Secureum CARE Report: **Sushi**

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): rootulp#9326, Shana#2321, palina#6282, SWAT_1598#6083, wh01s7#7779, balag3#2465, n4motto#4531, yosriady#8285, patrickd#5542, QZ#6600, calvbore#4953, Dayitva Goel#5802, Apulina#6835, Razzor#0567, lukasz.glen#6296, overmn#3396, ctzurcanu+1#8710, reassor#7970, nahsiz#7945, fellu#1690, grmpyninja#9235 & Rajeev | Secureum#761

Scope: Review focused on these repositories/commits of Sushi protocol:

1. Repository/Commit: https://github.com/sushiswap/bentobox-strategies/tree/be407e27cd1a9ca8060ef9a389d1cf01b4e5540e
2. Contracts in scope: `BaseStrategy.sol` and `AaveStrategy.sol`

Note: Commit hash used is 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. **Single-step ownership change is risky**
    a. **Summary:** Single-step ownership change for contracts 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.
    `BaseStrategy` inherits from `Ownable`. If the new owner is set to a wrong address, this could lead to loss of funds.
    c. **Github Permalinks**: `BaseStrategy`
    d. **Mitigation**: Consider replacing single-step ownership change by a two-step change which involves separate propose/grant+claim steps. Also consider overriding the `renounceOwnership()` function to prevent it from ever being called.
2. **Unlocked and Multiple pragmas**
    a. **Summary**: Multiple versions of `solc` pragma, one of which is unlocked, are used across different contracts.
    b. **Details**: The pragma in `BaseStrategy` uses `pragma solidity >=0.8;` which allows the contract to be compiled with any version 0.8.0 and higher, while AaveStrategy uses `pragma solidity 0.8.7;`
    c. **Github Permalinks**: `BaseStrategy`
    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 or abstract contracts that need to work with contracts across different versions could be an exception to this best-practice. Evaluate the choice of pragmas based on these considerations.
3. **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. Typographical errors in comments.

ii. Comment for `strategyExecutors` says: "*@dev EOAs that can execute safeHarvest.*" but the modifier `onlyExecutor` is applied not only to `safeHarvest` but also to `swapExactTokensForUnderlying`. It is not clear if the comment is wrong or the applied modifier on `swapExactTokensForUnderlying`.

iii. `AaveStrategy` is missing NatSpec comments.

iv. Trusted actors (e.g. `Owner`, `BentoBox`, `Executor`), assumptions on their actions and assets are not specified in documentation, which leads one to assume that compromised privileged actors are outside the threat model. For example, it is not clear if anyone being able to call `skim` is intentional and by design, or an access control oversight.

v. The overall system specification and documentation is lacking sufficient details for external reviewers/users to understand terminology and flows.

c. **Github Permalinks**:

i. Remove "rewards", "necessary" to "necessarily", Abstrat to Abstract, loner to longer, Calcualte to Calculate, continiue to continue, seperately to separately

ii. Comment, modifier, `swapExactTokensForUnderlying`

iii. `AaveStrategy`

iv. `skim`

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

4. **Missing zero-address validation**

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

b. **Details**: The following address parameters are missing zero-address checks:

i. `to` in `afterExit`

ii. `aaveLendingPool` and `incentiveController` in `AaveStrategy.constructor`

iii. `strategyToken`, `bentoBox`, `factory` and `_allowedSwapPaths` items in `BaseStrategy.constructor`

iv. `executor` in `setStrategyExecutor`

v. `path` addresses in `setAllowedPath`

c. **Github Permalinks**: `afterExit`, `AaveStrategy.constructor`, `BaseStrategy.constructor`, `setStrategyExecutor`, `setAllowedPath`

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

or custom errors.

5. **Return value of functions ignored**
    a. **Summary**: The return values of functions need to be checked for the appropriate values/conditions to determine error conditions appropriately.
    b. **Details**:
        i. `afterExit` in `BaseStrategy` transfers the specified amount to the address `to` using a low-level `call`. However, the return value, i.e., success is not checked with the expectation that the caller will check this return value of `afterExit`. If `call` fails, funds can be locked in the contract.
        ii. `ILendingPool.withdraw` returns `uint256`. However, this return value is not used when this is called in `_harvest`, `_withdraw` or `_exit` in `AaveStrategy`.
        iii. `incentiveController.claimRewards` returns `uint256` but this is ignored in `_harvestRewards`.
    c. **Github Permalinks**: `afterExit`, `withdraw`, `claimRewards`
    d. **Mitigation**: Consider checking return values of the functions to detect error codes/conditions and verify expected values.
6. **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**: `afterExit`, `swapExactTokensForUnderlying`
    d. **Mitigation**: Change visibility from `public` to `external`.
7. **Public variable visibility can be made private**
    a. **Summary**: Variables should have `private` visibility to prevent deriving contracts from accidentally modifying them.
    b. **Details**: If a `public` variable is never meant to be changed by a deriving contract, its visibility should be made `private`.
        i. `exited` variable in `BaseStrategy` is set by `exit` whose access is controlled by `onlyBentoBox`. However, its visibility is `public` which allows deriving contracts to accidentally change its value in functions without access control thus leading to either reactivating an exited contract or deactivating an active one.
        ii. `maxBentoBoxBalance` defines the maximum balance of the underlying token that is allowed to be in `BentoBox`. A deriving contract might accidentally modify the `maxBentoBoxBalance` variable resulting in an unintended ratio of token allocation.
        iii. `strategyExecutors` mapping holds addresses that are allowed to execute the `safeHarvest` function. A deriving contract might

accidentally modify `strategyExecutors` mapping resulting in either removal of valid executors or addition of unauthorised/invalid ones.

c. **Github Permalinks**: `exited`, `maxBentoBoxBalance`, `strategyExecutors`

d. **Mitigation**: Change visibility from `public` to `private` and provide explicit getters where necessary.

8. **`skim` function is always callable**

a. **Summary**: `skim` function is missing `isActive` modifier which allows it to be always callable which deviates from specification inferred from code comment.

b. **Details**: The `@dev` comment for `exited` variable says: "*@dev After bentobox 'exits' the strategy harvest, skim and withdraw functions can no loner be called.*" However, the `skim` function is missing the `isActive` modifier (which checks for the `exited` variable) and therefore allows `skim` to be always callable. After the strategy is no longer active, there should be no reason for `skim` to be called.

c. **Github Permalinks**: `skim`, Comment

d. **Mitigation**: Use `isActive` modifier on `skim` function or correct the comment depending on which is the correct specification of expected behaviour.

9. **Missing events for critical functions**

a. **Summary**: Critical functions do not emit events.

b. **Details**: Events should be emitted for critical functions that are access-controlled, setters of protocol parameters or those affecting protocol in significant ways:

i. `afterExit` which is `onlyOwner` and has the comment: "*@dev After exited, the owner can perform ANY call. This is to rescue any funds that didn't get released during exit or got earned afterwards due to vesting or airdrops, etc.*" is missing an event.

ii. `exit` which allows `bentoBox` to transfer all tokens and exit the strategy.

iii. `withdraw` which allows `bentoBox` to withdraw from strategy.

iv. `safeHarvest` and `harvest` which allows `bentoBox` to harvest from strategy.

v. `swapExactTokensForUnderlying` allows the executor to swap tokens for the underlying.

vi. The `catch` blocks of `_exit`.

c. **Github Permalinks**: `afterExit`, `exit`, `withdraw`, `safeHarvest`, `harvest`, `swapExactTokensForUnderlying`, `_exit`

d. **Mitigation**: Emit appropriate events with necessary parameters.

10. **Unchecked downcasting**

a. **Summary**: Casting from `uint256` to `int256` should be checked for potential overflow scenarios.

b. **Details**: Downcasting in Solidity does not revert on overflows which can result

in vulnerabilities because developers usually assume that overflows raise errors with Solidity compiler >= 0.8. For example, downcasting from `uint256` to `int256` would truncate and overflow the value to a negative number if it were greater than `type(int256).max`.

i. `BaseStrategy harvest` casts `uint256 contractBalance` to `int256`.
ii. `AaveStrategy harvest` casts `uint256 currentBalance` to `int256`.
iii. `BaseStrategy exit` casts `uint256 balance` and `actualBalance` to `int256`.

c. **Github Permalinks**: Solidity, `BaseStrategy.harvest`, `AaveStrategy.harvest`, `BaseStrategy.exit`
d. **Mitigation**: While the particular findings reported appear highly unlikely to overflow, nevertheless evaluate if these values will ever exceed the max values of their downcast types and use appropriate checks or OpenZeppelin's `SafeCast` library as required.

11. Checks-Effects-Interactions (CEI) pattern is not followed

a. **Summary**: Not following CEI pattern is risky in the reentrancy context.
b. **Details**: Following CEI pattern is a best-practice even if external interactions with contracts are trusted. The protocol could interact with arbitrary tokens that may not be within the trust model. Such tokens may be malicious forcing reentrancies where sufficient mitigations are not in place.
   i. `exit` sets the `exited` storage variable to true after interaction with the external `strategyToken` contract. This may lead to undefined behaviour and unexpected exploitation scenarios if `strategyToken` can be controlled by an attacker.
   ii. `harvest` calls `_skim` after interaction with the external `strategyToken` contract. This may lead to undefined behaviour and unexpected exploitation scenarios if `strategyToken` can be controlled by an attacker.
c. **Github Permalinks**: `exit`, `harvest`
d. **Mitigation**: Follow best-practice to adhere to the CEI pattern wherever possible and use reentrancy guard wherever required.

12. Incorrect `pathId` emitted in `LogSetAllowedPath` event

a. **Summary:** `setAllowedPath` emits an event with the length of `_allowedSwapPaths` array instead of its index that was just set.
b. **Details:** Incorrect event emission may impact off-chain monitoring. `LogSetAllowedPath` is declared to emit pathIndex as its first parameter but `setAllowedPath` (unlike in the other two places) emits this event with the length of `allowedSwapPaths` array instead of its index (last element) that was just set.

Monitoring tools will have incorrect information about which path indices are set and the index of a certain path. If event information is used to call `swapExactTokensForUnderlying`, a swap for incorrect tokens may occur because of an incorrect path index being emitted.

c. **Github Permalinks:** `LogSetAllowedPath`, `setAllowedPath`

d. **Mitigation:** Change `emit LogSetAllowedPath(_allowedSwapPaths.length, true)` to `emit LogSetAllowedPath(_allowedSwapPaths.length - 1, true)`

13. Missing sanity/threshold checks in input validation

a. **Summary**: Sanity/threshold checks (depending on their types) on input parameters are missing.

b. **Details**:

i. There is no input validation in `setAllowedPath` which allows paths of length less than 2 to be added. A valid path should have a length of at least 2. The emission of `LogSetAllowedPath` for such invalid paths may confuse event monitoring tools. A check for `path.length >= 2` is required before adding the path.

ii. There is no input validation in `setAllowedPath` that disallows paths of length longer than some predefined threshold which may cause out-of-gas errors for `swapExactTokensForUnderlying`.

iii. There is no input validation in `setAllowedPath` which checks that the addresses in the path are different and valid.

iv. There is no input validation for valid `pathIndex` in `getAllowedPath` and `swapExactTokensForUnderlying`.

v. There is no input validation to check that the final output token in `swapExactTokensForUnderlying` is equal to the underlying strategy token.

vi. The zero-address check for the `factory` can be moved from `swapExactTokensForUnderlying` to the `constructor`.

c. **Github Permalinks:** `setAllowedPath`, `getAllowedPath`, `swapExactTokensForUnderlying`, `factory`

d. **Mitigation:** Perform appropriate sanity/threshold checks (depending on their types) on input parameters.

14. Implicit check for disabled paths could be confusing

a. **Summary:** Disabled paths are not checked explicitly. Instead, they are implicitly checked by accessing `path[0]` and forcing an out-of-bounds error for disabled paths.

b. **Details:** The expectation is that because a path is disabled by setting its index in `_allowedSwapPaths` to an empty array, attempting to access its `path[0]` will cause an out-of-bounds error and revert. However, this revert can be ambiguous because an out-of-bounds error can also occur while accessing `path[1]` later because of setting a path of length 1 (due to missing input validation in `setAllowedPath`).

c. **Github Permalinks:** `path[0]`, `path[1]`

d. **Mitigation:** Add a custom error for when `path.length == 0`.

15. Code readability

a. **Summary**: Improving code readability will help code developers, maintainers and reviewers.
b. **Details**: Instead of having an `if` statement in the `harvest` function spanning the whole function body, it would be more readable to replace the long `if` block with an immediate `if-return` at function entry.
c. **Github Permalinks**: `harvest`
d. **Mitigation**: Consider code layout/structure best-practices to improve readability.

16. **Account existence check for low-level calls**
a. **Summary**: Account existence for low-level calls is not checked.
b. **Details**: Low-level calls `call/delegatecall/staticcall` return true even if the account called is non-existent (per EVM design).
`afterExit` makes a low-level `call` to the `to` address specified without checking for contract existence at that address.
c. **Github Permalinks**: `afterExit`
d. **Mitigation**: Account existence must be checked prior to calling.

17. **Time-delayed change of critical parameters is absent**
a. **Summary**: Change of critical parameters should be emitted via events and enforced only after a time delay.
b. **Details**: When critical parameters of systems need to be changed, it is a best-practice to broadcast the change via event emission and recommended to enforce the changes after a time-delay. This is to allow system users to be aware of such critical changes and give them an opportunity to exit or adjust their engagement with the system accordingly. For e.g. reducing the rewards or increasing the fees in a system might not be acceptable to some users who may wish to withdraw their funds and exit. This allows a malicious or compromised owner account to change all these critical parameters without giving users time to react.
`BaseStrategy` `onlyOwner` functions `setStrategyExecutor`, `setAllowedPath` and `disallowPath` which set/change critical addresses and paths in these contracts could apply a timelock.
c. **Github Permalinks**: `setStrategyExecutor`, `setAllowedPath`, `disallowPath`
d. **Mitigation**: Critical address/parameter updates should happen only after an adequate time delay so users have time to react. If not, this should be clearly documented and communicated to all users as a potential risk.

18. **Named returns**
a. **Summary:** Mixing of named and explicit returns is confusing.
b. **Details:** The use of named returns in a few places decreases code readability and may be confusing for developers/reviewers.
c. **Github Permalinks:** `BaseStrategy.actualAmount`, `BaseStrategy.amountAdded`, `AaveStrategy.amountAdded`
d. **Mitigation:** Reconsider the use of named returns and instead explicitly use return statements with values to increase readability.

19. **Missing new != old checks in setter functions**
a. **Summary:** Setter functions should check that the new value being set is not the same as the old value being replaced.

b. **Details:** Setter functions that set state variables should check that the new value being set is not the same as the old value being replaced. While this does not affect contract state, it may lead to mismatched offchain-onchain state assumptions or events being emitted that confuse contract monitoring systems.
c. **Github Permalinks:** `setStrategyExecutor`, `setAllowedPath`, `disallowPath`
d. **Mitigation:** Add input validation in setters to check that the new value being set is not the same as the old value being replaced.

20. **Inline vendor interfaces**
a. **Summary:** Inline vendor interfaces reduce readability and reuse.
b. **Details:** `AaveStrategy` includes several inline vendor interfaces to the Aave protocol, such as `ILendingPool` and `IAaveIncentivesController`. This reduces readability and reuse.
c. **Github Permalinks:** `ILendingPool`, `IAaveIncentivesController`
d. **Mitigation:** Consider defining these vendor interfaces externally outside the contract and importing them because that would improve readability and reusability with the same interfaces potentially reused in other strategies.

21. **Gas optimizations**
a. **Summary:** There are various best-practices to make code more gas efficient.
b. **Details**: Functions that accept an amount parameter and transfer tokens/ETH based on that amount should check for non-zero amounts before making external calls.
c. **Github Permalinks:** `skim`, `withdraw`
d. **Mitigation:** Check for non-zero amounts before making external calls.

22. **`BaseStrategy.exit` can be called only once**
a. **Summary:** The comment in `BaseStrategy.exit` states "*allow bentobox to call strategy.exit() multiple times.*" But `BentoBox.setStrategy` is allowed to call it only once.
b. **Details:** `BaseStrategy.exit` is called by `BentoBox` only when a strategy is replaced. It is not possible to call it a second time unless the replaced strategy is registered again.
c. **Github Permalinks:** `BaseStrategy.exit`, `BentoBox.setStrategy`
d. **Mitigation:** Clarify documentation or evaluate/implement flows that will allow the documented behaviour.

23. **`strategyExecutors` not checked for EOAs**
a. **Summary:** It is commented that `strategyExecutors` are Externally-Owned-Accounts (EOA) but it is not checked anywhere.
b. **Details:** `BaseStrategy` comments that `strategyExecutors` are "*/// @dev EOAs that can execute safeHarvest.*" However, neither the `constructor` nor `setStrategyExecutor` which add strategy executors check for their addresses to be EOA i.e. not contracts.
c. **Github Permalinks:** `strategyExecutors`, `constructor`, `setStrategyExecutor`

d. **Mitigation:** Clarify documentation or implement checks for `strategyExecutors` addresses to be EOAs.

24. **Insufficient test coverage**
    a. **Summary:** The coverage of existing automated tests is insufficient.
    b. **Details:** Example scenarios which appear to lack test coverage:
        i. Access control of `BaseStrategy's getAllowedPath`, `setAllowedPath`, `disallowPath` and `setStrategyExecutor` functions.
        ii. `BaseStrategy's isActive` modifier application and effectiveness.
        iii. Effectiveness of the `maxChangeAmount` parameter.
        iv. Effectiveness of slippage protection offered by the `amountOutMin` parameter of `BaseStrategy.swapExactTokensForUnderlying`.
        v. Correctness of `BaseStrategy._swap` for paths with more than 2 assets.
    c. **Github Permalinks:** All contracts and tests.
    d. **Mitigation:** Increase coverage of automated tests.

25. **Error messages**
    a. **Summary**: Custom errors may reduce gas costs.
    b. **Details**: `disallowPath` uses a `require` check with an error message. To save gas, this can be replaced by custom errors which were introduced in Solidity `0.8.4`.
    c. **Github Permalinks**: `disallowPath`
    d. **Mitigation**: Consider using custom errors instead of `require` checks.

26. **`afterExit` function does not use OpenZeppelin's `Address` library**
    a. **Summary**: `BaseStrategy's afterExit` uses a low level `call` where mandatory checks are missing.
    b. **Details**: OpenZeppelin maintains a library of standard, audited, community-reviewed and battle-tested smart contracts. `Address` library has everything needed to achieve `afterExit` functionality in a secure way.
    c. **Github Permalinks**: `afterExit`
    d. **Mitigation**: Consider using the `functionCallWithValue` from `Address` library which incorporates contract existence check and `call` return value check.

27. **Incorrect operator**
    a. **Summary:** `harvest` should use '>' instead of '>=' for profit check.
    b. **Details:** `harvest` checks for profit reported by `_harvest` using the '>=' operator against 0, but a value of 0 is a breakeven and not a profit. It is a profit only when the value is '`> 0`'.
    c. **Github Permalinks:** `harvest`
    d. **Mitigation:** Replace '>=' by '>'.

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.