DRAFT Secureum CARE Report
Increment

March 2022

CARE: Why

CARE stands for "Comprehensive Audit Readiness Evaluation." CARE is *not* a replacement for a security audit, but is intended to happen before an audit so that protocol code becomes ready for future audit(s) to get a better security outcome from the process.

CARE reviews protocol code mainly for common security pitfalls and best-practices as related to smart contracts written in Solidity specifically for Ethereum blockchain or associated Layer-2 protocols. The pitfalls & best-practices are evaluated from (but not limited to) Secureum's Security Pitfalls & Best Practices 101 and 201.

CARE aims to help identify such common pitfalls & best-practices so that they can be fixed before audit(s). This improves protocol's risk posture earlier in the design & development lifecycle and enables future audit(s) to focus more on deeper/harder application-specific and economic vulnerabilities. CARE helps smart contract security "shift-left" which is widely regarded as significantly improving security posture and outcome.

CARE reviews are performed by "CAREtakers" which includes a Secureum representative (who has a proven track-record of smart contract security expertise/experience) along with invited participants who are top-performing members of the Secureum community and aspiring smart contract security experts. They are invited based on their performance in Secureum RACEs.

CARE: When, Who & What

Dates: 31st March - 6th April, 2022

CAREtakers (Discord handles): blockdev#0246, neumo#7347, teryanarmen#2961, @Đravee, @yash90, @Kenshin, @Bahurum, Deivitto#9076, @tin537, @popular, baraa#7157 & @Rajeev | Secureum

Scope: Review focused on this repository/commit: https://github.com/Increment-Finance/increment-secureum and some immediately related dependencies.

Note: If there were changes/commits after that across repositories, some of them may have been used in Github Permalinks below but not necessarily reviewed.

CARE-X Security Pitfalls & Best-Practices: Checklist

1. **Single-step/direct transfer of ownership is risky**
    a. **Summary**: Single-step change of ownership of contracts is risky.
    b. **Details**: `Vault.sol`, `Insurance.sol`, and `ClearingHouse.sol` inherit from `IncreOwnable.sol` which implements the `transferOwner` function. Although this function does have a two-step change option, it also has a single-step change option. When the parameter `direct` is passed a value `true`, the ownership will be transferred directly. There is a possibility that direct transfer of ownership is done accidentally and to an incorrect address.
    c. **Github Permalinks**: `transferOwner`
    d. **Mitigation**: Re-consider the utility of direct transfer of ownership when a two-step change mechanism is also there in the `transferOwner` function.
2. **Mixing of (unused) named and explicit returns affects readability**
    a. **Summary**: Mixing of named and explicit returns affects readability and is especially confusing when named returns are unused.
    b. **Details**:
        i. in `ClearingHouseViewer.sol`, `getFundingPayments` function has both named and explicit returns. This reduces code readability.
        ii. In `Perpetual.sol`, several functions have both named returns and an explicit return statement, which is redundant.
    c. **Github Permalinks**:
        i. `getFundingPayments`
        ii. `extendPosition`, `reducePosition`, `provideLiquidity`, `settleLiquidityProvider`, `marketPrice`, `getFundingPayments`, `_settleFundingRate`, `_settleLpFundingRate`, `_checkProposedAmount`, `_getPositionDirection`
    d. **Mitigation**: Reconsider the use of named returns and instead use explicit return statements with values to increase readability
3. **No way to remove or pause a Perpetual Market**
    a. **Summary**: Once a perpetual market is allowlisted, it cannot be removed.
    b. **Details**: A perpetual market has to be allowlisted by `ClearingHouse owner.allowListPerpetual` which pushes the perpetual in `perpetuals` array. After that, there is no way to pause a specific perpetual or remove it. The only possible mitigation, if a vulnerability is discovered in a specific perpetual, is to pause the entire protocol.
    c. **Github Permalinks**: `allowListPerpetual`
    d. **Mitigation**: Consider adding functionality to remove/pause a specific perpetual.
4. **Code, comments, and documentation**
    a. **Summary**: Discrepancies in/between or inaccuracies/deficiencies in code, comment, and documentation can be misleading and could indicate the presence of inaccurate implementation or documentation.
    b. **Details**:

i. Threshold value mismatch between documentation and code. Documentation mentions that for any action by a trader or liquidity provider, the maximum accepted price deviation is 2%, but code checks for a 5% price deviation. Update the documentation or change `MAX_PRICE_DEVIATION` to `2e16`.
ii. Incorrect Natspec comment for `reducePosition`. `reducePosition` allows traders to reduce or close their position, but the comment only mentions closing the position. Update the comment to: "Reduces/Closes position from account holder."
iii. NatSpec comments for `extendPositionWithCollateral` do not describe return values.
iv. NatSpec comments are absent for `getUnrealizedPnL`.
v. Missing NatSpec @param for `getLpBalance user.`
vi. NatSpec comments for `getProposedAmount` do not describe return value.

c. **Github Permalinks**:
i. `MAX_PRICE_DEVIATION`, Documentation
ii. `reducePosition`
iii. `extendPositionWithCollateral`
iv. `getUnrealizedPnL`
v. `getLpBalance`
vi. `getProposedAmount`

d. **Mitigation**: Code, comments, and documentation should all be accurate and consistent.

5. **Maximum comment line length exceeded**
a. **Summary**: Long comments should be wrapped to conform with Solidity Style guidelines.
b. **Details**: Lines 42, 52, 358, 366, 419, 420, 597 and some other lines in `ClearingHouse.sol;` Line 44 in `ClearingHouseViewer.sol;` Line 203 `Perpetual.sol;` Line 261 in `Vault.sol;` exceed the 79 (or 99) character length suggested by the Solidity Style guidelines.
c. **Github Permalinks**: `ClearingHouse`, `ClearingHouseViewer`, `Perpetual.sol`, `Vault.sol`
d. **Mitigation**: Lines mentioned should be wrapped to a maximum of 79 (or 99) characters to help readers easily parse the comments.

6. **Division by zero**
a. **Summary**: In `Perpetual.sol` there is a potential division by zero that could revert without a reason.
b. **Details**: In `Perpetual.sol` function `_getVBasePositionAfterVirtualWithdrawal` does not check `getTotalLiquidityProvided()` value before division. This could lead to a division by zero revert without returning a descriptive message to the user.
c. **Github permalinks**: `_getVBasePositionAfterVirtualWithdrawal`
d. **Mitigation**: Add `require(getTotalLiquidityProvided() > 0, "Division by zero")`

7. **Potential miscalculation of funding payments**

a. **Summary**: The calculation of funding payments in `Perpetual.sol` may be incorrect.
b. **Details**: In `Perpetual.sol` the first parameter of the calls to `_getFundingPayments` is calculated with three different methods. The parameter, `isLong`, is mostly assigned the result of the call to `_getPositionDirection`, but in one case it is assigned true if the call to `_getVBasePositionAfterVirtualWithdrawal` is greater than zero and false otherwise, and in other case it is assigned true if `user.positionSize` is greater than zero and false otherwise. Both functions take the user position struct as their only parameter. It could happen that, if `user.liquidityBalance` is zero and `user.positionSize` is one, the call to `_getPositionDirection` would return true (so `isLong` = true), but the call to `_getVBasePositionAfterVirtualWithdrawal` would return zero, so `isLong` would be false. The call to `_getFundingPayments` using the value of `user.positionSize` could also imply a different value for `isLong`, because using `_getPositionDirection` or `_getVBasePositionAfterVirtualWithdrawal` could both return different values for `isLong`, because they not only take into account the `positionSize` but also the `liquidityBalance` of the user.
c. **Github permalinks**:
    i. `_getPositionDirection`
    ii. `_getVBasePositionAfterVirtualWithdrawal`
    iii. Call to `_getFundingPayments` using `_getPositionDirection` to calculate `isLong`
    iv. Call to `_getFundingPayments` using `_getVBasePositionAfterVirtualWithdrawal` to calculate `isLong`
    v. Call to `_getFundingPayments` using `user.positionSize` greater than zero to calculate `isLong`
d. **Mitigation**: Ensure that for a given `UserPosition` struct, the direction calculated is always the same (it shouldn't be Short or Long depending on the method used to calculate it). Consider using `virtualPositionSize` instead of `positionSize` as appropriate.

8. **Public function visibility can be made external**
    a. **Summary**: Functions should have the strictest visibility possible. Public functions may lead to more gas usage by forcing the copy of their parameters to memory from calldata.
    b. **Details**: If a public function is never called from the contract it should be marked as external. This will save gas.
    c. **Github permalinks**: `approve` and `transfer`
    d. **Mitigation**: Change visibility from `public` to `external`.
9. **Redundant parameter**
    a. **Summary:** Use of deposit and withdraw token parameters may be redundant.

b. **Details**: `depositToken` and `withdrawToken` parameters in `deposit`, `withdraw`, `withdrawAll` and `withdrawPartial` functions in Vault are redundant because they can only be equal to `reserveToken` which is already stored in the contract.
c. **Github Permalinks**: `deposit`, `withdraw`
d. **Mitigation**: Consider removing this parameter to simply use `reserveToken` instead.

10. Missing check for special indices

a. **Summary:** Missing check for special indices 0 or 1 may allow users to accidentally have their tokens locked.
b. **Details**: Indices 0 and 1 of `traderBalances` have a special meaning in the Vault code where they are used to track insurance reserves of the protocol and profit earned by governance from selling dust respectively. However functions in `ClearingHouse` allow users to call them with these special indices 0 or 1 which are irrelevant to addresses other than `clearingHouse`. This can allow users to accidentally call these functions with the special indices leading to their tokens getting locked in the protocol.
c. **Github Permalinks**: Comment
d. **Mitigation**: Consider adding a check such as `require(idx > 1)` wherever necessary to prevent users from calling functions with these special indices 0 or 1.

11. Test code in production

a. **Summary**: Contracts import Hardhat `console.sol` library in production.
b. **Details**: Testing related code should be removed from production. Contracts import Hardhat `console.sol` library that is used for testing which should be removed before production deployment.
c. **Github Permalinks**: `ClearingHouse.sol#L23`, `Perpetual.sol#L19`, `Vault.sol#L20`
d. **Mitigation**: Remove imports of Hardhat `console.sol` before production deployment.

12. Custom Errors can be used

a. **Summary**: Given the use of Solidity version 0.8.4, custom errors can be used.
b. **Details**: Solidity v0.8.4 introduced a convenient and gas-efficient way to explain to users why an operation failed through the use of custom errors. They provide an alternative to using revert strings which can be gas expensive and cannot capture dynamic information.
More:
c. **Github Permalinks**: Custom errors
d. **Mitigation**: Consider using custom errors.

13. Unused imports

a. **Summary**: Unused imports increase gas costs and affect readability.
b. **Details**: Some contracts and interfaces imported are never used. Any unused imports and inherited contracts should be removed or used appropriately after careful evaluation. This will not only reduce gas costs but improve readability and maintainability of the code.

c. **Github Permalinks**: ClearingHouse.sol#L6, ClearingHouse.sol#L42, Perpetual.sol#L5, Vault.sol#L6, VirtualToken.sol#L9, VBase.sol#L6, VQuote.sol#L8
d. **Mitigation**: Evaluate and remove all unused imports.

14. Redundant check on `vault.deposit` return value

a. **Summary**: Function `_deposit` checks `vault.deposit` return value with a `require` which will always pass.
b. **Details**: `vault.deposit` already requires `wadAmount.toInt256()>=MIN_DEPOSIT_AMOUNT` before returning `wadAmount` and so the returned value will strictly be positive (`MIN_DEPOSIT_AMOUNT` is constant > 0, so `wadAmount` will always be > 0). Checking this return value in `_deposit` of `ClearingHouse.sol` with a `require` is redundant.
c. **Github Permalinks**: ClearingHouse.sol#L514, Vault.sol#L103
d. **Mitigation**: Remove the require statement.

15. Unnecessary type conversions affect readability

a. **Summary**: Variables of a given type are explicitly converted to their own type.
b. **Details**: In `VirtualToken.sol` functions `mint` and `burn`, `owner` is already of `address` type. In `Perpetual.sol` `constructor`, `_vBase` and `_market` are already of type `IVBase` and `ICryptoSwap` respectively. In `VBase.sol constructor, decimals` can be called directly on `_aggregator` without explicit conversion to `address` and then back to `AggregatorV3Interface`.
c. **Github Permalinks**: mint, burn, Perpetual.sol#L106, VBase.sol#L23
d. **Mitigation**: Remove unnecessary type conversions to improve readability.

16. Unnecessary conditional operator

a. **Summary**: In function `extendPosition` of `Perpetual.sol`, the statement `bool isLong = direction == LibPerpetual.Side.Long ? true : false` may be replaced by the equivalent statement `bool isLong = direction == LibPerpetual.Side.Long`.
b. **Details**: Expression of type `x == condition ? true : false` has the same effect as `x == condition`. The conditional operator is not needed.
c. **Github Permalinks:** Perpetual.sol#L164
d. **Mitigation**: Replace with `bool isLong = direction == LibPerpetual.Side.Long`.

17. Inconsistency in usage of `wadMul`

a. **Summary**: In `Perpetual.sol`, `wadMul` is used to perform decimal multiplication except in one place.
b. **Details**: `_checkPriceDeviation` performs a multiplication without using `wadMul` in `(MAX_PRICE_DEVIATION * currentPrice > (currentPrice - startBlockPrice).abs() * 1e18)`.
c. **Github Permalinks**: _checkPriceDeviation

d. **Mitigation**: Unless there is a reason for exception, it is best to use `wadMul` to be consistent with decimal multiplication elsewhere in the contract.

18. Undocumented privileged functions

a. **Summary**: The contract owner address is capable of performing additional privileged functions in addition to those documented.

b. **Details**: The governance page in documentation indicates that the governance address is capable of performing these four privileged functions: Pausing/unpausing, adding new pairs, removing excess insurance funds and selling CryptoSwap dust. However, there are several additional functions marked `onlyOwner` that constitute privileged functionality. `Vault.sol`: Set a new clearing house contract, potentially to a malicious contract that could drain the vault through `withdraw` which is marked `onlyClearingHouse`, set a new insurance contract and set a new max TVL.`Insurance.sol`: Withdraw all tokens in the Insurance contract.

c. **Github Permalinks**: `Vault.setClearingHouse`, `Vault.setInsurance`, `Vault.setMaxTVL`, `Insurance.withdrawRemainder`

d. **Mitigation**: These permissions should either be documented for transparency or removed if deemed unnecessary. As it currently stands, the governance address could remove all funds from the vault or insurance contract. If the permissions are kept, consider including a timelock before the changes are executed. This would allow users time to remove their funds from the vault before a major modification is made, or in the event of a governance attack/key compromise.

19. Missing zero-address check in ERC20 functions

a. **Summary**: `BaseERC20` implementation does not include zero-address check.

b. **Details**: `BaseERC20` is documented as a more modern and gas efficient version of the OpenZeppelin library as modified from the Solmate version. One of the items removed appears to be the zero-address check in functions.

c. **Github Permalinks**: `approve`, `transfer`, `transferFrom`, `_mint`, `_burn`

d. **Mitigation**: Consider including zero-address checks (where necessary) on user functions that may otherwise cause tokens to be burned.

20. Missing `sweep` function could be useful

a. **Summary:** ERC20 tokens, other than protocol-specific ones, may become locked in contracts.

b. **Details**: Tokens sent by mistake to `Vault` or `ClearingHouse` can be locked without a mechanism to recover them.

c. **Github Permalinks**: `Vault`, `ClearingHouse`

d. **Mitigation**: Consider adding a `sweep` function allowing governance to withdraw tokens other than `reserveToken` from these contracts.

21. Gas optimisation by avoiding initialisation to default values

a. **Summary:** Avoiding initialisation of variables to default values of their types will save gas.

b. **Details**: Integers in Solidity are initialised to zero by default. Therefore, there is no need to again initialise the integer variable to zero after declaration.

c. **Github Permalinks:** `fundingPayments`, `fundingPayments`
d. **Mitigation:** Avoid initialisation of variables to default values of their types

22. Gas optimisation by avoiding strings larger than bytes32

a. **Summary**: Revert strings larger than bytes32 will cost more gas
b. **Details**: EVM is a stack machine with 256-bits (32-bytes) for each stack slot. Storing data larger than 32 bytes requires accessing more stack slots and therefore uses more gas.
c. **Github Permalinks**: `_reducePosition`, `liquidate`
d. **Mitigation**: Use a string less than 32 bytes or consider using custom errors.

23. Gas optimisation by using memory variables

a. **Summary**: Gas optimizations can be achieved by using parameters from memory and caching repeatedly accessed state variables or return values of external function calls in local memory variables.
b. **Details**:
   i. Parameters from memory can be used for `_vBase` and `_vQuote` instead of reading their corresponding state variables assigned earlier from the same parameters.
   ii. Function `withdraw` of contract `Vault` makes two identical calls to `withdrawToken.balanceOf(address(this))` where the return value is the same for the two calls. Caching the return value in memory can save gas.
c. **Github Permalinks**:
   i. `Perpetual.sol#L102-L103`
   ii. `Vault.sol#L181-L182`
d. **Mitigation**: Caching in memory can save gas.

CARE: Disclaimer

CARE is *not* an audit as is typically performed by smart contract security audit firms. CARE participants aim primarily to identify commonly known security pitfalls & best-practices but *not* necessarily application-specific or economic vulnerabilities which are expected to be the focus of future security audits. CARE assumes (as notified and agreed upon earlier in the CARE SoW) that the project will get one or more security audits *after* CARE to cover those aspects. Furthermore, CARE is a best-effort endeavour. Secureum will not be held responsible for any loss/lock of funds/services resulting from vulnerabilities/exploits in projects after they have gone through CARE review.