Float Capital contest
Findings & Analysis Report

2021-09-16

Table of contents

Overview

About C4

Code 432n4 (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 code 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 code contest outlined in this document, C4 conducted an analysis of Float Capital smart contract system written in Solidity. The code contest took place between August 4—August 11.

Wardens

16 Wardens contributed reports to the Float Capital code contest:

1. gpersoon
2. shw
3. hickuphh3
4. cmichel
5. jonah1005
6. 0xRajeev
7. pauliax
8. tensors
9. hack3r-0m
10. Jmukesh
11. 0xImpostor
12. evertkors
13. loop
14. hrkrshnn
15. patitonar

16. bw

This contest was judged by Oxean (judge).

Final report assembled by moneylegobatman and ninek.

Summary

The C4 analysis yielded an aggregated total of 24 unique vulnerabilities. All of the issues presented here are linked back to their original finding

Of these vulnerabilities, 3 received a risk rating in the category of HIGH severity, 6 received a risk rating in the category of MEDIUM severity, and 15 received a risk rating in the category of LOW severity.

C4 analysis also identified 25 non-critical recommendations and 21 gas optimizations.

Scope

The code under review can be found within the C4 Float Capital code contest repository is comprised of 43 smart contracts written in the Solidity programming language and includes 4,215 lines of Solidity, 7,512 lines of ReasonML, and 8,062 lines of Rescript code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into three primary risk categories: high, medium, and low.

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

- Malicious Input Handling
- Escalation of privileges
- Arithmetic

- Gas use

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on [the C4 website](#).

High Risk Findings

[H-01] copy paste error in `_batchConfirmOutstandingPendingActions`

Submitted by gpersoon, also found by cmichel and shw

The function `_batchConfirmOutstandingPendingActions` of `LongShort.sol` processes the variable `batched_amountSyntheticToken_toShiftAwayFrom_marketSide`, and sets it to 0 after processing. However, probably due to a copy/paste error, in the second instance, where `batched_amountSyntheticToken_toShiftAwayFrom_marketSide[marketIndex][false]` is processed, the wrong version is set to 0: `batched_amountSyntheticToken_toShiftAwayFrom_marketSide[marketIndex][true]` = 0

This means the next time the `batched_amountSyntheticToken_toShiftAwayFrom_marketSide[marketIndex][false]` is processed again. As it is never reset, it keeps increasing. The result is that the internal administration will be off and far too many tokens will be shifted tokens from SHORT to LONG.

`LongShort.sol` L1126

```
    function _batchConfirmOutstandingPendingActions(
    ..
        amountForCurrentAction_workingVariable = batched_amountSynth
        batched_amountSyntheticToken_toShiftAwayFrom_marketSide[mark
    ...
        amountForCurrentAction_workingVariable = batched_amountSynth
        batched_amountSyntheticToken_toShiftAwayFrom_marketSide[mark
```

```
    )
```

Recommend changing the second instance of the following (on line 1207) `batched_amountSyntheticToken_toShiftAwayFrom_marketSide[marketIndex][true] = 0` to `batched_amountSyntheticToken_toShiftAwayFrom_marketSide[marketIndex][false] = 0`

p.s. confirmed by Jason of Floatcapital: “Yes, that should definitely be false!”

JasoonS (Float) commented:

> Mitigation

```
- batched_amountSyntheticToken_toShiftAwayFrom_marketSide[market
+ batched_amountSyntheticToken_toShiftAwayFrom_marketSide[market
```

moose-code (Float) confirmed:

> Good attention to detail. Silly on our part.

DenhamPreen (Float) patched:

> https://github.com/Float-Capital/monorepo/pull/1087

> Resolved PR

> *note this repo is still private

[H-02] 2 variables not indexed by `marketIndex`

Submitted by gpersoon

In the token contract: `batched_stakerNextTokenShiftIndex` is indexed by `marketIndex`, so it can have separate (or the same) values for each different `marketIndex`.

`stakerTokenShiftIndex_to_longShortMarketPriceSnapshotIndex_mapping` and `stakerTokenShiftIndex_to_accumulativeFloatIssuanceSnapshotIndex_mapping` are not indexed by `marketIndex`. So the values of `stakerTokenShiftIndex_to_longShortMarketPriceSnapshotIndex_mapping` and `stakerTokenShiftIndex_to_accumulativeFloatIssuanceSnapshotIndex_mapping` can be overwritten by a different market, if `batched_stakerNextTokenShiftIndex[market1] == batched_stakerNextTokenShiftIndex[market2]`

This will lead to weird results in `_calculateAccumulatedFloat`, allocating too much or too little float.

`Staker.sol` **L622**

```
    function pushUpdatedMarketPricesToUpdateFloatIssuanceCalculatior
        ...
          stakerTokenShiftIndex_to_longShortMarketPriceSnapshotIndex
          stakerTokenShiftIndex_to_accumulativeFloatIssuanceSnapshot
          batched_stakerNextTokenShiftIndex[marketIndex] += 1;
    ...
    )
```

Recommend adding an index with `marketIndex` to the variables:

- `stakerTokenShiftIndex_to_longShortMarketPriceSnapshotIndex_mapping`
- `stakerTokenShiftIndex_to_accumulativeFloatIssuanceSnapshotIndex_mapping`

Also consider shortening the variable names, this way mistakes can be spotted easier.

Confirmed by Jason of Float Capital: Yes, you are totally right, it should use the `marketIndex` since they are specific per market!

JasoonS (Float) confirmed:

> 🙈 Embarrassed by this one!

> Thank you for the report.

> Fixed!!

[H-03] Users could shift tokens on `Staker` with more than he has staked

Submitted by shw

The `shiftTokens` function of `Staker` checks whether the user has staked at least the number of tokens he wants to shift from one side to the other (line 885). A user could call the `shiftTokens` function multiple times before the next price update to shift the staker's token from one side to the other with more than he has staked.
Staker.sol#L885

Recommend adding checks on `userNextPrice_amountStakedSyntheticToken_toShiftAwayFrom_long` and `userNextPrice_amountStakedSyntheticToken_toShiftAwayFrom_short` to ensure that the sum of the two variables does not exceed user's stake balance.

JasoonS (Float) confirmed:

> Yes, spot on! We spotted this the next morning after launching the competition. Token shifting was a last minute addition to the codebase. Really glad someone spotted it, but only in the last few hours, phew!

> This would allow a malicious user to completely shift all the tokens (even those not belonging to them to one side or the other!!) No funds could be stolen by the user directly (since the execution of those shifts would fail on the user level), but it could be done for personal gain (eg improving the users FLT issuance rate, or similar economic manipulation).

Medium Risk Findings (6)

[M-01] `latestMarket` used where `marketIndex` should have been used

Submitted by gpersoon, also found by 0xImpostor, cmichel, shw, hack3r-0m, jonah1005, loop and pauliax

The functions `initializeMarket` and `_seedMarketInitially` use the variable `latestMarket`. If these functions would be called seperately from `createNewSyntheticMarket`, then `latestMarket` would have the same value for each call of `initializeMarket` and `_seedMarketInitially`

This would mean that the `latestMarket` is initialized multiple times and the previous market(s) are not initialized properly. Note: the call to addNewStakingFund could have prevented this issue, but also allows this, see separate issue.

Note: the functions can only be called by the admin, so if `createNewSyntheticMarket` and `initializeMarket` are called in combination, then it would not lead to problems, but in future release of the software the calls to `createNewSyntheticMarket` and `initializeMarket` might get separated.

LongShort.sol #L304

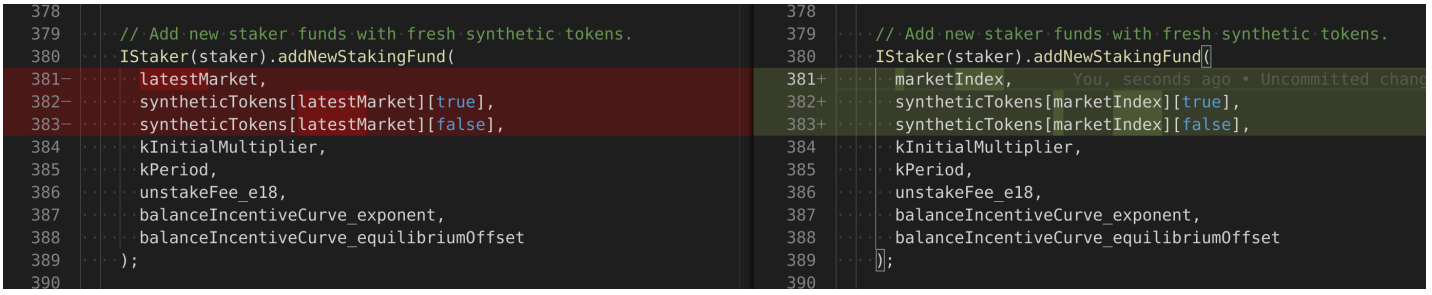
```
    function _seedMarketInitially(uint256 initialMarketSeedForEachMa
      ...
      ISyntheticToken(syntheticTokens[latestMarket][true]).mint(PERM
      ISyntheticToken(syntheticTokens[latestMarket][false]).mint(PER

    function initializeMarket(
        uint32 marketIndex,....)
    ...
      require(!marketExists[marketIndex], "already initialized");
      require(marketIndex <= latestMarket, "index too high");
      marketExists[marketIndex] = true;
    ..
      IStaker(staker).addNewStakingFund(
        `latestMarket`,                                          // shc
        syntheticTokens[latestMarket][true],   // should be marketIr
        syntheticTokens[latestMarket][false],  // should be marketIr
    ...
```

Recommend replacing `latestMarket` with `marketIndex` in the functions `initializeMarket` and `_seedMarketInitially`.

p.s. confirmed by Jason of float capital: Definitely an issue, luckily both of those functions are adminOnly. But that is definitely not ideal!

JasoonS (Float) confirmed:

> Great spot!



> Not a risk if you know about it (you just need to launch markets sequentially not in batches), but we didn't. So 2 - medium risk is fair :)

DenhamPreen (Float) patched:

> Resolved PR https://github.com/Float-Capital/monorepo/pull/1106

[M-02] Incorrect balance computed in `getUsersConfirmedButNotSettledSynthBalance()`

Submitted by hack3r-0m and cmichel

Consider the following state:

```
long_synth_balace = 300;
short_synth_balace = 200;

marketUpdateIndex[1] = x;
userNextPrice_currentUpdateIndex = 0;
userNextPrice_syntheticToken_toShiftAwayFrom_marketSide[1][true]
batched_amountSyntheticToken_toShiftAwayFrom_marketSide[1][true]
```

User calls `shiftPositionFromLongNextPrice(marketIndex=1, amountSyntheticTokensToShift=100)`

This results in following state changes:

```
long_synth_balace = 200;
short_synth_balace = 200;
userNextPrice_syntheticToken_toShiftAwayFrom_marketSide[1][true]
batched_amountSyntheticToken_toShiftAwayFrom_marketSide[1][true]
userNextPrice_currentUpdateIndex = x+1 ;
```

Due to some other transactions, oracle updates twice, and now the `marketUpdateIndex[1]` is x+2 and also updating price snapshots.

When User calls `getUsersConfirmedButNotSettledSynthBalance(user, 1)`

initial condition:

```
if (
  userNextPrice_currentUpdateIndex[marketIndex][user] != 0 &&
  userNextPrice_currentUpdateIndex[marketIndex][user] <= current
)
```

will be true; `LongShort.sol` **L532**

```
syntheticToken_priceSnapshot[marketIndex][isLong][currentMarketI
```

This uses price of current x+2 th update while it should balance of accounting for price of x+1 th update.

JasoonS (Float) confirmed and disagreed with severity:

> Yes good spot.

> This function is view only, and is only used for view only purposes. The rest of the system will always operate correctly because it rather uses

> `_executeOutstandingNextPriceSettlements` than the `getUsersConfirmedButNotSettledSynthBalance`. Therefore I propose this as a **1 Low Risk** vulnerability.

0xean (judge) (judge) commented:

> I am going to align with the 2 (Med Risk) severity. Reporting the incorrect position in a UI to a user could definitely lead unexpected loss of funds in a sharp market move where a user is intending on hedging elsewhere.

[M-03] Missing events/timelocks for owner/admin only functions that change critical parameters

Submitted by 0xRajeev, also found by tensors

Owner/admin only functions that change critical parameters should emit events and have timelocks. Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes with timelocks that allow users to evaluate them and consider if they would like to engage/exit based on how they perceive the changes as affecting the trustworthiness of the protocol or profitability of the implemented financial services. The alternative of directly querying on-chain contract state for such changes is not considered practical for most users/usages.

Missing events and timelocks do not promote transparency and if such changes immediately affect users' perception of fairness or trustworthiness, they could exit the protocol causing a reduction in liquidity which could negatively impact protocol TVL and reputation.

There are owner/admin functions that do not emit any events in `LongShort.sol`. It is not apparent that any owner/admin functions will have timelocks.

See similar High-severity H03 finding in OpenZeppelin's Audit of Audius and Medium-severity M01 finding OpenZeppelin's Audit of UMA Phase 4

See issue page for referenced code.

Recommend adding events to all owner/admin functions that change critical parameters. Add timelocks to introduce time delays for critical parameter changes that significantly impact market/user incentives/security.

JasoonS (Float) acknowledged:

> We will manage timelocks and multi-sigs externally to these contracts.

JasoonS (Float) commented:

> I would consider this a duplicate of #84 in many ways. (or at least #84 is a sub-issue of this issue)

Oxean (judge) commented:

> duplicate of #84 as both offer solutions for dealing with privileged functionality (including the transfer of ownership). Leaving severity as 2 based on the potential risks associated with an incorrect admin change or similar.

[M-04] Staker.sol: Wrong values returned in edge cases of `_calculateFloatPerSecond()`

Submitted by hickuphh3

In `_calculateFloatPerSecond()`, the edge cases where full rewards go to either the long or short token returns

`return (1e18 * k * longPrice, 0);` and

`return (0, 1e18 * k * shortPrice);` respectively.

This is however `1e18` times too large. We can verify this by checking the equivalent calculation in the 'normal case', where we assume all the rewards go to the short token, ie. `longRewardUnscaled = 0` and `shortRewardUnscaled = 1e18`. Plugging this into the calculation below,

`return ((longRewardUnscaled * k * longPrice) / 1e18, (shortRewardUnscaled * k * shortPrice) / 1e18);` results in

`(0, 1e18 * k * shortPrice / 1e18)` or `(0, k * shortPrice)`.

As we can see, this would result in an extremely large float token issuance rate, which would be disastrous.

The edge cases should return `(k * longPrice, 0)` and `(0, k * shortPrice)` in the cases where rewards should go fully to long and short token holders respectively.

JasoonS (Float) confirmed:

> Fix:

```
-return (1e18 * k * longPrice, 0);
+return (k * longPrice, 0);
```

> and

```
-return (0, 1e18 * k * shortPrice);
+return (0, k * shortPrice);
```

DenhamPreen (Float) patched:

> Resolved https://github.com/Float-Capital/monorepo/pull/1085

[M-05] Wrong aave usage of `claimRewards`

Submitted by jonah1005

Aave yield manager claims rewards with the payment token. According to aave's document, aToken should be provided. The aave rewards would be unclaimable.

YieldManager's logic in L161-L170

Reference: https://docs.aave.com/developers/guides/liquidity-mining#claimrewards

Recommend changing to

```
        address[] memory rewardsDepositedAssets = new address[](1);
        rewardsDepositedAssets[0] = address(aToken);
```

DenhamPreen (Float) confirmed:

> Great catch!

> This contract is going to be upgradable but really applicable within this context 👍

moose-code (Float) commented:

> Oof yeah! Good one :)

> Devil in those documentation details :)

> | Parameter Name | Type | Description |
> | --- | --- | --- |
> | `assets` | address[] | addresses of the asset that accrue rewards, i.e. aTokens or debtTokens.

To get a list of associated tokens per asset, see also `getReserveTokensAddresses()` in the Protocol Data Provider. |

[M-06] Prevent markets getting stuck when prices don't move

Submitted by gpersoon, also found by cmichel

Suppose there is a synthetic token where the price stays constant, for example:

- synthetic DAI (with a payment token of DAI the price will not move)
- binary option token (for example tracking the USA elections; after the election results there will be no more price movements)

In that case `assetPriceHasChanged` will never be true (again) and `marketUpdateIndex[marketIndex]` will never increase. This means the `_executeOutstandingNextPrice` * functions will never be executed, which means the market effectively will be stuck.

LongShort.sol L669

```
function `_updateSystemStateInternal`(uint32 marketIndex) i
  ...
  int256 newAssetPrice = IOracleManager(oracleManagers[mark
  int256 oldAssetPrice = int256(assetPrice[marketIndex]);
  bool assetPriceHasChanged = oldAssetPrice != newAssetPric

  if (assetPriceHasChanged || msg.sender == staker) {
    ....
    if (!assetPriceHasChanged) {
      return;
    }
    ....
    marketUpdateIndex[marketIndex] += 1;  // never reaches
```

```
1035
1036 function _executeOutstandingNextPriceSettlements(address us
1037   uint256 userCurrentUpdateIndex = userNextPrice_currentUpd
1038   if (userCurrentUpdateIndex != 0 && userCurrentUpdateIndex
1039     _executeOutstandingNextPriceMints(marketIndex, user, tr
1040     _executeOutstandingNextPriceMints(marketIndex, user, fa
1041     _executeOutstandingNextPriceRedeems(marketIndex, user,
1042     _executeOutstandingNextPriceRedeems(marketIndex, user,
1043     _executeOutstandingNextPriceTokenShifts(marketIndex, us
1044     _executeOutstandingNextPriceTokenShifts(marketIndex, us
```

Recommend enhancing `_updateSystemStateInternal` so that after a certain period of time without price movements (for example 1 day), the entire function is executed (including the `marketUpdateIndex[marketIndex] += 1;`)

JasoonS (Float) disputed:

> Good point, we thought the time since last update check wasn't necessary.

> I'll chat with the team about what they think the risk is. But I don't think it is 3 given that we don't plan to launch any assets that don't have regular change (so market would be stuck for a limited time - even if it is long).

> In a lot of ways our 'nextPriceExecution' model is designed for this case. Some more traditional markets close for the weekend and over night. Our mechanism

> means that users will be able to buy and trade these assets at any time and get the asset as soon as there is an update.

JasoonS (Float) commented:

> The more I think about this the more I think it is a safety feature. It is way more likely that if the oracle keeps returning the same value that something is broken (which means we can catch the issue before it negatively impacts the system by unfairly managing user funds or similar). If it really is stuck on the same value legitimately it can replace the OracleManager that is being used to help with that.

moose-code (Float) commented:

> Since there is no current plan for a binary market, and the system would definitely need other accommodations to allow a binary market, since next price execution etc, this doesn't make sense as an issue in this case. The system is built for markets where continuous price updates will occur.

> Agree with Jason, if not price update is occurring, there is likely an issue with the oracle, and our system is not failing even in light of this issue. It is effectively paused until a new price update is given. As Jason mentions we can use the oracle manager to fix this

0xean (judge) commented:

> Due to the ability to update the oracle, funds would not be lost, but it would be an availability risk (even if temporary) for the system. Based on that I am downgrading to a 2.

Low Risk Findings (15)

[L-01] Missing input validation on many functions throughout the code

Submitted by tensors, also found by 0xRajeev, JMukesh and pauliax

Many functions throughout `LongShort.sol` and `YieldManager.sol` have no simple checks for validating inputs. Below some examples are linked. See `LongShort.sol` L254, and `YieldManager.sol` L149.

Recommend simple validations like requiring non-zero address or checking that amounts are non-zero would fix this.

JasoonS (Float) disputed:

> 0 - non-critical

> Both examples given are not vulnerabilities since they are not public functions.

> L254 in LongShort.sol This is an `onlyAdmin` function. We purposely removed null checks from admin functions (we will only call these functions under highly controlled circumstances with via code (ie ethers.js) that checks for forgotten arguments).

> L149 in YieldManagerAave.sol This function is `longShortOnly` so ONLY the LongShort contract. If you can find a bug (an actual example) in the `LongShort.sol` contract where it can give the incorrect arguments to the `YieldManagerAave` contract then this could be a valid issue.

0xean (judge) commented:

> given the downside of not having these checks in place and for example the admin being set to a null address there seems little benefit to not having them. Downgrading to 1

[L-02] Comment-code mismatch for `_balanceIncentiveCurve_exponent` threshold

Submitted by 0xRajeev, also found by hickuphh3

The code comment says: "// The exponent has to be less than 5 in these versions of the contracts." but the code immediately after the comment implements a check "< 6." It is unclear if the comment is incorrect or the check is wrong. An incorrect check may have mathematical implications. Staker.sol L276-L277

Recommend revisiting comment and code to sync them by fixing the comment or the code whichever is incorrect.

JasoonS (Float) acknowledged and disagreed with severity:

> Thanks - has been pointed out before. **0 non-critical**

0xean (judge) commented:

> Per https://docs.code4rena.com/roles/wardens/judging-criteria#estimating-risk-tl-dr - comments are a 1 (Low Risk). Agreeing with the warden here.

[L-03] Use of floating pragma

Submitted by JMukesh

https://swcregistry.io/docs/SWC-103

`ILendingPool.sol` have floating pragma and its been used in `YieldManger.sol` L9.

Recommend using fixed solidity version

JasoonS (Float) disagreed:

> We want you to have fixed versions.

> **0 non-critical**

> 0xean (judge) commented: @JasoonS (Float) - I am not sure I understand your comment, the warden is stating that ILendingPool *should* have a fixed version.

> Leaving as 1 and valid for now.

[L-04] prevent reentrancy

Submitted by gpersoon, also found by hickuphh3

If the payment token would be an ERC777 token (or another token that has callbacks), then an reentrancy attack could be tried. Especially in `function_executeOutstandingNextPriceSettlements` multiple transfers are

called, which could call callbacks. These callbacks could go to an attacker contract which could call functions of the `LongShort.sol` contract

Although I haven't found a scenario to misuse the reentrancy its better to prevent this. `LongShort.sol` **#L1035**

```
function _executeOutstandingNextPriceSettlements(address user, u
    uint256 userCurrentUpdateIndex = userNextPrice_currentUpdate
    if (userCurrentUpdateIndex != 0 && userCurrentUpdateIndex <=
      _executeOutstandingNextPriceMints(marketIndex, user, true)
      _executeOutstandingNextPriceMints(marketIndex, user, false
      _executeOutstandingNextPriceRedeems(marketIndex, user, tru
      _executeOutstandingNextPriceRedeems(marketIndex, user, fal
      _executeOutstandingNextPriceTokenShifts(marketIndex, user,
      _executeOutstandingNextPriceTokenShifts(marketIndex, user,

      userNextPrice_currentUpdateIndex[marketIndex][user] = 0;

      emit ExecuteNextPriceSettlementsUser(user, marketIndex);
    }
  }
```

Recommend preventing reentrancy attacks in one of the following ways:

- make sure the payment tokens don't have call back function / are not ERC777
- or add reentrancy guards to _executeOutstandingNextPriceSettlements (see ReentrancyGuard.sol)

JasoonS (Float) disputed:

> Very good point! We are aware of this, and will make sure of this for V2 for sure.

> We wrote in the readme that we will only use DAI as a payment token (there are lots of weird types of tokens that could break our system, not only tokens with hooks!)

> See readme: https://github.com/code-423n4/2021-08-floatcapital/blob/bd419abf68e775103df6e40d8f0e8d40156c2f81/README.md#L156

> I now see that I didn't mention 'hooks' in that comment, but it does say we will deeply analyse any potential payment token and only use DAI initially. I'll leave this to the judges.

Oxean (judge) commented:

> Given the readme doesn't specifically mentions hooks, I think the warden is highlighting a good thing for the sponsor to consider in future iterations. Leaving current severity.

[L-05] `PERMANENT_INITIAL_LIQUIDITY_HOLDER` not 100% safe

Submitted by gpersoon

The initial tokens are minted to the address `PERMANENT_INITIAL_LIQUIDITY_HOLDER` The comments suggest they can never be moved from there. However `transferFrom` in `SyntheticToken.sol` allows `longShort` to move tokens from any address so also from address `PERMANENT_INITIAL_LIQUIDITY_HOLDER`.

This is unlikely to happen because the current source of `LongShort.sol` doesn't allow for this action. However `LongShort.sol` is upgradable to in theory a future version could allow this. **LongShort.sol L34**

```
    /// @notice this is the address that permanently locked init
    /// These tokens will never move so market can never have ze
    /// @dev f10a7 spells float in hex - for fun - important par
    address public constant PERMANENT_INITIAL_LIQUIDITY_HOLDER =
```

```
304
305 function _seedMarketInitially(uint256 initialMarketSeedForEa
306 ...
307   ISyntheticToken(syntheticTokens[latestMarket][true]).mint(
308   ISyntheticToken(syntheticTokens[latestMarket][false]).mint
```

`SyntheticToken.sol` **L91**

```
function transferFrom(address sender, address recipient, uint256
  if (recipient == longShort && msg.sender == longShort) {    //
    super._transfer(sender, recipient, amount);
    return true;
  } else {
    return super.transferFrom(sender, recipient, amount);
  }
}
```

Recommend accepting the risk and document this in the contract. Or, update `transferFrom` to contain the following:

```
if (recipient == longShort && msg.sender == longShort && sender!
```

JasoonS (Float) acknowledged and disagreed with severity:

> Note, even if we were to remove this initial liquidity there is no issue as long as the market has users on both sides.

> This initial liquidity is solely for the first few minutes (or however long) of the market when there is no external users yet.

> If the market ever has zero users (zero additional liquidity) and we were to somehow force those last tokens to be redeemed, 'zero users' would care that the market has collapsed and is non-functional anymore.

moose-code (Float) commented:

> Interesting point but its way too far fetched we upgrade the smart contract to include a function that would ever call transferFrom from the very awkward address 0xf10A7*F10A7*f10A7*F10a7*f10A7*f10a7*f10A7_f10a7

JasoonS (Float) commented:

> tldr: there are far worse things that we could upgrade into the contracts than this where there is basically nothing to gain anyway.

Oxean (judge) commented:

> Leaving as 1, sponsor acknowledges it's an edge case that could cause non functioning markets which certainly qualifies as "state handling" in

```
1 — Low: Low: Assets are not at risk. State handling, function i
```

[L-06] Staker.sol: Updating `kValue` requires interpolation with initial timestamp

Submitted by hickuphh3

Updating a `kValue` of a market requires interpolation against the initial timestamp, which can be a hassle and might lead to a wrong value set from what is expected.

Consider the following scenario:

- Initially set `kValue = 2e18`, `kPeriod = 2592000` (30 days)
- After 15 days, would like to refresh the market incentive (start again with `kValue = 2e18`), lasting another 30 days.

In the current implementation, the admin would call `_changeMarketLaunchIncentiveParameters()` with the following inputs:

- `period = 3888000` (45 days)
- `kValue` needs to be worked backwards from the formula

  `kInitialMultiplier - (((kInitialMultiplier - 1e18) * (block.timestamp - initialTimestamp)) / kPeriod)`. To achieve the desired effect, we would get `kValue = 25e17` (formula returns 2e18 after 15 days with kPeriod = 45 days).

This isn't immediately intuitive and could lead to mistakes.

Recommend that Instead of calculating from `initialTimestamp` (when `addNewStakingFund()` was called), calculate from when the market incentives were

last updated. This would require a new mapping to store last updated timestamps of market incentives.

For example, using the scenario above, refreshing the market incentive would mean using inputs `period = 2592000` (30 days) with `kValue = 2e18`.

```
// marketIndex => timestamp of updated market launch incentive p
mapping(uint32 => uint256) public marketLaunchIncentive_update_t

function _changeMarketLaunchIncentiveParameters(
  uint32 marketIndex,
  uint256 period,
  uint256 initialMultiplier
) internal virtual {
        require(initialMultiplier >= 1e18, "marketLaunchIncentiv

  marketLaunchIncentive_period[marketIndex] = period;
  marketLaunchIncentive_multipliers[marketIndex] = initialMultip
        marketLaunchIncentive_update_timestamps[marketIndex] = b
};

function _getKValue(uint32 marketIndex) internal view virtual re
  // Parameters controlling the float issuance multiplier.
  (uint256 kPeriod, uint256 kInitialMultiplier) = _getMarketLaun

  // Sanity check - under normal circumstances, the multipliers
  // *never* be set to a value < 1e18, as there are guards agair
  assert(kInitialMultiplier >= 1e18);

        // currently: uint256 initialTimestamp = accumulativeFlo
        // changed to take from last updated timestamp instead o
  uint256 initialTimestamp = marketLaunchIncentive_update_timest

  if (block.timestamp - initialTimestamp <= kPeriod) {
    return kInitialMultiplier - (((kInitialMultiplier - 1e18) *
  } else {
    return 1e18;
  }
}
```

JasoonS (Float) confirmed:

> You are right. we should probably just delete the external `changeMarketLaunchIncentiveParameters` function so that it can only be set once.

[L-07] TokenFactory.sol: DEFAULT_ADMIN_ROLE has wrong value

Submitted by hickuphh3

`TokenFactory.sol` defines `DEFAULT_ADMIN_ROLE = keccak256("DEFAULT_ADMIN_ROLE");` , but OpenZeppelin's `AccessControl.sol` defines `DEFAULT_ADMIN_ROLE = 0x00` , so that by default, all other roles defined will have their admin role to be `DEFAULT_ADMIN_ROLE` .

This makes the following lines erroneous:

```
// Give minter roles
SyntheticToken(syntheticToken).grantRole(DEFAULT_ADMIN_ROLE, lor
SyntheticToken(syntheticToken).grantRole(MINTER_ROLE, longShort)
SyntheticToken(syntheticToken).grantRole(PAUSER_ROLE, longShort)

// Revoke roles
SyntheticToken(syntheticToken).revokeRole(DEFAULT_ADMIN_ROLE, ac
SyntheticToken(syntheticToken).revokeRole(MINTER_ROLE, address(t
SyntheticToken(syntheticToken).revokeRole(PAUSER_ROLE, address(t
```

Due to how `grantRole()` and `revokeRole()` works, the lines above will not revert. However, note that `TokenFactory` will have `DEFAULT_ADMIN_ROLE (0x00)` instead of `LongShort` . This by itself doesn't seem to have any adverse effects, since `TokenFactory` doesn't do anything else apart from creating new synthetic tokens.

Nonetheless, I believe that `DEFAULT_ADMIN_ROLE` was unintentionally defined as `keccak256("DEFAULT_ADMIN_ROLE")` , and should be amended.

The revoking role order will also have to be swapped so that `DEFAULT_ADMIN_ROLE` is revoked last.

```
bytes32 public constant DEFAULT_ADMIN_ROLE = 0x00;

function createSyntheticToken(
  string calldata syntheticName,
  string calldata syntheticSymbol,
  address staker,
  uint32 marketIndex,
  bool isLong
) external override onlyLongShort returns (ISyntheticToken synth
      ...
  // Revoke roles
  _syntheticToken.revokeRole(MINTER_ROLE, address(this));
  _syntheticToken.revokeRole(PAUSER_ROLE, address(this));
      _syntheticToken.revokeRole(DEFAULT_ADMIN_ROLE, address(t
}
```

[JasoonS (Float) acknowledged]:

> Thanks they changed that more recently (or bad copy pasting...)

[L-08] `YieldManagerAave.sol` : Wrong branch in `depositPaymentToken()` if `amountReservedInCaseOfInsufficientAaveLiquidity == amount`

Submitted by hickuphh3

In the unlikely event `amountReservedInCaseOfInsufficientAaveLiquidity == amount`, the `else` case will be executed, which means `lendingPool.deposit()` is called with a value of zero. It would therefore be better to change the condition so that the `if` case is executed instead.

```
function depositPaymentToken(uint256 amount) external override
  // If amountReservedInCaseOfInsufficientAaveLiquidity isn't ze
  // It basically always be zero besides extreme and unlikely si
  if (amountReservedInCaseOfInsufficientAaveLiquidity != 0) {
              // instead of strictly greater than
    if (amountReservedInCaseOfInsufficientAaveLiquidity >= amour
      amountReservedInCaseOfInsufficientAaveLiquidity -= amount;
```

```
            // Return early, nothing to deposit into the lending pool
            return;
        }
            ...
    }
```

JasoonS (Float) confirmed:

> Hmm, yes, the deposit function in aave will revert if zero right?

> Thanks for reporting.

[L-09] `LongShort` should not shares the same Yield Manager between different markets

Submitted by jonah1005, also found by cmichel and gpersoon

`LongShort` should not shares the same Yield Manager between different markets The `LongShort` contract would not stop different markets from using the same yield manager contracts. Any extra aToken in the yield manager would be considered as market incentives in function `distributeYieldForTreasuryAndReturnMarketAllocation`. Thus, using the same yield manager for different markets would break the markets and allow users to withdraw fund that doesn't belong to them. `YieldManagerAave.sol` **L179-L204**

Given the fluency of programming skills the dev shows, I believe they wouldn't make this mistake on deployment. Still, I think there's space to improve in the YieldManagerAave contract. IMHO. As it's tightly coupled with `longshort` contract and its market logic, a initialize market function in the yield manager seems more reasonable.

JasoonS (Float) confirmed:

> Duplicate #10

> I agree there is no way I would actually make that mistake. But technically possible, so that is a fair comment.

> Why not have some code in the yield manager like the following (pseudo code, not tested):

```
boolean isInitialized;
function initializeForMarket() onlyLongShort {
        require(!isInitialized, "Yield Manager is already in use
    isInitialized = true;
}
```

> And that function gets called by long short.

DenhamPreen (Float) resolved:

> https://github.com/Float-Capital/monorepo/pull/1139

[L-10] The address of Aave `lendingPool` may change

Submitted by pauliax

Contract `YieldManagerAave` caches `lendingPool`, however, in theory, it is possible that the implementation may change (see L58-L65 of Aave's `LendingPoolAddressesProvider.sol`). I am not sure how likely in practice is that but a common solution that I see in other protocols that integrate with Aave is querying the `lendingPool` on the go (of course then you also need to handle the change in approvals).

An example solution you can see here.

JasoonS (Float) sponsor:

> Thanks

[L-11] confusing comments

Submitted by gpersoon, also found by 0xImpostor

I've seen comments which are confusing: `~10^31 or 10 Trillion (10^13)` ==> probably should be 2^31 `x * 5e17 ==` (x * 10e18) / 2`==> probably should be 1e18/2

// https://github.com/code-423n4/2021-08-floatcapital/blob/main/contracts/contracts/Staker.sol#L19 // 2^52 ~= 4.5e15 // With an exponent of 5, the largest total liquidity possible in a market (to avoid integer overflow on exponentiation) is ~10^31 or 10 Trillion (10^13)

//https://github.com/code-423n4/2021-08-floatcapital/blob/main/contracts/contracts/Staker.sol#L480 // NOTE: `x * 5e17 == (x * 10e18) / 2`

Recommend double checking the comments.

JasoonS (Float) confirmed:

> ~10^31 or 10 Trillion (10^13) ==> probably should be 2^31

> This is just an approximation to justify why the values we chose are safe. Assuming a maximum of 10 Trillion DAI (very very conservative upper estimate to market size), then that would be 10^31 DAI decimal units (so not 2^31). Can clarify this, thanks!

> x * 5e17==(x * 10e18) / 2`==> probably should be 1e18/2

> Yes, we picked this one up, it should be 1e18 :)

Oxean (judge) commented:

> Based on C4's documentation this is a 1

```
1 — Low: Low: Assets are not at risk. State handling, function i
```

[L-12] Docstring

Submitted by evertkors, also found by loop

A lot of docstrings for `marketIndex` are @param marketIndex An int32 which uniquely identifies a market. but it is a `uint32` not an `int32`

Oxean (judge) commented:

> based on https://docs.code4rena.com/roles/wardens/judging-criteria#estimating-risk-tl-dr upgrading this to 1

[L-13] Possibly not all synths can be withdrawn

Submitted by cmichel

The `LongShort._handleTotalPaymentTokenValueChangeForMarketWithYieldManager` function assumes that the `YieldManager` indeed withdraws all of the desired payment tokens, but it could be that they are currently lent out at Aave.

```
    // NB there will be issues here if not enough liquidity exists t
    // Boolean should be returned from yield manager and think how t
    IYieldManager(yieldManagers[marketIndex]).removePaymentTokenFrom
      uint256(-totalPaymentTokenValueChangeForMarket)
    );
```

Recommend trying to withdraw these tokens will then fail.

> Boolean should be returned from yield manager and think how to appropriately handle this 😁

JasoonS (Float) disputed:

> That happens on the user level, they can just try again later (our UI will give a good message).

> It was designed like this.

Oxean (judge) commented:

> based on the incorrect comment in the code I am going to leave as a 1 (per - https://docs.code4rena.com/roles/wardens/judging-criteria#estimating-risk-tl-dr)

> but it does look like the system is designed to handle this scenario regardless of the boolean returned (or not being returned)

[L-14] Protocol requires a running bot in order to make sure trades are actually executed

Submitted by tensors

Because smart contracts need to be poked to execute, trades placed before an oracle update won't be executed until someone else calls the function to execute queued trades. This means that a bot must run to constantly execute trades after every oracle update.

If such a bot was not running, users would have an incentive to only execute their trades after a favorable oracle update. However, having a dedicated bot run by the team centralizes the project with a single failure point. The typical solution here is to create keeper incentives for the protocol.

Recommend either making sure the team has a bot running or preferably create incentives for other users to constantly keep the queued orders executing.

JasoonS (Float) disputed:

> **0 - non-critical**

> In the worst case of the bot going down, no vulnerabilities open up.

> The only issue that arises is that the price won't be tracked as accurately during those periods (but it will still track it, just not as accurately).

> The protocol allows anyone to call an update, and any user interaction (non-update interaction) will also call the update if an update is due.

> If such a bot was not running, users would have an incentive to only execute their trades after a favorable oracle update.

> That is great, then the incentives are working, they cannot do better than a fair price update which is 'fair' in my opinion.

> Indeed: We have been in talks with the chainlink keeprs team to be the first project to use them when they launch on polygon. We should have mentioned this in the README. Until then we have built a robust bot.

> Note, this bot is for UX, not to patch up a vulnerability.

moose-code (Float) commented:

> Agree with @JasoonS (Float) . The bot failing opens no vulnerabilities, the synthetics may just less closely track the underlying if no contract interactions are present.

> Given rational markets with actors on both sides (long and short), and given it will always be beneficial for one side to execute the update to capitalize on the movement, its safe to assume that in a rational market the updateSystemState in every case should be called by at least one participant, meaning theoretically a bot isn't even necessary.

Stentonian commented:

> trades placed before an oracle update won't be executed until someone else calls the function to execute queued trades.

> What function is this? There are multiple update functions so it's not clear which one the warden is referring to. If they are referring to the function `_executeOutstandingNextPriceSettlements` (the one that they linked in the 'Proof of concept' section) then the warden's statement is a non-issue. The system is designed to do exactly what the warden says. So the next statement is incorrect:

> This means that a bot must run to constantly execute trades after every oracle update.

> LongShort keeps track of which trade should happen at which price for each user, so when `_executeOutstandingNextPriceSettlements` is eventually called (no matter how many price updates have occurred since the trade request) it will use the oracle's next price update that occurred **right after the time when trade was requestd.** And `_executeOutstandingNextPriceSettlements` is called before any other action is taken by the user that would require the data from the trade having completed. No need to have a bot call this function.

> If the warden is referring to another update function then it's not clear which.

> The only issue that arises is that the price won't be tracked as accurately during those periods (but it will still track it, just not as accurately).

> This is not actually mentioned by the warden in the issue description, even tho it is a real problem. So I don't think we need to include it here. The warden is specifically pointing out `executeOutstandingNextPriceSettlements` which will never have any issues no matter if there is a bot or not.

Oxean (judge) commented:

> There are incentives built into the market already for users to update if the bot was to stop running. Yes, the bot should be treated as a tier-1 service for a reasonable user experience and for the most efficient tracking, but the risk of this to the system as a whole locking user funds or allowing people to significantly and unfairly profit doesn't seem to be there.

> All that being said, it is a failure point within the system but one with low risk, downgrading to 1 - Low Risk.

[L-15] Race-condition risk with initialize functions

Submitted by 0xRajeev, also found by cmichel

Race-condition risk with initialize functions if deployment script is not robust to create and initialize contracts atomically or if factory contracts do not create and initialize appropriately.

If this is not implemented correctly, an attacker can front-run to initialize contracts with their parameters. This, if noticed, will require a redeployment of contracts resulting in potential DoS and reputational damage. See Short.sol L188-L193, FloatToken.sol L21-L25, and Staker.sol L179-L186.

Recommend ensuring deployment script is robust to create and initialize contracts atomically or factory contracts create and initialize appropriately.

JasoonS (Float) disputed:

> We use open-zeppelin scripts todo this automatically.

> Additionally we initialize the base implementations too to prevent any foul play by pranksters.

Oxean (judge) commented:

> Given that the contest didn't include the scope of the scripts and that this is a risk in the contract implementation without a factory I believe this is a valid risk even if the sponsor believes its mitigated.

Non-Critical Findings (25)

- [N-01] Markets cannot be initialized with payment tokens of few decimals
- [N-02] Assuming tokens are compliant with ERC20 could cause transactions to revert unexpectedly
- [N-03] Received amount of transfer-on-fee/deflationary tokens are not correctly accounted
- [N-04] Interface notations are used for abstract contracts
- [N-05] extra safety in distributeYieldForTreasuryAndReturnMarketAllocation
- [N-06] consistently use `msg.sender` or `_msgSender()` (recommended)
- [N-07] 0xf10A7*F10A7*f10A7*F10a7*f10a7_f10a7
- [N-08] Constant values used inline
- [N-09] emit event at stage changes
- [N-10] Index Events
- [N-11] LongShort.sol & YieldManagerAave.sol: Verify / derive input arguments
- [N-12] `LongShort.sol` : Inconsistency in _claimAndDistributeYieldThenRebalanceMarket()
- [N-13] Single Source of Truth
- [N-14] Spelling Errors
- [N-15] Staker.sol: Shift event emissions to internal functions
- [N-16] Staker.sol: TODO add link in comment

- [N-17] Staker.sol: withdrawAll() does not include incoming outstanding shifts to the user
- [N-18] Multiple initialize functions
- [N-19] FloatToken would revoke stakerAddress's permission if msg.sender == stakerAddress
- [N-20] Aave's claimRewards returns the actual rewards claimed
- [N-21] Style issues
- [N-22] executeOutstandingNextPriceSettlementsUserMulti may exceed gas limits
- [N-23] Missing use of requireMarketExists modifier on multiple functions
- [N-24] Solution is susceptible to MEV, harming users.
- [N-25] Oracle updates can be frontrun by stakers to gain a profit

Gas Optimizations (21)

- [G-01] Pass time delta into internal functions
- [G-02] Staker.sol: Cache `marketIndex`
- [G-03] Caching state variables in local variables can save gas
- [G-04] Unused named returns can be removed for optimization
- [G-05] Function visibility can be changed from public to external
- [G-06] Immutable Variables
- [G-07] Gas: `SyntheticToken` does not use pausing functionality
- [G-08] Internal _withdraw, reading from storage twice.
- [G-09] slight difference between withdraw and withdrawAll
- [G-10] Drop require checks for synthetic tokens
- [G-11] Increase Solc Optimiser Runs
- [G-12] `LongShort.sol` : Cache marketUpdateIndex[marketIndex]
- [G-13] `LongShort.sol` : Some math can be unchecked in _getYieldSplit()
- [G-14] Staker.sol: Cache shift amounts
- [G-15] Staker.sol: Redundant zero intialization for accumulativeFloatPerSyntheticTokenSnapshots

- [G-16] TokenFactory.sol: Appropriate type declaration to avoid numerous casting
- [G-17] [Optimization] Cache length in the loop
- [G-18] Appropriate storage variable type declaration to save on casting
- [G-19] Cache storage access and duplicate calculations
- [G-20] treasury state variable in `LongShort`
- [G-21] onlyValidMarket is never used

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