Competitive Security Assessment

Gambit-Wemix

Sep 25th, 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-Wemix
Platform & Language	Solidity
Codebase	• https://github.com/changerio/changer-futures-contracts-public/tree/dev/0.1.0 • audit commit - c18a975fe77652c8dbdd1241f9d7984867f403de • final commit - 850d1a8e2c1a8aed8b3a3a0952d320df47e95ab8
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Code Vulnerability Review Summary

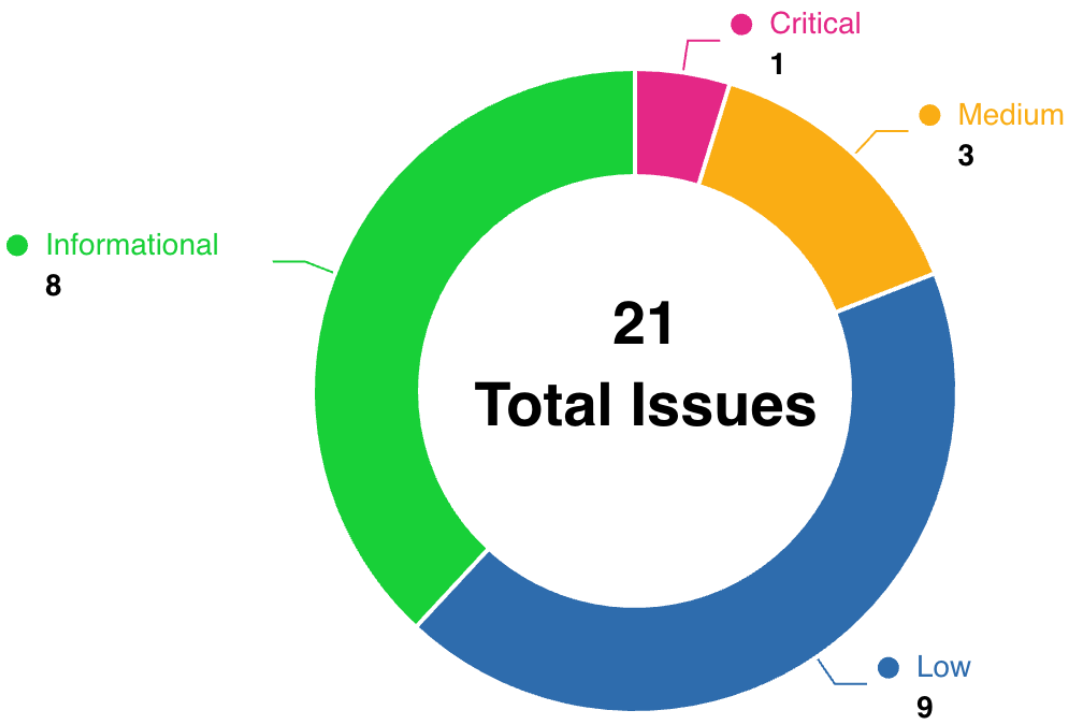
Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	1	0	0	1	0	0
Medium	3	0	1	2	0	0
Low	9	0	4	4	1	0
Informational	8	0	3	5	0	0

Audit Scope

File	SHA256 Hash
contracts/v1/callback/GambitTradingCallbacksV1.sol	4e61bfa15a348f709e2a119e218dd74e212e43de535345dc4e544347c27a23bd
contracts/v1/trading-storage/GambitTradingStorageV1.sol	d5800c21a7841dc8adc523c4c4f3a90684272972f78350d245929a71ed9398de
contracts/v1/trading/GambitTradingV1.sol	2c66843cb253a8854bef1f03a07cd36f6c4168d4f8aa6901ece12a5c26b66816
contracts/v1/pair-infos/GambitPairInfosV1.sol	c36dfdca81419c03165189c6abe379a81af0f6c4c428e091bebf303e130612d4
contracts/v1/vault/SimpleGToken.sol	a07da77fd865ae82263069f82c4896b20a6cdbe92f383e0ef6d99ede2269a1bc
contracts/v1/trading/GambitReferralsV1.sol	fa87210d0b5da0cc168493e62b2c8020ae54e56c632dad687a97aa9b2a40c7bb
contracts/v1/vault/GTokenOpenPnlFeed.sol	47b8dbfe8d1271847d789680605d25cf2b3e8ca534c27e2e1303a194532cb247
contracts/v1/callback/GambitPriceAggregatorV1.sol	04f634d91494e467aa29fdf481798d6dbfdc2927b9043f205473eebf4cc7901a
contracts/misc/GnosisMultiSigWallet.sol	0c3c61ccdbd2d4a90f5daa577a623ff7ba861d04c8f38ef859ff262cb7d25d69
contracts/v1/pair-storage/GambitPairsStorageV1.sol	a18bdc8f3a12ac0c51109969369a1dd6e0dbc17a3789f1d2bab6a4a8ac90c170
contracts/v1/pair-storage/GambitNftRewardsV1.sol	82ca9348743de6c6db001e6db21061990b9b29769aa706f731ead54142688838
contracts/v1/trading-storage/interfaces/IGambitTradingStorageV1.sol	3d10b5b8d2d277fbdd27480ae6c8f2266be2004e889238cfa1ba67d4815b5238
contracts/v1/staking/GambitStakingV1.sol	a8457823df39aefc26793b39478cc24c822b43cdb69c77061563966184da43c3
contracts/v1/callback/TWAPPriceGetter.sol	2f7c000701502f80ec1b88a099c63780458d4158d52860de1f8d0f17880ee074
contracts/v1/treasury/Treasury.sol	fbf84b813732af0c3402046fe89c505ddf166b9afc19a5c764ab913dfadb36cf

contracts/multicall/Multicall.sol	20d0a1d7f65fc4ccbb11330d1ee32038fcddf7a6f58423d61ae8f0821aef37bd
contracts/misc/TokenDistributor.sol	26021a4a8b23e1f0b197d8827395ffc5ea8706e9bb76c9690e9fcc458e7763ff
contracts/v1/staking/interfaces/IGambitStakingV1.sol	89b8eb38de6b122722ea56b3e62062c95d79ca2e173b4bdad4faf060e7fa0678
contracts/misc/GambitTimelockOwner.sol	0301c828923d55c645cf4b5d2f4cfbfbcdcf7105fcd701f97906591f1120ae9b
contracts/multicall/IMulticall.sol	411a9c103d942a469370add02a3b749e2c26934022dd729731f8f371a47521c5
contracts/_import_sol_0_8_17.sol	de7db6ad0f2c5be7d3c3c6b6398bd657ffe2e2621480501094ce5fcd41f74b25
contracts/v1/common/IStableCoinDecimals.sol	3ba41304c80580f78baf503b162a31867c84a9bc3e0cffeafa6308c2ab0672eb
contracts/v1/trading-storage/interfaces/IGov.sol	a53c6f993098f85ec1769f4d39390d425b355285b30f552a8a5aae512f1d789c
contracts/v1/trading-storage/interfaces/IStableCoinDecimals.sol	8c0159dff04f274fa5665925117ad171327d46c12d513bd708076b552785afc0

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
GBW-1	Potential rounding down issue	Logical	Critical	Fixed	biakia
GBW-2	Use safeTransfer instead of transfer in TokenDistributor contract distribute function	Logical	Medium	Fixed	ginlee, biakia
GBW-3	Rewards will be locked in the contract when there are no stakers	Logical	Medium	Fixed	biakia, BradMoonUESTC

GBW-4	`latestRoundData` returns has been ignored in `GambitPriceAggregatorV1.sol`	Oracle Manipulation	Medium	Acknowledged	xfu
GBW-5	In “GambitPriceAggregatorV1” contract updatePrice function, price oracle will use the wrong price if the Chainlink returns price outside min/max range	Oracle Manipulation	Low	Acknowledged	ginlee
GBW-6	should follow CEI pattern in "GambitTradingStorageV1" contract "handleTokens" function	Reentrancy	Low	Fixed	ginlee, Atlas
GBW-7	Lack of address check	Logical	Low	Fixed	biakia
GBW-8	Use `SafeERC20` to approve tokens	Logical	Low	Fixed	biakia
GBW-9	Use `getPrice` instead of `getPriceUnsafe`	Oracle Manipulation	Low	Acknowledged	biakia
GBW-10	Using vulnerable dependency of OpenZeppelin	Language Specific	Low	Fixed	xfu
GBW-11	Lack of a double-step `transferOwnership()` pattern	Logical	Low	Mitigated	xfu
GBW-12	Wrong gap layout in `GambitNftRewardsV1.sol` contract	Logical	Low	Acknowledged	Atlas
GBW-13	Access Control: `TokenDistributor.distribute()` can be called by anyone	Privilege Related	Low	Acknowledged	0xgm
GBW-14	Missing error message in require statement	Language Specific	Informational	Fixed	biakia
GBW-15	Redundant Code	Gas Optimization	Informational	Fixed	biakia
GBW-16	Unlocked Pragma Version	Language Specific	Informational	Acknowledged	biakia
GBW-17	Erroneous comments with `zkSync`	Code Style	Informational	Acknowledged	xfu
GBW-18	Use `calldata` instead of `memory` for function parameters	Gas Optimization	Informational	Fixed	xfu

GBW-19	Use indexed events for value types as they are less costly compared to non-indexed ones	Gas Optimization	Informational	Acknowledged	xfu
GBW-20	Cache the `<array>.length` for the loop condition	Gas Optimization	Informational	Fixed	xfu
GBW-21	Variable can be declared as memory in `GambitStakingV1::pendingRewardUsdc()` function	Gas Optimization	Informational	Fixed	Atlas

GBW-1:Potential rounding down issue

Category	Severity	Client Response	Contributor
Logical	Critical	Fixed	biakia

Code Reference

- code/contracts/v1/staking/GambitStakingV1.sol#L77-L88
- code/contracts/v1/vault/SimpleGToken.sol#L343-L354

```
77:function distributeRewardUsdc(
78:        uint amount // 1e6 (USDC) or 1e18 (DAI)
79:    ) external {
80:        usdc.safeTransferFrom(msg.sender, address(this), amount);
81:
82:        if (tokenBalance > 0) {
83:            accUsdcPerToken += (amount * 1e18) / tokenBalance;
84:            totalRewardsDistributedUsdc += amount;
85:        }
86:
87:        emit UsdcDistributed(amount);
88:    }

343:function updateShareToAssetsPrice(int assets, uint supply) private {
344:        uint priceDeltaAbs = uint(10 ** usdcDecimals()).mulDiv(
345:            assets > 0 ? uint(assets) : uint(-assets),
346:            supply
347:        );
348:
349:        // 1e6 (USDC) or 1e18 (DAI)
350:        if (assets > 0) shareToAssetsPrice += priceDeltaAbs;
351:        else shareToAssetsPrice -= priceDeltaAbs;
352:
353:        emit ShareToAssetsPriceUpdated(shareToAssetsPrice);
354:    }
```

Description

biakia : The contract `GambitStakingV1` supports two types of rewards, USDC and DAI. When distributing `USDC` rewards, it is possible to encounter a rounding-down issue when calculating the `accUsdcPerToken`:

```
if (tokenBalance > 0) {
            accUsdcPerToken += (amount * 1e18) / tokenBalance;
            totalRewardsDistributedUsdc += amount;
}
```

We know that the `tokenBalance` is the amount of `CNG` which decimal is 18, and the `amount` is the amount of `USDC` which decimal is 6. Consider the current `tokenBalance` is 1000000 * 1e18 and the `amount` is 1e5(means 0.1 USDC rewards). The formula `(amount * 1e18) / tokenBalance` will be `1e5*1e18/(1000000*1e18) = 0` due to the rounding down issue. Ultimately, the 0.1 USDC reward will not be distributed to the user, but be permanently locked in the contract.

In contract `SimpleGToken`, the function `distributeReward` will distribute a reward evenly to all stakers of the vault. It will call the function `updateShareToAssetsPrice`:

```
function updateShareToAssetsPrice(int assets, uint supply) private {
        uint priceDeltaAbs = uint(10 ** usdcDecimals()).mulDiv(
            assets > 0 ? uint(assets) : uint(-assets),
            supply
        );

        // 1e6 (USDC) or 1e18 (DAI)
        if (assets > 0) shareToAssetsPrice += priceDeltaAbs;
        else shareToAssetsPrice -= priceDeltaAbs;

        emit ShareToAssetsPriceUpdated(shareToAssetsPrice);
    }
```

The same rounding down issue exists in this function when the asset is `USDC`. Consider the `assets` is 1e6(means 1USDC) and the supply is 1e18(the ERC4626 has a default 18 decimal), the `priceDeltaAbs` will be `10**6 * 1e6 / 1e18 = 0`.

Recommendation

biakia : In contract `GambitStakingV1`, consider using a larger amplification factor:

```
accUsdcPerToken += (amount * 1e30) / tokenBalance;
```

In contract `SimpleGToken`, consider redesigning the logic of the calculation of `shareToAssetsPrice`.

Client Response

Fixed. For GambitStakingV1 contract, we increased the precision of accUsdcPerToken as follow:

USDC: 1e6 → 1e24

DAI: 1e18 → 1e36

For SimpleGToken contract, the precision of supply is equal to asset's one as folllow:

USDC: asset: 1e6, supply (Vault's precision): 1e6

DAI: asset: 1e18, supply (Vault's precision): 1e18 So, rounding error happens in updateShareToAssetsPrice function when asset = 0.1 USDC and supply = 1,000,000 USDC (not 1 USDC in the report). To fix this, we increased the precision of shareToAssetsPrice as follow:

USDC: 1e6 → 1e24

DAI: 1e18 → 1e36

GBW-2:Use safeTransfer instead of transfer in TokenDistributor contract distribute function

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	ginlee, biakia

Code Reference

- code/contracts/misc/TokenDistributor.sol#L7-L18
- code/contracts/misc/TokenDistributor.sol#L15

```
7:function distribute(
8:        IERC20 token,
9:        address[] calldata accounts,
10:        uint[] calldata amounts
11:    ) external {
12:        require(accounts.length == amounts.length, "L");
13:
14:        for (uint i = 0; i < accounts.length; i++) {
15:            token.transferFrom(msg.sender, accounts[i], amounts[i]);
16:        }
17:    }
18:}

15:token.transferFrom(msg.sender, accounts[i], amounts[i]);
```

Description

ginlee :

```
token.transferFrom(msg.sender, accounts[i], amounts[i])
```

The ERC20.transfer() functions return a boolean value indicating success. This parameter needs to be checked for success Some tokens do not return a bool (e.g. USDT, BNB, OMG) on ERC20 methods. Some tokens (e.g. BNB) may return a bool for some methods, but fail to do so for others. Some particularly pathological tokens (e.g. Tether Gold) declare a bool return, but then return false even when the transfer was successful

biakia : The return value of the `transferFrom()` call is not checked.

Recommendation

ginlee : Recommend using OpenZeppelin's SafeERC20 versions with the safeTransfer functions that handle the return value check as well as non-standard-compliant tokens.

biakia : Since some ERC-20 tokens return no values and others return a `bool` value, they should be handled with care. We advise using the OpenZeppelin's `SafeERC20.sol` implementation to interact with the `transfer()` and `transferFrom()` functions of external ERC-20 tokens. The OpenZeppelin implementation checks for the existence of a return value and reverts if `false` is returned, making it compatible with all ERC-20 token implementations.

Client Response

Fixed

GBW-3:Rewards will be locked in the contract when there are no stakers

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	biakia, BradMoonUESTC

Code Reference

- code/contracts/v1/staking/GambitStakingV1.sol#L77-L88
- code/contracts/v1/staking/GambitStakingV1.sol#L77

```
77:function distributeRewardUsdc(
78:        uint amount // 1e6 (USDC) or 1e18 (DAI)
79:    ) external {
80:        usdc.safeTransferFrom(msg.sender, address(this), amount);
81:
82:        if (tokenBalance > 0) {
83:            accUsdcPerToken += (amount * 1e18) / tokenBalance;
84:            totalRewardsDistributedUsdc += amount;
85:        }
86:
87:        emit UsdcDistributed(amount);
88:    }

77:function distributeRewardUsdc(
```

Description

BradMoonUESTC : The smart contract contains a logical vulnerability within the distributeRewardUsdc function. Specifically, even when there's no token balance within the contract (tokenBalance is 0), the function still permits senders to distribute USDC rewards to the contract. However, as there are no tokens staked in the pool, these USDC rewards are not allocated to any users and remain stranded within the contract. Future users staking tokens won't benefit from these rewards as the accUsdcPerToken only updates based on new rewards being distributed and not based on the stranded USDC already in the contract.

Recommendation

biakia : We recommend distributing these rewards to the `treasury` when there are no stakers in the contract.
BradMoonUESTC : To mitigate this vulnerability, a condition should be added at the beginning of the distributeRewardUsdc function to ensure that the tokenBalance is greater than 0 before proceeding. If tokenBalance is 0,

the function should revert the transaction with an appropriate error message. This ensures that rewards are only distributed when there are staked tokens in the contract, preventing stranded USDC rewards.

Client Response

Fixed

GBW-4: `latestRoundData` returns has been ignored in `GambitPriceAggregatorV1.sol`

Category	Severity	Client Response	Contributor
Oracle Manipulation	Medium	Acknowledged	xfu

Code Reference

- code/contracts/v1/callback/GambitPriceAggregatorV1.sol#L347-L348
- code/contracts/v1/callback/GambitPriceAggregatorV1.sol#L364-L365

```
347:(, int feedPrice1, , , ) = ChainlinkFeedInterfaceV5(f.feed1)
348:                .latestRoundData();

364:(, int feedPrice2, , , ) = ChainlinkFeedInterfaceV5(f.feed2)
365:                    .latestRoundData();
```

Description

xfu : The `latestRoundData` function in the contract `GambitPriceAggregatorV1.sol` fetches the asset price from a Chainlink aggregator using the latestRoundData function. However, the returns is ignored.

If there is a problem with chainlink starting a new round and finding consensus on the new value for the oracle (e.g. chainlink nodes abandon the oracle, chain congestion, vulnerability/attacks on the chainlink system) consumers of this contract may continue using outdated stale data (if oracles are unable to submit no new round is started)

There are `2` instances of this issue:

- (None,feedPrice1,None,None,None) = ChainlinkFeedInterfaceV5(f.feed1).latestRoundData() returns has been ignored.
- (None,feedPrice2,None,None,None) = ChainlinkFeedInterfaceV5(f.feed2).latestRoundData() returns has been ignored.

Recommendation

xfu : Consider checking the all oracle responses value after calling out to `chainlinkOracle.latestRoundData()` verifying that the result is within an allowed margin.

For example:

```
    (
        uint80 roundId,
        int256 price,
        uint256 startedAt,
        uint256 updatedAt,
        uint80 answeredInRound
    ) = aggregator.latestRoundData();

    if (answeredInRound < roundId){
        revert("answer is being carried over");
    }
    if (startedAt == 0) {
        revert("Round not complete");
    }
    if (price == 0) {
        revert("answer reporting 0");
    }
    if (updatedAt < block.timestamp - maxDelayTime) {
        revert("time err");
    }
```

Client Response

Acknowledged. We acknowledge this issue but don't change the codebase because we use 2 sources of a price, chainlink to get valid price range and pyth network to get actual price. Even though chainlink's price is stale and incorrect, we can use fresh and accurate price from pyth network unless pyth network's price is in range of [chainlink's price * 0.99, chainlink's price * 1.01].

GBW-5:In “GambitPriceAggregatorV1” contract updatePrice function, price oracle will use the wrong price if the Chainlink returns price outside min/max range

Category	Severity	Client Response	Contributor
Oracle Manipulation	Low	Acknowledged	ginlee

Code Reference

- code/contracts/v1/callback/GambitPriceAggregatorV1.sol#L347-L350

```
347:(, int feedPrice1, , , ) = ChainlinkFeedInterfaceV5(f.feed1)
348:                .latestRoundData();
349:            require(feedPrice1 > 0, "INVALID_PRICE");
350:            feedPrice = uint(feedPrice1);
```

Description

ginlee :

```
(, int feedPrice1, , , ) = ChainlinkFeedInterfaceV5(f.feed1)
.latestRoundData();
require(feedPrice1 > 0, "INVALID_PRICE");
feedPrice = uint(feedPrice1);
```

Chainlink aggregators have a built in circuit breaker if the price of an asset goes outside of a predetermined price band. The result is that if an asset experiences a huge drop in value (i.e. LUNA crash) the price of the oracle will continue to return the minPrice instead of the actual price of the asset. Note there is only a check for price to be non-negative, and not within an acceptable range.

Recommendation

ginlee : Implement the proper check for each asset. It must revert in the case of bad price.

```
require(feedPrice1 >= minPrice && feedPrice1 <= maxPrice, "invalid price");
```

Also in this function, updatedAt params should be added to check price feed staleness

```
require (updatedAt >= block.timestamp - 3600, "stale price")
```

Client Response

Acknowledged. We acknowledge this issue but don’t change the codebase because we use 2 sources of a price, chainlink to get valid price range and pyth network to get actual price. Even though chainlink’s price is stale and incorrect,

we can use fresh and accurate price from pyth network unless pyth network’s price is in range of [chainlink’s price * 0.99, chainlink’s price * 1.01].

GBW-6:should follow CEI pattern in "GambitTradingStorageV1" contract "handleTokens" function

Category	Severity	Client Response	Contributor
Reentrancy	Low	Fixed	ginlee, Atlas

Code Reference

- code/contracts/v1/staking/GambitStakingV1.sol#L118-L133
- code/contracts/v1/trading-storage/GambitTradingStorageV1.sol#L842-L848
- code/contracts/v1/staking/GambitStakingV1.sol#L119-L133

```
118:// Stake tokens
119:    function stakeTokens(
120:        uint amount // 1e18
121:    ) external {
122:        User storage u = users[msg.sender];
123:
124:        token.safeTransferFrom(msg.sender, address(this), amount);
125:
126:        harvest();
127:
128:        u.stakedTokens += amount;
129:        u.debtUsdc = (u.stakedTokens * accUsdcPerToken) / 1e18;
130:        tokenBalance += amount;
131:
132:        emit TokensStaked(msg.sender, amount);
133:    }
```

```
119:function stakeTokens(
120:        uint amount // 1e18
121:    ) external {
122:        User storage u = users[msg.sender];
123:
124:        token.safeTransferFrom(msg.sender, address(this), amount);
125:
126:        harvest();
127:
128:        u.stakedTokens += amount;
129:        u.debtUsdc = (u.stakedTokens * accUsdcPerToken) / 1e18;
130:        tokenBalance += amount;
131:
132:        emit TokensStaked(msg.sender, amount);
133:    }
```

```
842:if (_mint) {
843:            token.mint(_a, _amount);
844:            tokensMinted += _amount;
845:        } else {
846:            token.burn(_a, _amount);
847:            tokensBurned += _amount;
848:        }
```

Description

ginlee :

```
    if (_mint) {
            token.mint(_a, _amount);
            tokensMinted += _amount;
        } else {
            token.burn(_a, _amount);
            tokensBurned += _amount;
        }
```

In this case, state variables like tokensMinted and tokensBurned are updated after the external calls. Ensure that the order of state changes is safe

```
     function stakeTokens(
         uint amount // 1e18
     ) external {
         User storage u = users[msg.sender];
         token.safeTransferFrom(msg.sender, address(this), amount);
         harvest();
         u.stakedTokens += amount;
         u.debtUsdc = (u.stakedTokens * accUsdcPerToken) / 1e18;
         tokenBalance += amount;
         emit TokensStaked(msg.sender, amount);
     }
```

State changes after token transfer which is an obvious reentrancy exploit design pattern

Atlas : In the contracts GambitStakingV1.sol, the Checks-Effects-Interactions pattern is not being followed in `stakeTokens()` functions. Although the safeTransfer method is calling a expected known CNG token, but the other two functions, `harvest()` and `unstakeTokens()`, follow this pattern. It's still recommended for `stakeTokens()` to follow this pattern also.

Recommendation

ginlee : Use the Checks-Effects-Interactions and make all state changes before calling external contracts. Consider using function modifiers such as nonReentrant from Openzeppelin ReentrancyGuard library to prevent re-entrancy.

Atlas : Use the Checks-Effects-Interactions best practice and make all state changes before calling external contracts. Consider below fix:

```
function stakeTokens(
    uint amount // 1e18
) external {
    User storage u = users[msg.sender];

    harvest();

    u.stakedTokens += amount;
    u.debtUsdc = (u.stakedTokens * accUsdcPerToken) / 1e18;
    tokenBalance += amount;

    token.safeTransferFrom(msg.sender, address(this), amount);

    emit TokensStaked(msg.sender, amount);
}
```

Client Response

Fixed

GBW-7:Lack of address check

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	biakia

Code Reference

- code/contracts/misc/GnosisMultiSigWallet.sol#L151-L164

```
151:function replaceOwner(
152:        address owner,
153:        address newOwner
154:    ) public onlyWallet ownerExists(owner) ownerDoesNotExist(newOwner) {
155:        for (uint i = 0; i < owners.length; i++)
156:            if (owners[i] == owner) {
157:                owners[i] = newOwner;
158:                break;
159:            }
160:        isOwner[owner] = false;
161:        isOwner[newOwner] = true;
162:        emit OwnerRemoval(owner);
163:        emit OwnerAddition(newOwner);
164:    }
```

Description

biakia : In contract `GnosisMultiSigWallet`, the function `addOwner` will check whether the input param `owner` is address(0) by the modifier `notNull`:

```
function addOwner(
        address owner
    )
        public
        onlyWallet
        ownerDoesNotExist(owner)
        notNull(owner)
        validRequirement(owners.length + 1, required)
    {
...
```

However, the function `replaceOwner` doesn't check the `newOwner`:

```
function replaceOwner(
        address owner,
        address newOwner
    ) public onlyWallet ownerExists(owner) ownerDoesNotExist(newOwner) {
        for (uint i = 0; i < owners.length; i++)
            if (owners[i] == owner) {
                owners[i] = newOwner;
                break;
            }
        isOwner[owner] = false;
        isOwner[newOwner] = true;
        emit OwnerRemoval(owner);
        emit OwnerAddition(newOwner);
    }
```

As a result, the owner can be set as `address(0)` by the function `replaceOwner`.

Recommendation

biakia : Consider adding the modifier `notNull` in function `replaceOwner`:

```
function replaceOwner(
        address owner,
        address newOwner
    ) public onlyWallet ownerExists(owner) ownerDoesNotExist(newOwner) notNull(newOwner){
```

Client Response

Fixed

GBW-8:Use `SafeERC20` to approve tokens

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	biakia

Code Reference

- code/contracts/v1/callback/GambitTradingCallbacksV1.sol#L170

```
170:storageT.usdc().approve(address(staking), type(uint256).max);
```

Description

biakia : In contract `GambitTradingCallbacksV1`, the function `approve` will be called to approve tokens to the `staking` contract:

```
storageT.usdc().approve(address(staking), type(uint256).max);
```

However, the return value of approve function is not checked.

Recommendation

biakia : Consider using `SafeERC20` to approve tokens.

Client Response

Fixed

GBW-9:Use `getPrice` instead of `getPriceUnsafe`

Category	Severity	Client Response	Contributor
Oracle Manipulation	Low	Acknowledged	biakia

Code Reference

- code/contracts/v1/vault/GTokenOpenPnlFeed.sol#L299-L309

```
299:PythStructs.Price memory pythPrice1 = pyth.getPriceUnsafe(
300:                f.priceId1
301:            );
302:            require(pythPrice1.price > 0, "INVALID_PRICE");
303:            require(pythPrice1.expo <= 0, "INVALID_CONF");
304:
305:            uint price = (uint(uint64(pythPrice1.price)) * PRECISION) /
306:                (10 ** uint(uint32(-pythPrice1.expo)));
307:
308:            int pnl = pairInfos.getOpenPnL(pairIndex, price);
309:            value += pnl;
```

Description

biakia : In contract `GTokenOpenPnlFeed`, the function `fulfill` will call `getPriceUnsafe` to get price from the pyth network:

```
// Price from Pyth network
            PythStructs.Price memory pythPrice1 = pyth.getPriceUnsafe(
                f.priceId1
          );
```

The document(https://docs.pyth.network/evm/get-price-unsafe) of this function shows that `This function may return a price from arbitrarily far in the past. It is the caller's responsibility to check the returned publishTime to ensure that the update is recent enough for their use case.`. However, the function `fulfill` did not check the returned `publishTime`. As a result, the `pnl` maybe calculated based on a stale price.

Recommendation

biakia : Consider using the function `getPrice` instead of `getPriceUnsafe`.

Client Response

Acknowledged. We acknowledge this issue but we don’t change the codebase 1) because getPrice() requires updating the price data, and it can be done by anyone if it is needed and 2) because the updated price can be stale if the market is closed.

GBW-10:Using vulnerable dependency of OpenZeppelin

Category	Severity	Client Response	Contributor
Language Specific	Low	Fixed	xfu

Code Reference

- code/contracts/v1/callback/GambitPriceAggregatorV1.sol#L4
- code/contracts/v1/callback/GambitTradingCallbacksV1.sol#L4-L6
- code/contracts/v1/pair-infos/GambitPairInfosV1.sol#L3-L4
- code/contracts/v1/pair-storage/GambitNftRewardsV1.sol#L4
- code/contracts/v1/staking/GambitStakingV1.sol#L4-L5
- code/contracts/v1/trading-storage/GambitTradingStorageV1.sol#L4-L6
- code/contracts/v1/trading/GambitReferralsV1.sol#L4-L6
- code/contracts/v1/trading/GambitTradingV1.sol#L4-L5
- code/contracts/misc/TokenDistributor.sol#L2

```
2:import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

3:import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
4:import "@openzeppelin/contracts-upgradeable/token/ERC20/extensions/IERC20MetadataUpgradeable.sol";

4:import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";

4:import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
5:import "@openzeppelin/contracts-upgradeable/token/ERC20/utils/SafeERC20Upgradeable.sol";
6:import "@openzeppelin/contracts-upgradeable/token/ERC20/extensions/IERC20MetadataUpgradeable.sol";

4:import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";

4:import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
5:import "@openzeppelin/contracts-upgradeable/token/ERC20/utils/SafeERC20Upgradeable.sol";

4:import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
5:import "@openzeppelin/contracts-upgradeable/token/ERC20/utils/SafeERC20Upgradeable.sol";
6:import "@openzeppelin/contracts-upgradeable/token/ERC20/extensions/IERC20MetadataUpgradeable.sol";

4:import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
5:import "@openzeppelin/contracts-upgradeable/token/ERC20/utils/SafeERC20Upgradeable.sol";
6:import "@openzeppelin/contracts-upgradeable/token/ERC20/extensions/IERC20MetadataUpgradeable.sol";

4:import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
5:import "@openzeppelin/contracts-upgradeable/token/ERC20/extensions/IERC20MetadataUpgradeable.sol";
```

Description

Recommendation

xfu : - Update `@openzeppelin/contracts` to the latest version.

- Update `@openzeppelin/contracts-upgradeable` to the latest version.

Client Response

Fixed

GBW-11:Lack of a double-step `transferOwnership()` pattern

Category	Severity	Client Response	Contributor
Logical	Low	Mitigated	xfu

Code Reference

- code/contracts/v1/vault/SimpleGToken.sol#L730-L734
- code/contracts/v1/vault/SimpleGToken.sol#L739-L743

```
730:contract SimpleGToken____6 is SimpleGToken {
731:    function usdcDecimals() public pure override returns (uint8) {
732:        return 6;
733:    }
734:}

739:contract SimpleGToken____18 is SimpleGToken {
740:    function usdcDecimals() public pure override returns (uint8) {
741:        return 18;
742:    }
743:}
```

Description

xfu : The current ownership transfer process for all the contracts inheriting from `OwnableUpgradeable` involves the current owner calling the transferOwnership() function:

```
    function transferOwnership(address newOwner) public virtual onlyOwner {
        if (newOwner == address(0)) {
            revert OwnableInvalidOwner(address(0));
        }
        _transferOwnership(newOwner);
    }
```

If the nominated EOA account is not a valid account, it is entirely possible that the owner may accidentally transfer ownership to an uncontrolled account, losing the access to all functions with the `onlyOwner` modifier.

There are `2` instances of this issue:

- SimpleGToken____6 does not implement a `2-Step-Process` for transferring ownership.
- SimpleGToken____18 does not implement a `2-Step-Process` for transferring ownership.

Recommendation

xfu : It is recommended to implement a two-step process where the owner nominates an account and the nominated account needs to call an `acceptOwnership()` function for the transfer of the ownership to fully succeed. This ensures the nominated EOA account is a valid and active account. This can be easily achieved by using OpenZeppelin's Ownable2StepUpgradeable contract instead of `OwnableUpgradeable`.

Client Response

Mitigated. We migrate this issue by using TimelockController owned by multisig as the owner of SimpleGToken contract.

GBW-12:Wrong gap layout in `GambitNftRewardsV1.sol` contract

Category	Severity	Client Response	Contributor
Logical	Low	Acknowledged	Atlas

Code Reference

- code/contracts/v1/pair-storage/GambitNftRewardsV1.sol#L12-L14

```
12:IGambitTradingStorageV1 public storageT;
13:
14:    bytes32[62] private _gap1; // storage slot gap (1 slot for above variable)
```

Description

Atlas : The storage layout of this contract is spaced by 64 slots. Before the `_gap1` variable, only one slot was occupied by `storageT`, so `gap1` should be 63. The comment is right, but the code is wrong.

Recommendation

Atlas : Correct code to `bytes32[63] private _gap1;`

Client Response

Acknowledged

GBW-13:Access Control: `TokenDistributor.distribute()` can be called by anyone

Category	Severity	Client Response	Contributor
Privilege Related	Low	Acknowledged	0xgm

Code Reference

- code/contracts/misc/TokenDistributor.sol#L11

```
11:) external {
```

Description

0xgm : The `TokenDistributor` contract contains a single function that transfers ERC20 token amounts to a provided set of accounts. This allows for easily transferring multiple amounts of say USDC to multiple addresses. This could potentially become problematic or entirely not important to the protocol since it is not actually being called from other contracts.

Recommendation

0xgm : Consider removing the contract altogether since it is not actually being used by another contract, i.e. `GambitStakingV1.distributeRewardUsdc()`.
If the contract is intended to be used within the protocol, consider adding access control to it using `onlyOwner` or a custom modifier of `onlyStakingContract`, or whichever contract is meant to interact with it.

```
contract TokenDistributor {
    address private _gambitStaking;

    error TokenDistributor__OnlyStakingContractCanDistribute();

    constructor(address _gambitStakingContract) {
        _gambitStaking = _gambitStakingContract;
    }

    modifier onlyStaking() {
        if (msg.sender != _gambitStaking) {
            revert TokenDistributor__OnlyStakingContractCanDistribute();
        }
        _;
    }

    function distribute(IERC20 token, address[] calldata accounts, uint256[] calldata amounts) exter
nal onlyStaking {
        require(accounts.length == amounts.length, "L");

        for (uint256 i = 0; i < accounts.length; i++) {
            token.transferFrom(msg.sender, accounts[i], amounts[i]);
        }
    }

    // Add onlyOwner from OZ
    function updateStakingContract(address _stakingContract) external onlyOnwer {
        _gambitStaking = _stakingContract;
    }
}
```

This may be over-engineered, so consider removing the contract if it is not intended to be used or integral to the protocol.

Client Response

Acknowledged. TokenDistributor contract is used to execute multiple token transfers(e.g., airdrop), not in Gambit protocol itself. We use TokenDistributor.distribute() with approving limited amount of tokens (meaning the total amount of tokens). And as the function use msg.sender's token and msg.sender is always EOA, there is no attack surface that attacker exploits this contract. Also, Not using onlyOwner is intended to allow any other EOAs to execute multiple token transfers.

GBW-14:Missing error message in require statement

Category	Severity	Client Response	Contributor
Language Specific	Informational	Fixed	biakia

Code Reference

- code/contracts/misc/GnosisMultiSigWallet.sol#L46
- code/contracts/misc/GnosisMultiSigWallet.sol#L51
- code/contracts/misc/GnosisMultiSigWallet.sol#L56
- code/contracts/misc/GnosisMultiSigWallet.sol#L61
- code/contracts/misc/GnosisMultiSigWallet.sol#L66
- code/contracts/misc/GnosisMultiSigWallet.sol#L71
- code/contracts/misc/GnosisMultiSigWallet.sol#L76
- code/contracts/misc/GnosisMultiSigWallet.sol#L81
- code/contracts/misc/GnosisMultiSigWallet.sol#L86-L91

```
46:require(msg.sender == address(this));

51:require(!isOwner[owner]);

56:require(isOwner[owner]);

61:require(transactions[transactionId].destination != address(0));

66:require(confirmations[transactionId][owner]);

71:require(!confirmations[transactionId][owner]);

76:require(!transactions[transactionId].executed);

81:require(_address != address(0));

86:require(
87:            ownerCount <= MAX_OWNER_COUNT &&
88:                _required <= ownerCount &&
89:                _required != 0 &&
90:                ownerCount != 0
91:        );
```

Description

biakia : An error message in require statement both helps user and dev to to understand why the execution has failed.

Recommendation

biakia : Consider adding error messages in require statement.

Client Response

Fixed

GBW-15:Redundant Code

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	biakia

Code Reference

- code/contracts/v1/callback/GambitPriceAggregatorV1.sol#L127-L130
- code/contracts/v1/trading-storage/GambitTradingStorageV1.sol#L37

```
37:address public tokenDaiRouter;

127:modifier onlyCallbacks() {
128:        require(msg.sender == storageT.callbacks(), "CALLBACKS_ONLY");
129:        _;
130:    }
```

Description

biakia : In the contract `GambitPriceAggregatorV1`, the modifier `onlyCallbacks` is never used. In the contract `GambitTradingStorageV1`, the variable `tokenDaiRouter` is never used. In the contract `GambitTradingStorageV1`, the event `TradingContractAdded` and `TradingContractRemoved` are never used.

Recommendation

biakia : If these variables are not intended to be used, it is recommended to remove them to save gas.

Client Response

Fixed

GBW-16:Unlocked Pragma Version

Category	Severity	Client Response	Contributor
Language Specific	Informational	Acknowledged	biakia

Code Reference

- code/contracts/v1/staking/interfaces/IGambitStakingV1.sol#L3
- code/contracts/v1/trading-storage/interfaces/IGambitTradingStorageV1.sol#L2
- code/contracts/v1/trading-storage/interfaces/IGov.sol#L2

```
2:pragma solidity ^0.8.0;

2:pragma solidity ^0.8.0;

3:pragma solidity ^0.8.0;
```

Description

biakia : Solidity files in packages have a pragma version `^0.8.0`. The caret `(^)` points to unlocked pragma, meaning the compiler will use the specified version or above.

Recommendation

biakia : It's good practice to use specific solidity versions to know compiler bug fixes and optimisations were enabled at the time of compiling the contracts.

Client Response

Acknowledged. We acknowledge this issue but we don’t change the codebase because refered files are all interfaces, not contract.

GBW-17:Erroneous comments with `zkSync`

Category	Severity	Client Response	Contributor
Code Style	Informational	Acknowledged	xfu

Code Reference

- code/contracts/v1/trading/GambitTradingV1.sol#L38
- code/contracts/v1/trading/GambitTradingV1.sol#L39
- code/contracts/v1/callback/GambitPriceAggregatorV1.sol#L115
- code/contracts/v1/treasury/Treasury.sol#L16

```
16:bytes32 public constant ADMIN_ROLE = keccak256("ADMIN_ROLE"); // note that zksync's hash function is different from ethereum's one.

38:uint public limitOrdersTimelock; // batch (zkSync) or block (other) (eg. 30)

39:uint public marketOrdersTimeout; // batch (zkSync) or block (other) (eg. 30)

115:PYTH_PRICE_AGE = 3 minutes; // TODO: reduce to 1 min when zksync supports L2 timestamp
```

Description

xfu : The following comments are not accurate:

```
// file: contracts/v1/trading/GambitTradingV1.sol
38:    uint public limitOrdersTimelock; // batch (zkSync) or block (other) (eg. 30)
39:    uint public marketOrdersTimeout; // batch (zkSync) or block (other) (eg. 30)

// file: contracts/v1/callback/GambitPriceAggregatorV1.sol
115:        PYTH_PRICE_AGE = 3 minutes; // TODO: reduce to 1 min when zksync supports L2 timestamp

// file: contracts/v1/treasury/Treasury.sol
16:    bytes32 public constant ADMIN_ROLE = keccak256("ADMIN_ROLE"); // note that zksync's hash function is different from ethereum's one.
```

These comments could cause mistakes for developers relying on them instead of the implementation. And `zkSync` has many differences from EVM, while `Wemix` is 100% compatible.

Consider updating the misleading comments.

Recommendation

xfu : It is recommended to rename the target chain `Wemix` instead of `zkSync`

Client Response

Acknowledged. We acknowledge this issue but do not change the codebase because we deploy same contracts to both zkSync and Wemix.

GBW-18:Use `calldata` instead of `memory` for function parameters

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	xfu

Code Reference

- code/contracts/misc/GnosisMultiSigWallet.sol#L180-L187
- code/contracts/v1/vault/SimpleGToken.sol#L271-L277

```
180:function submitTransaction(
181:        address destination,
182:        uint value,
183:        bytes memory data
184:    ) public returns (uint transactionId) {
185:        transactionId = addTransaction(destination, value, data);
186:        confirmTransaction(transactionId);
187:    }

271:function updateWithdrawLockThresholdsP(
272:        uint[2] memory newValue
273:    ) external onlyOwner {
274:        require(newValue[1] > newValue[0], "WRONG_VALUES");
275:        withdrawLockThresholdsP = newValue;
276:        emit WithdrawLockThresholdsPUpdated(newValue);
277:    }
```

Description

xfu : On external functions, when using the `memory` keyword with a function argument, what's happening is a `memory` acts as an intermediate.

When the function gets called externally, the array values are kept in `calldata` and copied to memory during ABI decoding (using the opcode `calldataload` and `mstore`). And during the for loop, the values in the array are accessed in memory using a `mload`. That is inefficient. Reading directly from `calldata` using `calldataload` instead of going via `memory` saves the gas from the intermediate memory operations that carry the values.

More detail see this

There are `2` instances of this issue:

- GnosisMultiSigWallet.submitTransaction(address,uint256,bytes) read-only `memory` parameters below should be changed to `calldata` :

- GnosisMultiSigWallet.submitTransaction(address,uint256,bytes).data

- SimpleGToken.updateWithdrawLockThresholdsP(uint256[2])read-only `memory` parameters below should be changed to `calldata` :
  - SimpleGToken.updateWithdrawLockThresholdsP(uint256[2]).newValue

Recommendation

xfu : Use `calldata` instead of `memory` for external functions where the function argument is read-only.

Client Response

Fixed

GBW-19:Use indexed events for value types as they are less costly compared to non-indexed ones

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Acknowledged	xfu

Code Reference

- code/contracts/misc/GnosisMultiSigWallet.sol#L15
- code/contracts/misc/GnosisMultiSigWallet.sol#L18
- code/contracts/v1/staking/GambitStakingV1.sol#L50
- code/contracts/v1/staking/GambitStakingV1.sol#L52
- code/contracts/v1/trading/GambitTradingV1.sol#L53
- code/contracts/v1/vault/SimpleGToken.sol#L117
- code/contracts/v1/vault/SimpleGToken.sol#L119-L123
- code/contracts/v1/vault/SimpleGToken.sol#L124-L128
- code/contracts/v1/vault/SimpleGToken.sol#L130-L136

```
15:event Deposit(address indexed sender, uint value);

18:event RequirementChange(uint required);

50:event UsdcHarvested(address indexed user, uint amount);

52:event TokensStaked(address indexed user, uint amount);

53:event NumberUpdated(string name, uint value);

117:event RewardDistributed(address indexed sender, uint assets);

119:event AssetsSent(
120:        address indexed sender,
121:        address indexed receiver,
122:        uint assets
123:    );

124:event AssetsReceived(
125:        address indexed sender,
126:        address indexed user,
127:        uint assets
128:    );

130:event AccPnlPerTokenUsedUpdated(
131:        address indexed sender,
132:        uint indexed newEpoch,
133:        uint prevPositiveOpenPnl, // 1e6 (USDC) or 1e18 (DAI)
134:        uint newPositiveOpenPnl, // 1e6 (USDC) or 1e18 (DAI)
135:        uint newEpochPositiveOpenPnl // 1e6 (USDC) or 1e18 (DAI)
136:    );
```

Description

xfu : Using the `indexed` keyword for value types (`bool/int/address/string/bytes`) saves gas costs, as seen in this example.

However, this is only the case for value types, whereas indexing reference types (`array/struct`) are more expensive than their unindexed version.

Recommendation

xfu : Using the `indexed` keyword for values types `bool/int/address/string/bytes` in event

Client Response

Acknowledged. We acknowledge this issue but we don’t change the codebase because contracts are already deployed and updated contract will not compatible with external service (e.g., subgraph)

GBW-20:Cache the `<array>.length` for the loop condition

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	xfu

Code Reference

- code/contracts/multicall/Multicall.sol#L13
- code/contracts/misc/TokenDistributor.sol#L14
- code/contracts/misc/GnosisMultiSigWallet.sol#L110
- code/contracts/misc/GnosisMultiSigWallet.sol#L138
- code/contracts/misc/GnosisMultiSigWallet.sol#L155
- code/contracts/misc/GnosisMultiSigWallet.sol#L248
- code/contracts/misc/GnosisMultiSigWallet.sol#L287
- code/contracts/misc/GnosisMultiSigWallet.sol#L321

```
13:for (uint256 i = 0; i < data.length; i++) {

14:for (uint i = 0; i < accounts.length; i++) {

110:for (uint i = 0; i < _owners.length; i++) {

138:for (uint i = 0; i < owners.length - 1; i++)

155:for (uint i = 0; i < owners.length; i++)

248:for (uint i = 0; i < owners.length; i++) {

287:for (uint i = 0; i < owners.length; i++)

321:for (i = 0; i < owners.length; i++)
```

Description

xfu : The overheads outlined below are *PER LOOP*, excluding the first loop

- storage arrays incur a Gwarmaccess (**100 gas**)
- memory arrays use `MLOAD` (**3 gas**)
- calldata arrays use `CALLDATALOAD` (**3 gas**)

Caching the length changes each of these to a `DUP<N>` (**3 gas**), and gets rid of the extra `DUP<N>` needed to store the stack offset. More detail optimization see this

There are `8` instances of this issue:

- i < data.length `<array>.length` should be cached.
- i < accounts.length `<array>.length` should be cached.
- i < _owners.length `<array>.length` should be cached.
- i < owners.length - 1 `<array>.length` should be cached.
- i < owners.length `<array>.length` should be cached.
- i < owners.length `<array>.length` should be cached.
- i < owners.length `<array>.length` should be cached.
- i < owners.length `<array>.length` should be cached.

Recommendation

xfu : Caching the `<array>.length` for the loop condition, for example:

```
// gas save (-230)
function loopArray_cached(uint256[] calldata ns) public returns (uint256 sum) {
  uint256 length = ns.length;
  for (uint256 i = 0; i < length; ) {
    sum += ns[i];
    unchecked {
      i++;
    }
  }
}
```

Client Response

Fixed

GBW-21:Variable can be declared as memory in `GambitStakingV1::pendingRewardUsdc()` function

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	Atlas

Code Reference

- code/contracts/v1/staking/GambitStakingV1.sol#L90-L103

```
90:// Rewards to be harvested
91:    function pendingRewardUsdc(
92:        address sender
93:    )
94:        public
95:        view
96:        returns (
97:            uint // 1e6 (USDC) or 1e18 (DAI)
98:        )
99:    {
100:        User storage u = users[sender];
101:
102:        return (u.stakedTokens * accUsdcPerToken) / 1e18 - u.debtUsdc;
103:    }
```

Description

Atlas : In the view `GambitStakingV1::pendingRewardUsdc()` function, It's better to use memory type for the `u` variable rather than the storage type.

Recommendation

Atlas : Change to `User memory u = users[sender];`

Client Response

Fixed

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.