Secureum CARE Report: **Primitive**

December 2021

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's RACEs (i.e. security challenges/quizzes).

CARE: When, Who & What

Dates: December 6th - December 17th, 2021

CAREtakers (Discord handles): patitonar#3132, cakebeef#3150, fyang1024#4857, Jonah1005#6074, ejolanix#0978, Exef#5221, Icar#2085, amansız_papaya#7268, nikitastupin#0422, stankleg#6900, mtz#5107, PranavGarg#5122, mauricio97#6512, bohendo#4221, SaharAP#0803, popular#6969, 구자현#7593, Fatemeh#8729 & Rajeev | Secureum#7615

Scope: Review focused on these three repositories/commits of Primitive Finance protocol:

1. RMM Core protocol : https://github.com/primitivefinance/rmm-core/tree/35820b518d9b452c84e415a84232e4de4299ef13
2. RMM Manager: https://github.com/primitivefinance/rmm-manager/tree/6bc8c436ca7c42a248073e46b1d82db61fa291ec

3. RMM Examples: https://github.com/primitivefinance/rmm-examples/tree/7048fc9efdbefd780b11def672b6233a4b57ec19

Note: Commit hashes used are as of December 6th. 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: Security Pitfalls & Best-Practices Checklist

1. **Lack of (specification/documentation of) access control on `CashManager` functions**
    a. **Summary**: Functions of asset management contract (`CashManager.sol`) are callable by arbitrary users.
    b. **Details**: The asset management functions (`wrap, unwrap, sweepToken, refundETH`) all have features directly transferring assets managed by the contract. These functions have no access control logic. Anyone can call these functions as they notice unintended tokens or ETH in the contract.
    c. **Github Permalinks**: wrap, unwrap, sweepToken, refundETH
    d. **Mitigation**: Although the `rmm-manager` has no documented specification of possessing ETH or tokens, users can mistakenly transfer their assets to the `rmm-manager`. These functions can be used to rescue the transferred assets but it is not clear if and how any access control is required or expected to be implemented by derived contracts. Specify and document the design and expected behaviour and usage.
2. **Unused `Margin.sol withdraw` return value**
    a. **Summary**: `withdraw` returns "`Data storage margin`" which is never used.
    b. **Details**: The `Margin` library function `withdraw` is used in `PrimitiveEngine.withdraw`, `PrimitiveEngine.allocate`, `PrimitiveEngine.swap`. However, the return value of `Margin.withdraw` function is never used and is not required in the callers' logic.
    c. **Github Permalinks**: `Margin.withdraw`
    d. **Mitigation**: Remove return value/statement of `Margin.withdraw` function.
3. **State variables can be declared constant to save gas**
    a. **Summary**: `PositionManager.sol _empty` variable and `data`, `_data` variables in `LiquidityWrapper` and `PrimitiveChef` can be declared constant.
    b. **Details**: `bytes private` variables `_empty`, `data` and `_data` can be declared as empty string constants. This would save gas by avoiding the expensive SLOAD instruction.
    c. **Github Permalinks**: `PositionManager.sol`, `LiquidityWrapper.sol`, `PrimitiveChef.sol`

d. **Mitigation**: Use `bytes constant` for variables `_empty`, `data` and `_data`

4. **Unused imports**
   a. **Summary**: There are unused import statements in the `PositionRenderer` contract.
   b. **Details**: There are 3 unused import statements of `IPrimitiveEngineView.sol`, `IERC20WithMetadata.sol` and `HexStrings.sol` in `PositionRenderer` contract. The import of `IPrimitiveManager` in `ManagerBase` is unnecessary. None of these are required for what's implemented in these contracts.
   c. **Github Permalinks**: `PositionRenderer.sol`, `ManagerBase.sol`
   d. **Mitigation**: Remove unused import statements.
5. **Unused parameters**
   a. **Summary**: There are unused parameters in the `PositionRenderer` contract.
   b. **Details**: The `PositionRenderer.render` function does not use its two parameters `engine` and `tokenId`. The `render` function returns an SVG representation of the position tokens. It takes two variables `engine` and `tokenId` but neither are used to construct the SVG.
   c. **Github Permalinks**: `PositionRenderer.sol`
   d. **Mitigation**: Remove unused function parameters or implement missing functionality to integrate unused variables to compose SVG.
6. **Documentation inconsistency**
   a. **Summary**: NatSpec documentation of `ReplicationMath` has differences between `rmm-core` and `rmm-math` repositories.
   b. **Details**: The `rmm-math` repository contains Typescript implementation of mathematics which is used in replication market maker logic in Solidity contracts of `rmm-core`. There are differences between their documentations which could lead to confusion. For example:
      i. `getStableGivenRisky` (`rmm-core`): "*Uses riskyPerLiquidity and invariant to calculate stablePerLiquidity*" versus `getStableGivenRisky` (`rmm-math`): "*Forward trading function to calculate the risky reserve given a stable reserve which has a 0 invariant*"
      ii. *`calcInvariant`* (`rmm-core`): "*Calculates the invariant of a curve*" versus `calcInvariant` (`rmm-math`): "*reserveRisky Pool's reserve of risky tokens per unit of liquidity*"
   c. **Github Permalinks**: `ReplicationMath.ts (rmm-math)`, `ReplicationMath.sol (rmm-core)`
   d. **Mitigation**: Make documentation consistent.
7. **Unlocked and Multiple pragmas**
   a. **Summary**: Multiple versions of `solc` pragma, some of which are unlocked/floating, are used across different contracts.
   b. **Details**: Several contracts use different/floating solc pragma: '`>=0.5.0`', `>=0.6.0`, '`>=0.8.0`', '`^0.8.0`', '`^0.8.4`', and '`>=0.8.6`'

c. **Github Permalinks**: SafeCast.sol, Transfers.sol, IPrimitiveEngine.sol, ABDKMath64x64.sol, ReplicationMath.sol, ICashManager.sol
d. **Mitigation**: Contracts should ideally be deployed using the same compiler version with which they have been tested. Locking the pragma ensures that contracts do not accidentally get deployed using an older compiler version with unfixed bugs or newer than the version tested with. Libraries that need to work with contracts across different versions could be an exception to this best-practice. Evaluate choice of pragma based on these considerations.

8. **Naming convention**
   a. **Summary**: The immutable variables `factory_` and `positionRenderer_` should be written in uppercase.
   b. **Details**: According to the Solidity Style Guide constants should be named with all capital letters with underscores separating words. Immutables can be treated similar to constants because they are replaced by their assigned values during contract construction.
   c. **Github Permalinks**: PrimitiveManager, ManagerBase
   d. **Mitigation**: Use uppercase to rename these variables as `FACTORY_` and `POSITION_RENDERER_`.
9. **Different variable ordering affects readability**
   a. **Summary**: Variables are in different order in the `callbackData` function compared to the `callbackData` struct definition.
   b. **Details**: The function `callbackData` in `PrimitiveManager.sol` orders the variables `risky`, `stable` and `payer` differently from the `callbackData` struct definition in `ManagerBase.sol` which orders them as `payer`, `risky` and `stable`.
   c. **Github Permalinks**: PrimitiveManager, ManagerBase
   d. **Mitigation**: To improve readability, consider using the variables in the same order in both places.
10. **State variable declaration ordering and necessity**
    a. **Summary**: State variable `_engine` is declared out of order and appears unnecessary.
    b. **Details**: State variable `_engine` is declared after functions which is not consistent with Solidity Style Guide best practice. State variables should be declared before functions according to the Solidity Style Guide.
    Also, this could simply be a local memory variable. The `allocate` function uses `_engine` state variable, by computing its value every time the function is called and sets that to zero-address before returning. The value is passed to every function called as an argument and is not read in any other place. This state variable declaration and use is an unnecessary waste of gas and may be replaced by a local variable as done in other places.
    c. **Github Permalinks**: _engine
    d. **Mitigation**: Consider declaring `_engine` above the function declarations and also consider replacing it with a local variable.
11. **Non-descriptive variable name**
    a. **Summary**: `cache` is a non-descriptive variable name.

b. **Details**: The `cache` variable is meant to map pool `id`'s with engine addresses. However, `cache` is not a descriptive variable name.
c. **Github Permalinks**: `cache`
d. **Mitigation**: Consider renaming `cache` to a more descriptive variable name.

12. **Maximum line length exceeded**
    a. **Summary**: Long comments should be wrapped to conform with Solidity Style guidelines.
    b. **Details**: Lines 83, 84 & 86 in `IPrimitiveManager.sol` exceed the 79 (or 99) character length suggested by the Solidity Style guidelines.
    c. **Github Permalinks**: Comments
    d. **Mitigation**: Lines 83, 84 & 86 should be wrapped to a maximum of 79 (or 99) characters to help readers easily parse the comments.
13. **Missing Natspec**
    a. **Summary**: Missing Natspec comments affect readability and maintainability of a codebase.
    b. **Details**:
        i. There are three variables in `IPrimitiveManager` missing NatSpec comments: parameters `minLiquidityOut` of `allocate` function and `minRiskyOut`, `minStableOut` of `remove` function are not included in its Natspec documentation.
        ii. `IERC20.sol` & `IERC20WithMetadata.sol` are missing Natspec comments.
        iii. `RewardToken.sol`
    c. **Github Permalinks**: `IPrimitiveManager.sol`, `IERC20`, `RewardToken`
    d. **Mitigation**:
        i. Add `@param` descriptors for `minLiquidityOut`, `minRiskyOut` and `minStableOut`.
        ii. Add Natspec comments.
        iii. Add Natspec comments.
14. **Documentation errors**
    a. **Summary**: There are discrepancies between the documentation and implementation of `CashManager.sol`.
    b. **Details**: The `pay` function in `CashManager.sol` is not included in the developer documents. The interface contract `ICashManager.sol` also does not include the `pay` function which may be by design. `CashManager` developer documents include references to `factory` and `positionRenderer`, neither of which are in `CashManager`.
    c. **Github Permalinks**: `CashManager` docs, `ICashManager` docs
    d. **Mitigation**: Consider fixing the documentation for `CashManager` to include `pay`. Remove references to `factory` and `positionRenderer` from `ICashManager.sol` documentation.
15. **Function ordering**
    a. **Summary**: Functions are out of order according to the Solidity Style Guide.
    b. **Details**: Function ordering helps readers identify which functions they can call and find constructor and fallback functions easier. Functions should be grouped according to their visibility and ordered: constructor, receive function

(if exists), fallback function (if exists), external, public, internal, private. A few public and external functions are mixed up in `PrimitiveChef`.

c. **Github Permalinks**: `PrimitiveChef.sol`

d. **Mitigation**: Consider reorganising function layout to adhere to Solidity Style Guide best practices.

16. **Shadowed state variable**

a. **Summary**: Shadowed state variable `owner` is misleading.

b. **Details**: The variable `owner` in the `IERC1155Permit` function call is shadowing the `owner` variable from base contract `Ownable.sol`. This does not appear to cause problems because a valid signature is required by the address `owner` in `selfPermit` and it would not be possible to generate a signature from `Ownable.owner` without controlling the address.

c. **Github Permalinks**: `PrimitiveChef.sol`

d. **Mitigation**: Consider changing the `owner` variable name in the `selfPermit` and `IERC1155Permit` function parameters or explain the shadowed variable with a code comment.

17. **`PrimitiveChef` missing validation for duplicate pools**

a. **Summary**: The `add` function is missing a sanity check for duplicate pools.

b. **Details**: The `add` function allows the contract `owner` to add a new liquidity pool token to the contract. However, there is no validation to see if the pool has already been added which can lead to miscalculated reward allocations. This is noted in the function comment as a warning: "*// XXX DO NOT add the same LP token more than once. Rewards will be messed up if you do.*"

c. **Github Permalinks**: `add`

d. **Mitigation**: Consider adding a require statement to check if the LP token has previously been added to `pools[poolId]`.

18. **Unbounded `for` loop**

a. **Summary**: The function `massUpdatePools` contains an unbounded `for` loop.

b. **Details**: `massUpdatePools` has a `for` loop which updates the reward variables for all pools using `pools.length` as the upper bound of the loop. As more pools are added, it costs more gas to run the `massUpdatePools` function. This could ultimately lead to an out of gas exception which would prevent updating of pool rewards using this function.

c. **Github Permalinks**: `massUpdatePools`

d. **Mitigation**: Consider adding optional bounds variables to `massUpdatePools` to allow running the update function in chunks.

19. **Missing zero-address validation**

a. **Summary**: Missing zero-address validation for address parameters may lead to ETH/tokens getting burned or reverts.

b. **Details**:

i. In `PrimitiveChef.sol`, `setCollector` function lacks zero-address validation for `newCollector`. The constructor lacks zero-address validation for `rewardToken_` and `collector_`.

ii. In `RewardToken.sol`, constructor lacks zero-address validation for `chef`.

iii. In `CashManager.sol`, `unwrap` function lacks zero-address validation for `recipient`.
iv. In `PrimitiveEngine.sol`, `swap` and `allocate` functions lack zero-address validation for `recipient`.
v. In `LiquidityWrapper.sol`, constructor lacks zero-address validation for `manager`.
vi. In `MarginManager.sol`, `deposit` function lacks zero-address validation for `recipient`, `risky`, `stable`.

c. **Github Permalinks**: setCollector, PrimitiveChef.constructor, RewardToken constructor, unwrap, swap, allocate, LiquidityWrapper.constructor, deposit

d. **Mitigation**: Consider adding zero-address validation with a require statement.

20. Functions with unnecessary `payable` specifier may lead to locked ETH

a. **Summary**: Functions that do not need to accept ETH are declared `payable`.

b. **Details**: Functions that do not need to accept ETH are declared `payable` which allows callers to mistakenly send ETH to such contracts and have it locked/lost.

i. `PrimitiveManager create` and `allocate`.
ii. `CashManager unwrap`, `sweepToken` and `refundETH`.
iii. `MarginManager deposit`.
iv. `SelfPermit selfPermit`, `selfPermitIfNecessary`, `selfPermitAllowed` and `selfPermitAllowedIfNecessary`.
v. `SwapManager swap`.

c. **Github Permalinks**: PrimitiveManager create, PrimitiveManager allocate, CashManager unwrap, CashManager sweepToken, CashManager refundETH, MarginManager deposit, SelfPermit selfPermit, SelfPermit selfPermitIfNecessary, SelfPermit selfPermitAllowed, SelfPermit selfPermitAllowedIfNecessary, SwapManager swap

d. **Mitigation**: Remove unnecessary `payable` modifier from functions that do not need to accept ETH.

21. Unnecessary comparison with boolean constant

a. **Summary**: Comparison with boolean constant is not required.

b. **Details**: Checking for `true` or `false` in conditional expressions can simply use the boolean variable with/without the '!' operator instead of explicitly comparing with boolean constants `true` or `false`.

i. `PrimitiveManager create` and `allocate`
ii. `MarginManager deposit`
iii. `SwapManager swap`
iv. `TransferHelper safeTransferETH`

c. **Github Permalinks**: PrimitiveManager create, PrimitiveManager allocate, MarginManager deposit, SwapManager swap, TransferHelper safeTransferETH

d. **Mitigation**: Remove the boolean constant or replace it with '!' on the boolean variable as necessary. For example, replace

`EngineAddress.isContract(engine) == false` with `!EngineAddress.isContract(engine)`

22. **Function mutability is not strict enough**
    a. **Summary**: `PositionRenderer` contract has a `view` function which can be `pure`.
    b. **Details**: The `render` function mutability can be restricted to pure since it does not read state variables.
    c. **Github Permalinks**: `PositionRenderer render`
    d. **Mitigation**: Replace `view` with `pure`.
23. **ERC20 `transfer` return values are ignored**
    a. **Summary**: Functions do not check return value of `transfer` function call on ERC20 tokens.
    b. **Details**: The `unwrap` and `pay` functions in `CashManager` do not check the return value of `transfer` function call on WETH token. The `safeRewardTransfer` function does not check the return value of `transfer` function call on the `RewardToken`.
    c. **Github Permalinks**: `unwrap`, `pay`, `safeRewardTransfer`
    d. **Mitigation**: Use `require` on the return value or use `TransferHelper.safeTransfer`.
24. **`block.timestamp` used as time proxy**
    a. **Summary**: Risk of using `block.timestamp` for time should be considered.
    b. **Details**: `block.timestamp` is not an ideal proxy for time because of issues with synchronisation, miner manipulation and changing block times.
    c. **Github Permalinks**: `ERC1155Permit permit`, `PrimitiveEngine.sol`, `SwapManager.sol`
    d. **Mitigation**: Consider the risk of using `block.timestamp` as time proxy and evaluate if block numbers can be used as an approximation for the application logic. Both have risks that need to be factored in.
25. **Missing events for critical functions**
    a. **Summary**: Critical functions do not emit events.
    b. **Details**: Events for critical state changes such as Ether transfer should be emitted for tracking off-chain. `CashManager unwrap` and `refundETH` functions do not emit events. `PrimitiveChef` does not emit an event when the variable `collector` is updated by the `setCollector` function.
    c. **Github Permalinks**: `unwrap`, `refundETH`, `setCollector`
    d. **Mitigation**: Emit appropriate events with necessary parameters.
26. **Special handling of zero-address is risky**
    a. **Summary**: Special handling of zero-address in `MarginManager.sol` is risky.
    b. **Details**: Zero-address is typically used as the burn address by most contracts and is enforced with zero-address checks. However, `MarginManager.withdraw` handles zero-address differently as noted in the comment: "*Setting address(0) as the recipient will result in the tokens being sent into the contract itself, useful to unwrap WETH for example.*" This is a risky convention for users interacting with this function compared to other functions of the protocol or other protocols.

A user might mistakenly withdraw their funds to the `PrimitiveManager` by setting the recipient to zero-address, immediately following which a malicious user monitoring the `PrimitiveManager` contract withdraws those funds that were left in the contract.

c. **Github Permalinks**: withdraw
d. **Mitigation**: Reconsider convention and evaluate risk. Consider using a boolean instead to indicate the caller's intention to send to the contract address. Document the expected usage and behaviour.

27. Cloning of widely-used standard libraries/interfaces/contracts is risky

a. **Summary**: Cloning of widely-used standard libraries/interfaces/contracts is risky.
b. **Details**: Copy-pasting code from other libraries, contracts or even different parts of the same contract may result in incorrect code semantics for the context being copied to, copy over any vulnerabilities or miss any security fixes applied to the original code. All these may lead to security issues.
    i. `rmm-core`
        1. `IERC20`: Consider reusing OpenZeppelin IERC20
        2. `ABDKMath64x64`: Consider reusing npm package abdk-libraries-solidity for (GitHub source)
        3. `SafeCast`: Consider reusing OpenZeppelin SafeCast
        4. `Transfers`: Consider reusing OpenZeppelin SafeERC20
    ii. `rmm-manager`
        1. `ERC1155`: Consider reusing OpenZeppelin ERC1155
        2. `Multicall`: Consider reusing OpenZeppelin Multicall
        3. `Reentrancy`: Consider reusing OpenZeppelin ReentrancyGuard
        4. `SelfPermit`: Consider reusing @uniswap/v3-periphery npm package for GitHub source
        5. `IERC20`: Consider reusing OpenZeppelin IERC20
        6. `IERC20WithMetadata`: Consider reusing OpenZeppelin IERC20Metadata
        7. `TransferHelper`: Consider reusing OpenZeppelin SafeERC20 to replace `IERC20` functions and OpenZeppelin Address to replace `safeTransferETH`
        8. `HexStrings`: Consider reusing OpenZeppelin Strings
        9. `ISelfPermit`: Consider reusing Uniswap V3 Periphery
        10. `rmm-manager/contracts/libraries/Margin.sol` mostly replicates `rmm-core/contracts/libraries/Margin.sol`, except
            a. Latter's `withdraw` function has a `mapping(address => Data) storage margins` as the first parameter while the former has a `Data storage margin`, and
            b. Latter returns a `(Data storage margin)` while the former returns nothing

c. **Github Permalinks**: IERC20.sol, ABDKMath64x64, SafeCast.sol, Transfers, ERC1155.sol, Multicall.sol, Reentrancy.sol, SelfPermit.sol, IERC20WithMetadata.sol, ISelfPermit.sol, HexStrings.sol, Margin.sol, TransferHelper.sol

d. **Mitigation**: Copying and trimming libraries is likely done to include only required library functions, reduce dependencies, lower deployment cost and reduce attack surface. Reevaluate these benefits of cloning/trimming/modifications against risks from missed security fixes and security issues introduced from code/context changes.

28. Incorrect event argument

a. **Summary**: The `MarginManager` `withdraw` function allows a user to withdraw their funds but the function has special handling for zero-address. However the `Withdraw` event does not follow the same handling rules.

b. **Details**: In `MarginManager` `withdraw`, if zero-address supplied to the `withdraw` function then the funds will be withdrawn to the `PrimitiveManager` contract. However, the `Withdraw` event emitted indicates `msg.sender` as the recipient of withdrawal which is incorrect.

c. **Github Permalinks**: withdraw

d. **Mitigation**: Consider correcting the argument of the `Withdraw` event from `msg.sender` to `address(this)`.

29. `PrimitiveChef` can be drained via reentrancy

a. **Summary:** An attacker can exploit the `emergencyWithdraw` function to drain Liquidity Provider (LP) tokens deposited into the contract by other users via an external contract call in ERC1155's `safeTransferFrom` function.

b. **Details:** By modifying `MasterChef` to treat LP tokens as ERC1155s, a vulnerability was introduced that does not exist in the original `SushiSwap` contract. Sushi's `MasterChef` treats `lpToken` as an ERC20, which does not have an external call in its `safeTransfer`.

i. The caller's balance is not updated until after the external call, resulting in a exploitable reentrancy scenario. An attacker can repeatedly call `emergencyWithdraw` from a custom `onERC721Received` and remove LP tokens equal to `user.amount` on each call.

c. **Github Permalinks:** emergencyWithdraw

d. **Mitigation:** Consider moving the `safeTransfer` call to the end of the function to follow the Checks-Effects-Interactions (CEI) pattern. Consider using a reentrancy guard.

30. 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 function is never called from the contract then it should be marked as `external`. This will save gas.

c. **Github Permalinks**: uri, ERC1155.safeTransferFrom, ERC1155.setApprovalForAll, ERC1155.safeBatchTransferFrom, ERC1155.balanceOfBatch

d. **Mitigation**: Change visibility from `public` to `external`.

31. Test coverage report

a. **Summary**: Test coverage reports are missing.
b. **Details**: `rmm-core` and `rmm-manager` repositories do not appear to have a tests coverage report.
c. **Github Permalinks**: None.
d. **Mitigation**: Add tests coverage reports for both repositories.

32. Lack of contract existence check for low-level calls

a. **Summary**: Low-level calls `call/delegatecall/staticcall` return true even if the account called is non-existent. Account existence must be checked prior to calling if needed.
b. **Details:** `TransferHelper` functions make a low-level `call` on arbitrary tokens without checking for token contract existence.
c. **Github Permalinks:** `TransferHelper.safeTransfer`, `TransferHelper.safeTransferFrom`, `TransferHelper.safeApprove`
d. **Mitigation:** Check for contract existence in `TransferHelper` functions.

33. `payable Multicall` is dangerous

a. **Summary**: `payable Multicall` may lead to exploits.
b. **Details**: `payable Multicall` can lead to exploits if any of the callable functions use `msg.value` in their logic. For references, see https://github.com/Uniswap/v3-periphery/issues/52 and https://www.paradigm.xyz/2021/08/two-rights-might-make-a-wrong/
c. **Github Permalinks**: `Multicall`
d. **Mitigation**: The current functions do not appear to use `msg.value` but evaluate any future modifications for such usage in the context of `Multicall`.

34. Inefficient `lock` modifier

a. **Summary**: The reentrancy protection `lock` modifier is not gas efficient.
b. **Details**: The `lock` modifier is switching the `unlocked` state variable between values of 0 and 1 while protecting against reentrancy. There is an increased storage gas cost while toggling between 0 and non-zero values which is paid every time a reentrancy-protected function is called.
c. **Github Permalinks:** `Reentrancy.sol`
d. **Mitigation**: Consider using an OpenZeppelin-like implementation which toggles between values of 1 and 2 which should maximise the gas refund.

35. Non-optimal implementation of `lock` in `rmm-core`

a. **Summary**: `unlocked` is a `uint8` which leads to storage slot packing with the previously declared address variable but consumes more gas for masking during SLOAD/SSTORE.
b. **Details**: `uint8` requires more gas than `uint256` because the EVM must read the contents of the 256-bit slot and mask the rest of the slot contents before reading/writing that value.
c. **Github Permalinks**: `unlocked`, `lock`
d. **Mitigation**: Use `uint256` instead of `uint8`.

36. Duplicate `lock` modifier

a. **Summary:** The reentrancy protection function modifier is implemented twice.
b. **Details**: The `lock` modifier is implemented in both `Reentrancy.sol` of `rmm-manager` and `PrimitiveEngine.sol` of `rmm-core`.
c. **Github Permalinks:** `Reentrancy`, `PrimitiveEngine`
d. **Mitigation**: Consider using the OpenZeppelin implementation of `ReentrancyGuard` in both places. Alternatively, consider implementing this modifier once & importing it in both locations. This would prevent a bug fix to one implementation from not being applied to the other.

37. Inconsistent function return patterns affects readability

a. **Summary**: Some functions explicitly return a value while others assign a value to a named return variable.
b. **Details**: If many functions explicitly return a value, one may incorrectly infer that functions assigning a value to a named return variable are not actually returning anything.
c. **Github Permalinks**: `balanceRisky` versus `updateLastTimestamp`
d. **Mitigation**: Consider refactoring functions to standardise on one return pattern, preferably explicit returns.

38. Documentation typos

a. **Summary**: Multiple instances of typos in documentation.
b. **Details**: Multiple instances of typos in documentation which may mislead developers and users.
c. **Github Permalinks**:
   i. Change $$ x\times y=k $$ to ```x*y=k```
   ii. Change “scaling” to “scale”
   iii. Change RMm to RMM
   iv. Change “three” to “four”
   v. Change “[Periphery]” to “[manager]”
   vi. Change “PrimitiveHouse” to “PrimitiveManager”
   vii. Change “{token]” to “{token}”
d. **Mitigation**: Make changes suggested above.

39. **Redundant checks**

a. **Summary**: Redundant zero value checks on `delRisky`, `delStable` and `delLiquidity`.
b. **Details**: The zero value check on `delRisky`, `delStable` and `delLiquidity` values in `deposit`, `withdraw`, `allocate` and `remove` functions of `PrimitiveEngine` is duplicated in `MarginManager` and `PrimitiveManager`. Removing redundant checks will not only reduce gas costs but improve readability and maintainability of the code.
c. **Github Permalinks**: `MarginManager`, `PrimitiveManager`, `PrimitiveEngine.deposit`, `PrimitiveEngine.remove`
d. **Mitigation**: Consider retaining the checks in `MarginManager` and `PrimitiveManager` while removing the checks in `PrimitiveEngine` so that errors are checked closer to the user interface and also saves gas from a reverting external call (when check fails).

40. **Numerical miscalculations due to operator precedence**

a. **Summary**: Incorrect numerical computation due to operator precedence when converting from `SafeMath` to regular arithmetic operators.
b. **Details**: There are numerical errors introduced during the conversion of `MasterChef` to `PrimitiveChef` and the switching of compiler/`SafeMath`. Missing parentheses to force required ordering of operations leads to incorrect order of operations due to implicit associativity of operators. While this worked in `MasterChef` (`solc 0.6.12`) due to LtoR calling of `SafeMath` functions, the current logic miscalculates and/or reverts. The logic to calculate multipliers in the `getMultiplier` function is incorrect. The subtraction should happen before the multiplication with the `BONUS_MULTIPLIER`.
c. **Github Permalinks**: `getMultiplier#L241`, `getMultiplier#L243`
d. **Mitigation**: Use parentheses to order operations in the same precedence as the `SafeMath` operation ordering in `MasterChef`.

41. Variables can be declared `immutable`

a. **Summary**: Declaring variables `immutable` can save gas.
b. **Details**: Declaring variables that do not change after deployment as `immutable` can save gas.
c. **Github Permalinks**: `PrimitiveChef`: `rewardToken`, `bonusEndBlock`, `rewardPerBlock`, `startBlock` `RewardToken`: `chef` `LiquidityWrapper`: `manager`, `poolId`
d. **Mitigation**: Use `immutable` keyword when declaring these state variables.

42. Save gas with `unchecked` subtraction

a. **Summary:** Gas savings on `unchecked` subtraction with bounded variables.
b. **Details:** If it is guaranteed that `cal.maturity > cal.lastTimestamp` it is impossible for the subtraction `cal.maturity - cal.lastTimestamp` to underflow and therefore can safely use `unchecked{}` block to save on gas by avoiding Solidity's intrinsic bound checks.
c. **Github Permalinks:** `PrimitiveEngine.create`, `PrimitiveEngine.invariantOf`
d. Mitigation: Wrap `uint32 tau = cal.maturity - cal.lastTimestamp;` with `unchecked{}` statement.

43. Redundant variable initialization

a. **Summary**: Initialization for variable `totalAllocPoint` is not needed.
b. **Details**: The default value of `uint256` is 0 thus the initialization to 0 is not needed and will save gas.
c. **Github Permalinks**: `PrimitiveChef`
d. **Mitigation**: Remove the initialization.

44. `delete` statement is preferred for removing a stored value

a. **Summary**: `delete` statement is preferred for removing a stored value than assigning it to 0. This accurately illustrates developer intent.
b. **Details**: `_engine` value is set to `address(0)` instead of using the `delete` statement in `PrimitiveManager`. Variables `user.amount` and `user.rewardDebt` are set to 0 instead of using the `delete` statement in `PrimitiveManager`.

c. **Github Permalinks**: `_engine`, `user.amount` and `user.rewardDebt`
d. **Mitigation**: Use `delete` instead of setting the values to 0.

45. **Gas optimization by not initialising loop indices**
    a. **Summary**: Loop indices need not be initialised to 0.
    b. **Details**: Loop indices are 0 valued by default and there is no need to explicitly initialise them again to 0. This saves gas.
    c. **Github Permalinks**: `Multicall`, `PrimitiveChef`, `ERC1155#L92`, `ERC1155#L302`, `ERC1155#L358`
    d. **Mitigation**: Remove explicit initialization to 0 for loop indices.
46. **Gas optimization by using or caching in local variables**
    a. **Summary**: Expensive SLOAD/SSTORE can be avoided by using or caching in local variables.
    b. **Details**: Instead of using state/storage variables, one can use or cache values in local variables to avoid expensive SLOAD/SSTORE instructions.
        i. `PrimitiveEngine.allocate`: Using `memory` instead of `storage` data location specifier will avoid SLOADs in reads on L251 & L252 and the state update on L257 can simply use `reserves[poolId]` directly.
        ii. `PrimitiveEngine.swap`: Using `memory` instead of `storage` here will avoid SLOADs in reads on L340-L347 and the state update on L360 can simply use `reserves[poolId]` directly.
    c. **Github Permalinks**: `PrimitiveEngine.allocate`, `PrimitiveEngine.swap`
    d. **Mitigation**: Replace use of state/storage variables with (cached) local variables.
47. **Missing owner-operator address equality check**
    a. **Summary:** Missing check on `owner != operator`.
    b. **Details:** As stated in EIP-1155: "*An owner SHOULD be assumed to always be able to operate on their own tokens regardless of approval status, so SHOULD NOT have to call setApprovalForAll to approve themselves as an operator before they can operate on them.*"
    `EIP1155.sol` appears to have been modified from the original OpenZeppelin implementation where the `owner != operator` check has been moved out of `_setApprovalForAll` function into the `setApprovalForAll` function. This misses the address equality check in the `ERC1155Permit` `permit` flow.
    c. **Github Permalinks**: `ERC1155._setApprovalForAll`, `ERC1155.setApprovalForAll`, `ERC1155Permit.permit`
    d. **Mitigation**: Move check on `owner != operator` to `_setApprovalForAll` function
48. **Single-step ownership change is risky**
    a. **Summary: Single-step ownership change in Ownable is risky**
    b. **Details:** When privileged roles are being changed, it is recommended to follow a two-step approach: 1) The current privileged role proposes a new address for the change 2) The newly proposed address then claims the privileged role in a separate transaction. This two-step change allows

accidental proposals to be corrected instead of leaving the system operationally with no/malicious privileged role. For e.g., in a single-step change, if the current admin accidentally changes the new admin to a zero-address or an incorrect address (where the private keys are not available), the system is left without an operational admin and will have to be redeployed.
`PrimitiveChef` derives from `Ownable` which implements a single-step ownership change.

c. **Github Permalinks**: `PrimitiveChef`
d. **Mitigation**: Single-step ownership change by `Ownable` can be overridden to a two-step change.

49. Address validity check missing

a. **Summary**: Check to validate engine address is missing.
b. **Details**: `MarginManager.withdraw` is missing an input validation check on the `engine` parameter to check that it is indeed a valid `EngineAddress`.
c. **Github Permalinks**: `MarginManager.withdraw`
d. **Mitigation**: Add check to validate `engine` address.

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.