# Gambit-Wemix Audit Report

Version 1.0

*Gin*

September 13, 2023

# Gambit-Wemix Audit Report

Gin

September 13, 2023

Prepared by: Gin

## Table of Contents



## Protocol Summary

Decentralized Leveraged Trading on Wemix

## Disclaimer

The security team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

## Risk Classification

| | | Impact | | |
|---|---|---|---|---|
| | | High | Medium | Low |
| | High | H | H/M | M |
| Likelihood | Medium | H/M | M | M/L |
| | Low | M | M/L | L |

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

### Scope

All files in codebase

### Issues found

| Severity | Number of Issues Found |
|---|---|
| High | 0 |
| Medium | 1 |
| Low | 2 |
| Info | 0 |
| Total | 3 |

# Findings

## Medium

### [M-01] Use safeTransfer instead of transfer in TokenDistributor contract distribute function

**Description:**

```
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 call is not checked.

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

## Low

### [L-01] should follow CEI pattern in "GambitTradingStorageV1" contract "handleTokens" function

**Description:**

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


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);
```

```
19 }
```

In this case, state variables like tokensMinted and tokensBurned are updated after the external calls. Ensure that the order of state changes is safe.State changes after token transfer which is an obvious reentrancy exploit design pattern

**Recommendation:** 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.

### [L-02] In "GambitPriceAggregatorV1" contract updatePrice function, price oracle will use the wrong price if the Chainlink returns price outside min/max range

**Description:**

```
(, 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:** 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");
```