        emit AddressUpdated("timelockOwner", _timelockOwner);
    }
```

Hacker007 : Check if `openInterestUsdc` is zero and return directly.

```
        if (openInterestToken == 0 || openInterestUsdc == 0) return 0;
```

Remove the redundant check.

```
        uint avgOpenPrice = (openInterestUsdc * d) / openInterestToken;
```

danielt : Recommend removing the redundant checks in the above functions.

Client Response

Fixed

GAM-9:Code Style in `IGambitTradingStorageV1::Trade()` struct

Category	Severity	Client Response	Contributor
Code Style	Informational	Acknowledged	newway55

Code Reference

- code/contracts/v1/trading-storage/interfaces/IGambitTradingStorageV1.sol#L24-L35

```
24:struct Trade {
25:         address trader;
26:         uint pairIndex;
27:         uint index;
28:         uint initialPosToken; // 1e18
29:         uint positionSizeUsdc; // 1e6 (USDC) or 1e18 (DAI)
30:         uint openPrice; // PRECISION
31:         bool buy;
32:         uint leverage; // 1e18
33:         uint tp; // PRECISION
34:         uint sl; // PRECISION
35:     }
```

Description

newway55 : In the `Trade` struct there is a `positionSizeUsdc` that can be either USDC (1e6) or DAI (1e18). Mixing these 2 different denominations can be confusing and there is risk of errors in calculations if not handled the right way.

Recommendation

newway55 : Add a type Stable coin which is an enum to make difference between the 2 tokens.

`enum Stablecoin { USDC, DAI }`

```
struct Trade {
    // ... other fields
    Stablecoin stablecoin; // To indicate which stablecoin is being used
}
```

Client Response

Acknowledged,We acknowledge this issue but don’t change the codebase because contract takes only one of DAI or USDC, not both. Instead of using enum, renaming it to positionSizeStablecoin or positionSizeUSD seems more

appropriate. We will change the variable name in the future.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3's prior written consent in each instance.

This report is not an "endorsement" or "disapproval" of any particular project or team. This report is not an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3's position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.