Secureum CARE Report: **Sherlock**

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): GDee#8611, Lyner#9077, 0xEmile 🇦🇲#8143, Sahar#3109, Zer0dot#1420, obelisk#8700, AgusDuha#6717, Callius#1540, bravo#4464, qbs#2188, datapunk#6402, harrigan#4239, TimmyToes#2581, pedrorivera#4394, danielrees.eth#5336, Smye#7489, minter#1369, IvanP#3695, RamiRond#1730, aysha_a#0860, olivierdem#4348, baltaromero#9741, 0xkowloon#3337, markus#2108 & Rajeev | Secureum#7615

Scope: Review focused on these repositories/commits of Sherlock protocol V2: https://github.com/sherlock-protocol/sherlock-v2-core/tree/877056fd1d30aa2d74db7f673ee289ddc75e449f

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. **Separation of privileges and time-locked actions**
    a. **Summary**: Protocol uses the `onlyOwner` modifier (from `Ownable`) to restrict access to privileged actions. Currently this is intended to be a multisig and comments suggest it will transition to DAO over time. There is only one such admin role and it is able to call all functions with no time delays.
    b. **Details**: By having a single admin address, the risk of private keys being exposed/phished is higher due their increased use, as is the risk to the system in the event that it happens. If an EOA is used instead of a multisig then EOA has a bigger risk of being compromised and malicious actors can potentially change protocol parameters and can do so immediately without a timelock.
    c. **Github Permalinks**: `Sherlock`, `Manager`
    d. **Mitigation**:
        i. Consider changing the design to use two admin addresses. One role can be used for less critical actions such as updating constants, pausing the system or other lower risk events while the other separate role can manage more critical system operations such as updating manager addresses, addition or removal of protocols and unpausing. Such a separation of privileges used with time-locked delayed actions is a best-practice to prevent single points of immediate failure.
        ii. Owners for different contracts could be different addresses for separation of privileges.
        iii. Consider not having any EOAs but only multisigs with a reasonable configuration of say 3-of-5 or 5-of-9, and signatories being independent actors.
        iv. Consider having all privileged roles controlled by OpenZeppelin's `Governor` contract with time lock. Consider having `SHER` token to inherit from OpenZeppelin's `ERC20Votes` for snapshots and vote delegation.
        v. Consider OpenZeppelin's role based access control, even if one or more of those roles would be a DAO or multisig.
2. **Missing zero-address validation**
    a. **Summary**: Missing zero-address validation for address parameters may lead to ETH/tokens getting burned or reverts.
    b. **Details**: The following address parameters are missing zero-address checks:
        i. `_sherDistributionManager` in `Sherlock.sol`. The `updateSherDistributionManager` function includes a zero address check suggesting we should also have one for `_sherDistributionManager` in the constructor.

ii. `aaveLmReceiver` in `AaveV2Strategy.sol`. This is especially important for the `aaveLmReceiver` as it handles user funds and there is no function to change the address once it is set in the constructor.
iii. `_sherlock` in `Manager.sol`
iv. `_receiver` in `Manager.sol`

c. **Github Permalinks**: Sherlock, AaveV2Strategy, Manager._sherlock, Manager._receiver

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

3. **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.
Given that all contracts inherit from `Ownable`, the following protocol addresses can be changed in a single-step: `Sherlock`, `sherDistributionManager`, `nonStakerAddress`, `sherlockProtocolManager`, `sherlockClaimManager` and `yieldStrategy`. If set to the wrong address this could lead to loss of funds sent to it or even worse allow a malicious smart to interact with the project.

c. **Github Permalinks**: Sherlock, Manager, sherDistributionManager, nonStakerAddress, sherlockProtocolManager, sherlockClaimManager, yieldStrategy

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.

4. **Protocol pause/unpause may not work as expected**

a. **Summary**: Sherlock `pause` and `unpause` call related functions in each of the four underlying manager contracts. If any one of these calls fails then the pause/unpause of the entire protocol also fails.

b. **Details**: Sherlock `pause` calls the pause function in all four manager contracts: `yieldStrategy`, `sherlockProtocolManager`, `sherlockClaimManager` and `sherDistributionManager`. If one of

the first three contracts (`yieldStrategy`, `sherlockProtocolManager`, `sherlockClaimManager`) is set to the wrong address or, what is more likely as specified in the protocol, if `sherDistributionManager` is set to a zero-address from a prior call to `removeSherDistributionManager` then the call to Sherlock `pause` function will fail. All of this is also true for `unpause`. This will make it impossible to pause/unpause contracts as expected, which could be critical in an emergency incident response situation.

c. **Github Permalinks**: `Sherlock.pause`, `Sherlock.unpause`

d. **Mitigation**:

i. Ensure that the contracts mentioned are set to the correct addresses.

ii. Zero-address checks should be included on the `sherDistributionManager` address and if/when it is removed (to stop release of `SHER` rewards as documented) then a boolean indicating that it has been removed can be used to avoid calling `pause` on it or simply checking if it's a non-zero address before calling `pause`/`unpause` on it.

iii. Enable (un)pausing each manager independently.

This enables admins who call `pause`/`unpause` in the Sherlock contract to be certain that all four underlying active contracts can be paused/unpaused and that the overall protocol behaves as expected without failure.

5. **Similar variable names**

a. **Summary**: Variables with similar names could be confused for each other and therefore should be avoided.

b. **Details**:

i. Function `initialStake` makes use of both local variable `totalStakeShares_` and global `totalStakeShares` in equations. Because both are `uint256` values and initially set to the same value, it could be difficult to notice an error if one was mistakenly used instead of the other.

ii. Function `pullReward` has a call to the ERC20 `sher` and uses a local `uint256` variable `_sher` to store a reward quantity.

iii. Multiple functions have a local variable `premium` that is used to cache the state variable `premiums_` which also has a getter function called `premium`.

c. **Github Permalinks**: `initialStake`, `pullReward`, `premium`

d. **Mitigation**: Consider changing the names of similar variables so they are distinct and accurately represent their intended purpose.

6. **Redundant local variable**

a. **Summary**: The outcome of a function is stored in a local variable that is then copied to another variable which is returned thus making the first variable redundant.
b. **Details**: In `viewRewardForArbRestaking`, the boolean result of `_calcShareForArbRestake` is stored in `_able`. This value is then copied to the named return boolean `able`, effectively making the declaration/use of `_able` unnecessary.
c. **Github Permalinks**: `viewRewardForArbRestake`
d. **Mitigation**: Store the outcome of `_calcShareForArbRestake` in the named return variable `able`.

7. **Return value of functions ignored**
    a. **Summary**: The return value of functions need to be checked for appropriate values/conditions to determine error conditions appropriately.
    b. **Details**:
        i. `yieldStrategy.WithdrawAll`: final amount withdrawn by `AaveV2Strategy.withdrawAll` is ignored.
        ii. `AaveV2Strategy.claimRewards`: amount of rewards claimed is ignored.
        iii. `SherlockClaimManager.escalate`: `totalBond` requested by `UMAOO` is ignored.
        iv. `SherlockClaimManager.escalate`: return value of `UMA.disputePriceFor` is ignored.
    c. **Github Permalinks**: `yieldStrategy.WithdrawAll`, `AaveV2Strategy.claimRewards`, `SherlockClaimManager.escalate`, `SherlockClaimManager.escalate`
    d. **Mitigation**: Consider checking return values of the functions to detect error codes/conditions and verify expected values.
8. **Additional tokens earned by `yieldStrategy` are locked/lost**
    a. **Summary**: Strategies may yield extra tokens as part of their yield like `stkAAVE`. These are sent to `aaveLmReceiver` where they get locked/lost because there is no implemented way to distribute them to stakers in the protocol.
    b. **Details**: Function `claimRewards` sends any additional reward tokens generated by a yield strategy to the `aaveLmReceiver` address. The mechanism by which such tokens are distributed to stakers appears unimplemented and undocumented.
    c. **Github Permalinks**: `claimRewards`

d. **Mitigation**: Implement and document the process by which other token rewards generated by a yield strategy are returned to stakers in the protocol.

9. **Insufficient protocol arbitrage incentives after system paused**
    a. **Summary**: Insufficient protocol arbitrage incentives compared to the gas fee could be incurred after the system has been paused long enough.
    b. **Details**: Because protocol removal arbitrage functions cannot be called while the system is paused, it is possible that the time spent in the paused state results in insufficient protocol premiums being leftover after the system is unpaused to incentivize arbitrageurs to remove protocols. This is because the gas paid by the arbitrageur may exceed any return they earn and so they have no incentive to remove the protocol. This would lead to accounting errors because the system would wrongly assume the protocol was still paying for their coverage.
    While it doesn't cause explicit loss of funds, it does break accounting systems. Non-removed protocols who have insufficient funds to pay for coverage will cause debt accounting to be incorrect.
    c. **Github Permalinks**: `SherlockProtocolManager.forceRemoveByActiveBalance`, `SherlockProtocolManager. forceRemoveBySecondsOfCoverage`
    d. **Mitigation**: Monitor protocol active balances while the system is paused and either refrain from pausing the system long enough to make arbitrage profits insufficient or, if this is not possible, Sherlock owner must manually remove these protocols using `protocolRemove`.

10. **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. The bond deposit amount in code and comment is different: The code has bond deposit as 9600 USDC as the deposit required for escalating a claim to UMA's optimistic oracle. However, the code comment states it should be 20K USDC while this value in the developer documentation is listed twice as 5K then later as 20K USDC.
        ii. The bond amount for escalating a claim is documented to be upgradable with a large timelock but it is implemented as a constant.
        iii. ETH and tokens: The `sweep` function removes ERC20's and ETH from an inactive `yieldStrategy` but the comments on L114 in function `isActive` do not fully reflect this, only mentioning ERC20's.
        iv. `cleanup`: This comment should say "*not* pursued".

v. The comment: "*// balanceOf() returns the USDC amount owed to this NFT ID*" should say `tokenBalanceOf()` instead of `balanceOf()`.
vi. The NatSpec comment for function `updateYieldStrategy` indicates that it would try to call `yieldStrategyWithdrawAll` before changing the strategy. However this is not actually implemented. If the comment is correct, then this call should be implemented.
vii. `ARB_RESTAKE_MAX_PERCENTAGE` is 20% not 10% as commented.
viii. Incomplete comment.
ix. Comment refers to a non-existing variable `claimablePremiumsStored`.
x. `MIN_SECONDS_LEFT` is 7 days and not 3.
xi. `token` could be renamed `usdcToken` because the current specification (trust/threat models) is specifically for USDC.
xii. Comment states that full balance can be withdrawn if premium is 0, but having a premium of 0 would fail a conditional check.
xiii. `DECIMALS` variable is not what it claims because it contains a scalar value used to multiply a number to achieve effective decimals instead of the number of decimals itself.
xiv. Questions, similar to TODOs, in code comments should be removed as they imply that the code is incomplete or buggy, or that the implementation may not have been entirely followed.
xv. Comment "*// Economically spearking*" should be "*// Economically speaking*"
xvi. Typo: "*Succes*" should be "*Success*"
xvii. Incomplete comment: "*// This is a beginning of an*"
xviii. Comments on line 385, 402 of `SherlockClaimManager.sol` are incorrect. It does not allow `msg.sender` to be the current protocol agent, but rather the one that initiated the claim.
xix. The address being transferred to is the parameter `_receiver`, not the nonstaker address as commented. Nonstaker address is earlier checked against `msg.sender` but does not need to be the recipient.
xx. Comment implies `SherlockCoreSet` is an error, not an event.
xxi. Comment in `claimPremiumsForStakers` claims this is called by `burn` function that does not exist in Sherlock contract. Also, there are many other functions that call `claimPremiumsForStakers`.
xxii. Comment should say `ReadyToProposeUmaDispute` and not `UmaDisputeProposed`.
xxiii. Label "nonstakers" is confusing in the documentation: "Nonstakers" are people who are getting paid but not because of staking. Given that everyone other than stakers are nonstakers, consider using a different label such as "fee earners" or "service providers".
xxiv. `SHER` token specification is missing e.g.: How will `SHER` get to `SherDistributionManager`? What systems are in place to stop it running out? What is the plan to remove excess `SHER` if the

`SherDistributionManager` contract is changed? Is `SHER`'s decimals 6?

xxv. Guarded launch specification is missing: Trusted actors, their roles (more details at https://docs.sherlock.xyz/developer/roles), capabilities, multisigs, pausing/unpausing details, emergency withdrawal scenarios and flows.

xxvi. Specification/Documentation: A lot of the documentation is in the comments which could be compiled and merged into official documentation with more details on different implemented scenarios and flows. Create a separate specification that describes the expected scenarios and flows.

c. **Github Permalinks**:
   i. BOND, Docs Exploit flow, Docs Claims process, Docs Claims lifecycle
   ii. BOND, Comment
   iii. Code, Comment
   iv. Comment
   v. Comment
   vi. Code, Comment
   vii. Code, Comment
   viii. Comment
   ix. Comment
   x. Code, Comment
   xi. Code and Comment
   xii. Code, Comment
   xiii. Code
   xiv. Questions
   xv. Comment
   xvi. Typo
   xvii. Comment
   xviii. Code, Comment
   xix. Code, Comment
   xx. Code, Comment
   xxi. Comment
   xxii. Comment

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

11. Unused variables

a. **Summary**: Unused variables indicate missing/unnecessary functionality

b. **Details**: Two parameters of `pullReward` function: `_id` and `_receiver` are never used and the Natspec comment acknowledges this but without any explanation for their presence.

c. **Github Permalinks**: `pullReward`

d. **Mitigation**: Remove the unused variables or document why they need to be present.

12. **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`. It is best practice to simply test `if (variable)` on its own for true, and `if (!variable)` for false.
        i. `Manager._sweep`
        ii. `SherDistributionManager.sweep`
        iii. `SherlockClaimManager.payoutClaim`
        iv. `SherlockClaimManager.escalate`
        v. `SherlockClaimManager.cleanup`
        vi. `SherlockProtocolManager.forceRemoveBySecondsOfCoverage`
    c. **Github Permalinks**: `_sweep`, `sweep`, `payoutClaim`, `escalate`, `cleanup`, `forceRemoveBySecondsOfCoverage`
    d. **Mitigation**: Remove the boolean constant or replace it with just the boolean variable or with '!' as necessary.

13. **Disputed claims may lead to loss of funds across disputing protocols**
    a. **Summary**: A protocol whose claim is disputed and escalated to `UMAOO` can take any excess USDC from `SherlockClaimManager`.
    b. **Details**: When escalating a claim to `UMAOO`, the protocol is assumed to be the original sender of all USDC present in the contract. The protocol agent initially sends some USDC to the claim manager to cover the bond required by `UMAOO`. However, once this process has been initiated, any USDC remaining in the contract is then sent back to the protocol agent. If claims from two different protocols are active and one of them is being paid by a reinsurer (as mentioned in the code comments), it is possible that `SherlockClaimManager` may hold USDC from the reinsurer that is unrelated to one of the two protocols and which could be stolen by that protocol.
    c. **Github Permalinks**: `escalate`
    d. **Mitigation**: Prior to `SherlockClaimManager` receiving the bond from a protocol, it should record its balance of USDC. Then if `UMAOO` decides to take less USDC than the specified bond, Sherlock would know how much USDC belongs to each escalating protocol and only this amount can be returned.

14. **State variables can be declared constant to save gas**
    a. **Summary**: State variables whose values never change can be declared as constants to save gas by avoiding unnecessary SLOADs during their reads.
    b. **Details**: `MAX_CALLBACKS` is never changed and can be declared as a

constant.

c. **Github Permalinks**: MAX_CALLBACKS

d. **Mitigation**: Variables whose values are constant should be declared accordingly to prevent accidental changes in future code updates and to save gas.

15. **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 it should be marked as `external`. This will save gas.

c. **Github Permalinks**: lockupEnd, sherRewards

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

16. **Missing events for critical functions**

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

b. **Details**: Events should be emitted for critical functions such as those with `onlyOwner` modifier or those affecting protocol in significant ways:

i. `yieldStrategyDeposit`
ii. `yieldStrategyWithdraw`
iii. `yieldStrategyWithdrawAll`
iv. `initialStake`
v. `redeemNFT`
vi. `_sweep`
vii. `nonStakersClaim`
viii. `claimPremiumsForStakers`
ix. `claimRewards`

c. **Github Permalinks**: yieldStrategyDeposit, yieldStrategyWithdraw, yieldStrategyWithdrawAll, initialStake, redeemNFT, _sweep, nonStakersClaim, claimPremiumsForStakers, claimRewards

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

17. **Missing sanity/threshold checks in input validation**

a. **Summary**: `enableStakingPeriod` should have sanity/threshold checks for `_period` parameter.

b. **Details**: If the owner sets a very large period or very close to existing periods accidentally, there is no sanity/threshold check to prevent this and will require detection and disabling of such periods later.

c. **Github Permalinks**: enableStakingPeriod

d. **Mitigation**: Add threshold validation for what is considered a reasonable maximum staking period. Also, given that staking periods are typically meaningful with differences of days/weeks/months from each other, it may help to enforce that via modulus and minimum difference from existing periods. Without that, periods differing even by 1 second will be considered as valid and added here.

18. **Checks after effects pattern is risky and inefficient**

a. **Summary**: `_stake` function modifies state before checks and this could lead to stale state and inefficient processing.
b. **Details**: In Sherlock `lockupEnd`, the provided position is updated followed by checks for `SHER` distribution and staked amount. These checks could precede the state change unless there is a reason to record the lockup period for positions that are staked with 0 amount or even when `SHER` rewards have stopped.
c. **Github Permalinks**: `_stake`
d. **Mitigation**: Following Checks-Effects-Interaction pattern, all relevant checks should be performed before modifying state.

19. **Time-delayed change of critical addresses/parameters is absent**
a. **Summary**: Change of critical addresses/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 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.
    i. `removeSherDitributionManager` function allows the owner to remove the distribution manager without giving stakers a warning period. It should be timelocked to allow users to unstake their tokens when `SHER` rewards are stopped.
    ii. *`updateSherDistributionManager`*, *`updateYieldStrategy`*, *`updateNonStakersAddress`*, *`updateSherlockProtocolManager`*, *`updateSherlockClaimManager`*
    iii. `disableStakingPeriod` allows the owner to remove a staking period.
c. **Github Permalinks**: `removeSherDistributionManager`, `updateSherDistributionManager`, `updateYieldStrategy`, `updateNonStakersAddress`, `updateSherlockProtocolManager`, `updateSherlockClaimManager`, `disableStakingPeriod`
d. **Mitigation**: Allow critical parameter updates to happen only after an adequate time delay so users have time to react. Optimally, this delay should be larger than the greatest possible staking period. If it is less, this should be clearly communicated to all users as a potential risk.

20. **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 a reentrancy guard is used or if the external interaction is trusted. The protocol interacts only with its own `SHER` token and USDC token, both of which are within the trust model. The only other external (beyond the Sherlock contracts) contract interactions are with UMA and other claim callback contracts (e.g. reinsurer) whose calls

are protected by the `nonReentrant` modifier. Nevertheless, it is a best-practice to use CEI pattern and reentrancy guard wherever possible, in case Sherlock decides to re-use the contract for other tokens or interact with other untrusted external contracts.

i. Comment in `SherlockProtocolManager` indicating that there is a concern about token callbacks therefore enforcing CEI pattern
ii. `_sendSherRewardsToOwner`: `transfer` function of the `SHER` token is called before deleting the current rewards. Even if token `SHER` is known and controlled by the protocol, it is a good practice to follow CEI pattern.
iii. `_redeemShares` reduces `stakeShares` after transfer of token payouts.
iv. `Sherlock.initialStake` makes external calls before updating state variables.
v. `SherlockProtocolManager.depositToActiveBalance` makes external calls before updating state variables.
vi. `SherlockProtocolManager._forceRemoveProtocol` makes external calls before updating state variables.
vii. `Sherlock.redeemNFT` makes external calls before updating state variables.
viii. `Sherlock._restake` makes external calls before updating state variables.
ix. Several other places where events are emitted after making external contract calls.

c. **Github Permalinks**: Comment, `_sendSherRewardsToOwner`, `_redeemShares`, `initialStake`, `depositToActiveBalance`, `_forceRemoveProtocol`, `redeemNFT`, `_restake`
d. **Mitigation**: Follow best-practice to adhere to the CEI pattern wherever possible and use reentrancy guard wherever required.

21. Solidity version

a. **Summary**: Using the latest versions might make contracts susceptible to yet undiscovered compiler bugs.
b. **Details**: Sherlock project uses Solidity compiler version `0.8.10` which was released in November 2021 (one month ago at time of the CARE review).
c. **Github Permalinks**: All contracts.
d. **Mitigation**: Consider using slightly older versions such as `0.8.7`.

22. Claimed premiums are not deposited into yield strategy

a. **Summary**: Newly claimed premiums for stakers are not put into yield strategy.
b. **Details**: Function `yieldStrategyDeposit` puts specified amount into yield strategy. Before doing so, it claims premiums available for stakers which are however retained in Sherlock contract and not put into yield strategy.
c. **Github Permalinks**: `yieldStrategyDeposit`

d. **Mitigation**: Deposit the newly claimed tokens to the yield strategy.

23. **Test-specific code in production**
    a. **Summary**: Any code or parameterisation used specifically for testing should be removed from production code.
    b. **Details**: There is a check for Hardhat chain ID in `setSherlockCoreAddress` function which should not be carried over to production.
    c. **Github Permalinks**: `setSherlockCoreAddress`
    d. **Mitigation**: Remove test-specific code from production and adapt the test suite to work without this test parameterization.
24. **Use of older `require` instead of custom errors**
    a. **Summary**: Use of custom error should be preferred over older `require`.
    b. **Details**: While most code uses custom errors, there are two instances of using `require`.
    c. **Github Permalinks**: `SherlockProtocolManager`, `SherlockDistributionManager`
    d. **Mitigation**: Replace the `require` statements with new custom errors to be consistent and efficient.
25. **Explicit return missing along function paths**
    a. **Summary**: All logical control flow paths should return values.
    b. **Details**: In `SherlockClaimManager`, `_isCleanupState` returns `bool`, but not all logical control flow paths explicitly return a bool value. The return value for non-specified states (!=`State.SpccDenied` and !=`State.SpccPending`) is missing in which case the function returns the default value of `false`.
    c. **Github Permalinks**: `_isCleanupState`
    d. **Mitigation**: Add explicit return statements for all non-reverting code paths.
26. **Function state mutability**
    a. **Summary**: Functions that do not read contract state can be marked `pure`.
    b. **Details**: In `SherlockClaimManager`, `_isCleanupState` is marked `view` but does not read contract state. Hence it can be marked `pure`.
    c. **Github Permalinks**: `_isCleanupState`
    d. **Mitigation**: Restrict state mutability to `pure`.
27. **Multiplication after division**
    a. **Summary**: Multiplication after division may lose precision.
    b. **Details**: Operations performing multiplication on the result of a division may lead to precision errors leading to less `SHER` rewards sent to the core contract.
    c. **Github Permalinks**: `SherlockDistributionManager`
    d. **Mitigation**: Perform multiplication before division.
28. **Use of coverage amount**
    a. **Summary**: The claim process uses the maximum of the current and previous coverage amounts. This is susceptible to insider attacks. The logic controlling the effective coverage level might sometimes cause a protocol's coverage to change in ways that might not be expected. In a case where the coverage amount has changed significantly, this might cause a business level issue.

b. **Details**: A malicious protocol could try to get a higher claim for an exploit than it is entitled to based on premiums paid. This is only possible if unnoticed by Sherlock protocol because only the owner can update protocol parameters which includes coverage amount.
The 'previous' coverage level is effectively a current coverage level because `SherlockClaimManager` sets `maxClaim` to be the greater of the two coverage levels (current and previous) registered for the protocol. There are a few scenarios where this could be an issue:
    - Coverage is lowered. If a protocol lowers its coverage, the value of its previous coverage does not expire. It could theoretically continue silently for years, effectively receiving extra value and exposing Sherlock stakers to additional risk while paying a reduced premium.
    - `nonStakersPercentage` is modified. This would be done by a call to `protocolUpdate` which could be a relatively minor administrative change but would result in a potentially large change in coverage size as the previous coverage amount is automatically overwritten.
    - A protocol is removed and added. A protocol's previous coverage is deleted if it is added after removal.
c. **Github Permalinks**: `startClaim`, `protocolUpdate`
d. **Mitigation**: Consider using the coverage amount that was active when the exploit first occurred as maximum claimable value or make coverage based on what most premiums have been paid for. Coverage levels could also be persisted for a fixed period of time regardless of other changes.

29. No indexed event parameters

a. **Summary**: None of Sherlock's events use indexed parameters. This could make it harder and inefficient for off-chain tools to analyse them.
b. **Details**: Indexed parameters ("topics") are searchable event parameters. They are stored separately from unindexed event parameters in an efficient manner to allow for faster access. This is useful for efficient off-chain-analysis, but it is also more costly gas-wise.
c. **Github Permalinks**: This applies to all Sherlock events (which are defined in interface contracts). Examples: `ISherlock.sol`, `ISherlockClaimManager.sol`, `ISherlockProtocolManager.sol`.
d. **Mitigation**: Consider which event parameters could be particularly useful for off-chain tools and index those.

30. Redundant event emission could be confusing

a. **Summary**: Emitting similarly named events twice with potentially different values by different protocol contracts could be confusing.
b. **Details**: Both `Sherlock` and `SherlockClaimManager` emit an event called `ClaimPayout` in the same transaction during a claim payout. The emitted claim values may not be identical: `SherlockClaimManager` emits the total amount sent (by `SherlockClaimManager` and `Sherlock`), while `Sherlock` only emits its portion.
c. **Github Permalinks**: `SherlockClaimManager`, `Sherlock`
d. **Mitigation**: Consider either removing event emission in `Sherlock` or renaming events to clearly describe their semantics, for example `CoreClaimPayout` and `ManagerClaimPayout`.

31. **Gas optimizations**
    a. **Summary**: Consider gas optimizations from cached usage in memory, constant/immutable variables, checking for zero amounts where required, simplifying operations, avoiding external calls where possible and using memory instead of storage data location specifiers appropriately.
    b. **Details**:
        i. State variable `premiums_[_protocol]` is copied to a local memory variable `premiums` but the state variable is later used instead.
        ii. Redundant memory copies of immutable variables `maxRewardsEndTVL` and `zeroRewardsStartTVL` are created.
        iii. Replace (`maxRewardArbTime - initialArbTime`) by constant `ARB_RESTAKE_GROWTH_TIME`.
        iv. `pullReward` is called by `sherlockCore` but then calls back into `sherlockCore` to get `totalTokenBalanceStakers` which can instead be passed as a parameter in the initial call.
        v. Local struct variables are only used to read and not to write anything which does not necessitate the storage data location specifier leading to SLOADs instead of cheaper MLOADs.
    c. **Github Permalinks**: `SherlockProtocolManager`, `SherDistributionManager`, `maxRewardArbTime - initialArbTime`, `pullReward`, local structs used for reading
    d. **Mitigation**: Caching state variables in local memory variables avoids unnecessary SLOADs during reads. Immutable/constant variables need not be unnecessarily copied to or derived from local variables. Avoid external calls where possible. Use memory instead of storage data location specifier for local variables where only reads are involved.

32. **Modifier checks not consistent and missing in some places**
    a. **Summary**: Modifiers should be used consistently where required.
    b. **Details**: Modifier `protocolExists` is not used consistently in some places where the underlying function `_verifyProtocolExists` is directly called instead. There are few places where this check is entirely missing.
    c. **Github Permalinks**:
        i. Function called: `SherlockProtocolManager.sol#L566`, `SherlockProtocolManager.sol#L704`, `SherlockProtocolManager.sol#L765`
        ii. Missing check: `nonStakersClaimable`
    d. **Mitigation**: Apply the modifier `protocolExists` instead of calling `_verifyProtocolExists` directly. Apply modifier to all places where protocol existence check is required.

33. **ClaimManager has no function to sweep funds**
    a. **Summary**: `SherlockClaimManager` does not implement a `sweep` function to withdraw funds accidentally sent to that contract.
    b. **Details**: Without a function to withdraw funds (tokens and ETH) that are accidentally sent to the `SherlockClaimManager` contract, they will be locked/lost forever. `SherlockProtocolManager`,

`SherlockDistributionManager` and `AaveV2Strategy` implement this functionality.

c. **Github Permalinks**: SherlockClaimManager, SherlockProtocolManager, SherlockDistributionManager, AaveV2Strategy

d. **Mitigation**: Add a `sweep` function to `SherlockClaimManager` similar to other managers. Similar to `_sweep` function, the `isActive` function could also be moved out of manager contracts into the common base `Manager`.

34. Code structure deviates from recommended best-practices

a. **Summary**: It is a best-practice to group ordering of function definitions by visibility and application logic.

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. Some functions deviate from this recommended best-practice.

i. `SherlockProtocolManager`: `view` functions above comments.

ii. `SherlockProtocolManager`: `pure` function between state modifying functions.

iii. `Sherlock`: `internal view` function between `external` functions.

iv. `SherlockClaimManager`: constructor in-between modifiers.

c. **Github Permalinks**:

i. SherlockProtocolManager.sol#L97-L137

ii. SherlockProtocolManager.sol#L364-L380

iii. Sherlock.sol#L567-L594

iv. SherlockClaimManager.sol#L96-L115

d. **Mitigation**: Consider reorganising functions as per the recommended best-practices.

35. `ABIEncoderV2` as experimental pragma

a. **Summary**: `ABIEncoderV2` no longer needs to be declared as `pragma experimental`.

b. **Details**: `ABIEncoderV2` was included as part of Solidity 0.8 and no longer needs to be declared as `pragma experimental`.

c. **Github Permalinks**:IAaveDistributionManager, IAaveIncentivesController, ILendingPool

d. **Mitigation**: Remove the declaration of `pragma experimental ABIEncoderV2`.

36. Two different pragmas for Solidity compiler are used in the project

a. **Summary**: While most contracts have a pragma of `0.8.10`, few use a floating `^0.8.0`.

b. **Details**: To be consistent and avoid using a floating pragma, all versions should be `0.8.10` or one of the previous versions that is not so recent.

c. **Github Permalinks**: OptimisticOracleInterface, OptimisticRequester, SkinnyOptimisticOracleInterface

d. **Mitigation**: Lock down all versions to `0.8.10` or one of the previous versions that is not so recent.

37. **USDC token as the single collateral may be risky**
    a. **Summary**: Sherlock uses USDC as its only reserve token.
    b. **Details**: USDC is a pausable and upgradable contract. Furthermore, Circle/Centre (USDC is managed by a consortium called Centre, which was founded by Circle) can blacklist USDC addresses which can permanently block funds of protocols/stakers/nonstakers or delay payments.
    c. **Github Permalinks**: All contracts.
    d. **Mitigation**: Short term: Inform users of this risk. Long term: Take measures to minimise this risk by incorporating other collateral tokens.
38. **Pausing the protocol does not provide a mechanism for stakers to emergency withdraw their staked amounts**
    a. **Summary**: If an attack is underway and contracts are paused as part of incident response, then there is no apparent mechanism to allow stakers to safely withdraw their staked amounts.
    b. **Details**: During incident response, the protocol owner may pause the `Manager` contracts, remove/update Managers via the main `Sherlock` contract and then do a sweep from the older/vulnerable Manager contracts. However, stakers' funds in the main `Sherlock` contract, which is not upgradeable, would be locked/lost.
    c. **Github Permalinks**: All contracts.
    d. **Mitigation**: Create an emergency withdrawal mechanism for stakers to withdraw their staked amounts when contracts are paused during an exploit incident response.
39. **External calls within a loop**
    a. **Summary**: External calls within an unbounded loop may lead to reverts from out of gas error or failing callbacks.
    b. **Details**: In `SherlockClaimManager`, `PreCorePayoutCallback` function is called on any external contracts registered in `claimCallbacks`. As commented, the intention of callbacks is to allow other contracts (e.g. reinsurers) to trigger payouts, etc. when Sherlock triggers one. Function `payoutClaim` loops over all the `claimCallbacks` on external contracts which could cause excessive gas usage leading to out of gas error if the number of callbacks is not bounded within the loop. Even if one of the callbacks fails, the entire transaction will revert.
    c. **Github Permalinks**: `payoutClaim`
    d. **Mitigation**: Evaluate gas usage and the number of callbacks that can typically be registered without leading to out of gas errors. Consider catching failing callbacks to prevent them from affecting execution of others.
40. **Missing equivalence check for new and old values**
    a. **Summary**: Setters should check if the new value being set is different from the old value being replaced.
    b. **Details**: `setMinActiveBalance` function does not check if the new `minActiveBalance` is being accidentally set to the same value as the old one.
    c. **Github Permalinks**: `setMinActiveBalance`
    d. **Mitigation**: Add a check to see if `_minActiveBalance` is the same as the current `minActiveBalance`.

41. **Implicit `virtual` keywords can be removed**
    a. **Summary**: Interface functions are implicitly virtual and therefore the `virtual` keyword can be removed.
    b. **Details**: `IGovernanceV2Helper.sol` is an interface (all functions are implicitly virtual) whose functions `getProposals`, `getProposal` and `getTokensPower` use the virtual keyword.
    c. **Github Permalinks**: `getProposals`, `getProposal`, `getTokensPower`
    d. **Mitigation**: Remove `virtual` keywords for interface functions.
42. **Test runs off a blockchain fork at a hard-coded block number**
    a. **Summary**: The project's testing framework allows getting snapshots of the state of the Ethereum mainnet with the help of Alchemy. The `AaveV2.js` test runs in the context of a (seemingly) arbitrarily hard-coded block number 13671132. As time passes, this stale blockchain state may become an invalid or inaccurate test scenario.
    b. **Details**: The relevance of the concern depends on (a) the mutability of the external components with which `AaveV2.js` generates interactions and (b) the time passed between running tests and mainnet deployment.
    The validity of tests degrades under the scenario where tests are successfully run and the contracts are deployed, but in the timeframe between block number 13671132 and time of deployment, the state of Aave/aUSDC pool changes in a way that would alter the outcome.
    c. **Github Permalinks**: `AaveV2.js:24`
    d. **Mitigation**: The concern with hard-coding the block number is that it's easy to be forgotten and become stale. Making the test run off the most recent block number can be done by querying a node. For example, via Etherscan.
    Even in the case that it is safe to assume external components won't change behaviour from block number 13671132 onwards, it is recommended to convert that magic number to a self-documenting constant name and a comment explaining the number choice (e.g. `TEST_BLOCKNUM`).
43. **Test code provided "without a battery"**
    a. **Summary**: The test harness provided with a project should be readily testable by reviewers and others.
    b. **Details**: The test process requires a `.env` file that provides an API key to Alchemy. Without this value, tests will fail.
    c. **Github Permalinks**: `snapshot.js:42`
    d. **Mitigation**: Add the `.env` file to the repo, (but make sure to delete your key). Add a note in the README about the need to set the API key.
44. **Missing visibility specifier**
    a. **Summary:** Missing visibility specifier for `nftCounter` variable relies on implicit behaviour in the Solidity compiler to make it internal.
    b. **Details:** Missing visibility specifier for `nftCounter` relies on implicit behaviour in the Solidity compiler to make it internal. This may be unintentional and should be either public or private. This is also true for many other state variables.
    c. **Github Permalinks:** `nftCounter`

d. **Mitigation:** Consider adding an explicit visibility specifier for all state variables to reduce ambiguity.

45. **Convention of using Solidity time scalers not consistent**
    a. **Summary:** Using Solidity's time scalers e.g. `days`, `hours` etc. is not consistently followed.
    b. **Details:** `LIVENESS` is set to 7200 to represent two hours. This is inconsistent with other time-related variable declarations that use built-in time scalers for clarity. Not using these scalers makes it more likely for mistakes to be made.
    c. **Github Permalinks:** `LIVENESS`
    d. **Mitigation**: Use `uint256 constant LIVENESS = 2 hours;`
46. **Unlimited approvals are risky**
    a. **Summary**: `AaveV2Strategy` approves Aave's lending pool to spend all its USDC balance.
    b. **Details**: In `AaveV2Strategy`'s `deposit` function, if Aave's lending pool does not have enough allowance to transfer the strategy contract's `want` token balance (USDC) into itself, the strategy contract sets maximum approval for the lending pool with `type(uint256).max` (code comment has a question asking whether they should only approve the necessary amount instead of the full amount). The strategy contract should not approve more than the necessary transfer amount. Even though Aave is a trusted ecosystem protocol, Sherlock should assume that any external contracts may become compromised.
    c. **GitHub Permalinks**: `AaveV2Strategy`
    d. **Mitigation**: Only approve the deposit amount, which is the strategy's current `want` token balance.
47. **`SafeERC20 safeApprove` risk**
    a. **Summary**: `SafeERC20 safeApprove` not only has the same race condition concern (arguably negligible when interacting with widely used/trusted contracts) as ERC20 `approve` but will also revert if allowance is != 0 or the amount being approved != 0.
    b. **Details**: According to OpenZeppelin's `SafeERC20` contract, `safeApprove` is deprecated due to front-running risk and its usage is discouraged. While the race-condition risk is arguably negligible, `AaveStrategy` uses `safeApprove` to approve Aave lending pool to transfer its `want` token balance when the current allowance is less than the contract's `want` token balance amount. If the current allowance is != 0 then `safeApprove` will revert.
    c. **GitHub Permalinks**: `safeApprove`
    d. **Mitigation**: use `safeIncreaseAllowance` instead.
48. **Incorrectly set protocol agent is hard to correct**
    a. **Summary**: If a protocol agent is set to the wrong address, there is no easy way to correct it.
    b. **Details**: In `SherlockProtocolManager`'s function `protocolAdd`, if a protocol agent is set to the wrong address, there is no easy way to correct it. The function `transferProtocolAgent` can only be called by the current protocol agent. So, by setting the wrong address, there is no way for the

insured protocol to submit a claim. The only way to update it is to call `protocolRemove` to remove the protocol and re-add it but, depending on when the error is noticed, that will have to reset the protocol's data and transfer back its active balance while the protocol may have already accrued debt in the meantime.

c. **GitHub Permalinks**: `_setProtocolAgent`

d. **Mitigation**: Consider having a two-step process for setting/claiming the protocol agent, where the new protocol agent has to claim the role. For the case where the protocol agent is set for the first time, allow the contract owner to reset the pending protocol agent as long as the role is not yet claimed. Alternatively, allow the contract owner to transfer the protocol agent role to another address directly.

49. Failing automated tests when run together

a. **Summary**: There are failing automated tests.

b. **Details**: `npx hardhat test` has 4 failing tests. However, when `npx hardhat test test/Sherlock.js` and `npx hardhat test test/SherlockProtocolManager.js` are run individually, the tests pass. There are likely dependencies between tests and that states are not properly cleaned up.

c. **GitHub Permalinks**: test, test, test and test

d. **Mitigation**: Investigate why tests are failing when they are run together and fix them.

50. Test readability and documentation

a. **Summary**: Test descriptions can be more descriptive.

b. **Details**: There are many test cases with descriptions such as `Do` and `Invalid` state which does not make it obvious as to what is being tested. Ideally, anyone should be able to look at the codebase, review test cases and get an overview of what the protocol/contracts/functions can or cannot do.

c. **GitHub Permalinks**: All test files.

d. **Mitigation**: Be more specific in terms of what each test aims to validate e.g. "protocol agent can call `startClaim` to initiate a claim process" or "protocol agent cannot submit a claim for more than the coverage amount." Test descriptions should be treated like documentation.

51. Unoptimised array removal

a. **Summary**: Removal of an array item can be optimised.

b. **Details**: While removing `claimCallbacks`, the last array item is copied over to the array index being removed, followed by an array pop to remove the last array item. However, when the index is equal to the last array item, this will unnecessarily copy over that same item (expensive SSTORE) followed by its removal.

c. **GitHub Permalinks**: `removeCallback`

d. **Mitigation**: An optimization is to check if the index is not equal to the last array item because that will avoid unnecessarily copying the same item to the same location.

52. Multiple potential addresses for tokens

a. **Summary**: USDC and `SHER` tokens are defined independently in different contracts.

b. **Details**: USDC and `SHER` tokens are defined independently in different contracts, and there is no check if they are the same across contracts. If any of the contracts is upgraded incorrectly, this could lead to a loss of funds.
c. **GitHub Permalinks**: Sherlock, AaveV2Strategy, SherDistributionManager, SherlockClaimManager, SherlockProtocolManager
d. **Mitigation**: Consider a common `Constants.sol` which defines all commonly used constants.

53. Unnecessary `nonReentrant` modifier

a. **Summary**: Use of `nonReentrant` modifiers on some functions is potentially unnecessary and can save gas.
b. **Details**: Functions that have no external calls to untrusted contracts and that are access controlled by trusted `onlyOwner`/`onlySPCC` addresses should not be at risk of re-entrancy (even cross-function) but they still have the `nonRentrant` modifier. In the best case, this affects code readability and causes gas wastage. In the worst case, this is an indication that the implementation is not what was originally intended.
c. **GitHub Permalinks**: addCallback, removeCallback, spccApprove, spccRefuse
d. **Mitigation**: Evaluate the trust model and the use of `nonRentrant` modifier on such functions with no external calls or with access control.

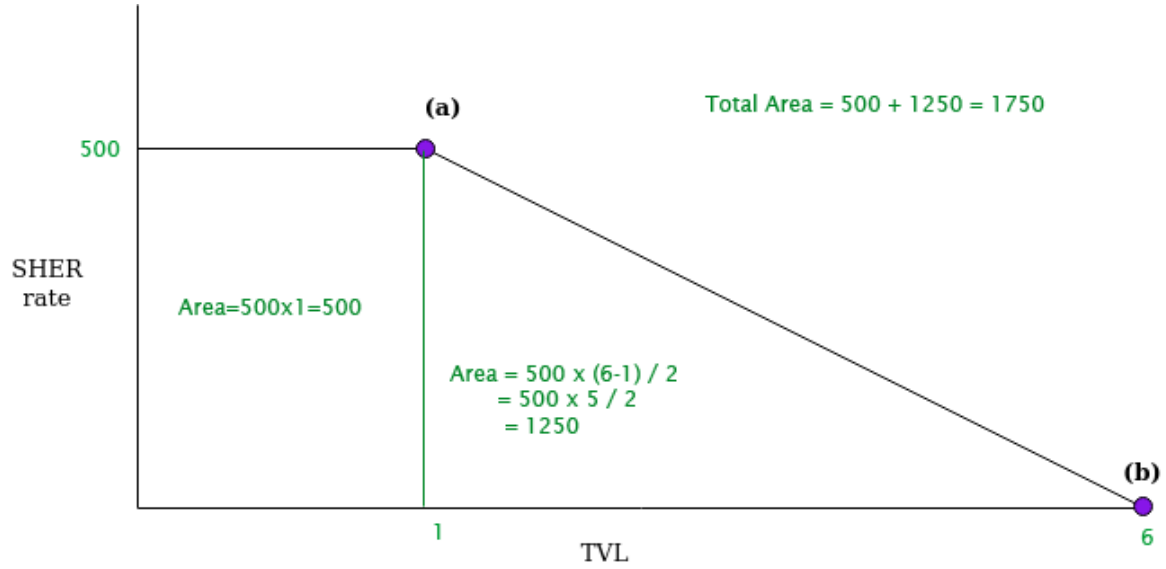
54. `removeCallback` can be optimised for UX

a. **Summary:** `removeCallback` expects the caller to know the index of the callback in the array data structure.
b. **Details:** `removeCallback` expects the caller (i.e. owner) to know the index of the callback being removed in the array data structure. Given that this changes with every removal, the owner is expected to keep track of the array order or check via the public getter before calling this function. This is a sub-optimal user experience.
c. **GitHub Permalinks:** removeCallback
d. **Mitigation:** Consider searching for the callback contract (to be removed) by looping the `claimCallbacks` array instead of accessing via index.

55. Slope rewards are incorrectly calculated for deposits that overlap the entire slope into the max rewards zone

a. **Summary**: In `sherDistributionManager.calcReward`, the variable `slopeRewardsAvailable` is calculated by subtracting the TVL from the end of the reward zone, but does not take account of the case where TVL > the entire horizontal length of the slope. This results in smaller rewards than those promised in the specification. The tests are also misconfigured to approve the incorrect result returned by this.
b. **Details**: Consider the simplest test that spans across all price ranges: Max rewards run from 0-1 and the slope runs from 1-5. The `SHER` rate is 500 and the period is just 1 and so can be ignored. There is no pre-existing TVL.
The expected rewards would be, as in the diagram below, 1750, but the tests starting line 95 of `SherDistributionManager.js` are expecting, and receiving, a value of 1700 (later tests repeat this error by being multiples of 1700, not 1750).
On line 107 of `SherDistributionManager.sol`, `slopeRewardsAvailable` is being set to 6, when it should be 5 because it is not considering the max rewards zone. This is causing position on line 137

to be 3 instead of 2.5 (actual numbers are magnitudes higher so decimals do work) and so when position is subtracted on line 145, it leads to a lower `SHER` reward.



c. **Github Permalinks**: `slopeRewardsAvailable`, `calcReward`, Test

d. **Mitigation**: Take into account the fact that slope rewards only start at `maxRewardsEndTVL`. Changing line 107 as shown fixes the output:

```
uint256 slopeRewardsAvailable = maxRewardsEndTVL_ > _tvl
? (zeroRewardsStartTVL_ - maxRewardsEndTVL_) :
(zeroRewardsStartTVL_ > _tvl ? zeroRewardsStartTVL_ -
_tvl : 0);
```

56. A corrupt/compromised `DEPLOYER` can steal a deposit to the strategy or siphon `SHER` from `SherDistributionManager`

a. **Summary**: If `DEPLOYER` does not set a Sherlock Core address after deployment, it can allow an initial deposit to the strategy manager and then call `setSherlockCoreAddress` to `withdrawAll` to an attacker controlled version of Sherlock Core. Or, it can allow an initial deposit to `SherDistributionManager` and then call `setSherlockCoreAddress` with an attacker controlled version of Sherlock Core.

b. **Details**: See exploit script. The main attack portion starts on line 128. Because it is running on Hardhat, all addresses are effectively `DEPLOYER` in this script. It is therefore just showing that one Sherlock Core will deposit USDC and another Sherlock Core can be set and then `withdrawAll()`. See exploit script. The main attack portion starts on line 43. As this script demonstrates, even if `SherDistributionManager` has a trusted owner, that owner cannot sweep the `SHER` to safety and the attacker controlled Sherlock Core can use `pullRewards` to siphon out the `SHER`.

c. **Github Permalinks**: `Manager`

d. **Mitigation**: `Manager` could have a modifier that reverts if `address(sherlockCore) == address(0)` and this could be applied to all state changing functions.

57. Insufficient verification between Sherlock Core and Managers

a. **Summary**: Manager contracts will perform actions without `setSherlockCoreAddress` having been called and `SherlockCore` will call Manager contracts that do not have it set as their `sherlockCore`.
b. **Details**: Even without a corrupt `DEPLOYER`, funds can still be locked if `sherlockCore` is accidentally misconfigured (see demo).
c. **Github Permalinks**: `Manager`
d. **Mitigation**: `Manager` could have a modifier that reverts if `address(sherlockCore) == address(0)` and this could be applied to all state changing functions. Sherlock Core could be deployed with `CREATE2` to a predetermined address and the managers could all be preconfigured to have that address set. Then Sherlock Core could handshake with all of them in its constructor.

58. Unnecessary calculation

a. **Summary**: Unnecessary calculation when `debt` and `balance` are equal.
b. **Details**: In `SherlockProtocolManager._activeBalance`, we can return zero if `debt` and `balance` are equal. There is no need to perform the additional calculation which will save gas in this case.
c. **Github Permalinks**: `_activeBalance`
d. **Mitigation**: Change '`>`' to '`>=`' on line 218

59. Constant is defined as a function in interface

a. **Summary**: Constant `UMA_IDENTIFIER` is defined as a function in its interface.
b. **Details**: `SherlockClaimManager.UMA_IDENTIFIER` is a constant but is defined as a function in its interface `ISherlockClaimManager.sol`. The constant is only used as a constant.
c. **Github Permalinks**: `ISherlockClaimManager`
d. **Mitigation**: Update the interface to remove the function and remove the `override` keyword from the constant declaration in `SherlockClaimManager`.

60. Naming convention of constants

a. **Summary**: Constant naming convention is all upper case.
b. **Details**: `AaveV2Strategy.lpAddressProvider` is a constant and should be in `UPPER_CASE_WITH_UNDERSCORES` as per Solidity Style Guide.
c. **Github Permalinks**: `lpAddressProvider`
d. **Mitigation**: Rename the constant to all uppercase.

61. Interface validation on addresses

a. **Summary**: Functions taking contract address parameters could check that addresses do indeed contain code that implement expected interfaces.
b. **Details**: `updateSherDistributionManager`, `updateSherlockProtocolManager`, `updateSherlockClaimManager` and `updateYieldStrategy` could check their address parameters to ensure they contain code that implement expected interfaces.
c. **Github Permalinks**: `Sherlock`
d. **Mitigation**: Check that contract address parameters do contain code that implement expected interfaces.

62. **Cannot `sweep` if receiver cannot receive ETH**
    a. **Summary**: In `Manager._sweep`, if `_receiver` address cannot receive ETH, `_sweep` will revert even if no ETH is being sent.
    b. **Details**: If `_receiver` address cannot receive ETH, `_sweep` will revert even if no ETH is being sent which will block attempts to move tokens out to contracts that cannot receive ETH but can handle tokens. It also wastes gas to make this external call when no ETH is being sent.
    c. **Github Permalinks**: `_sweep`
    d. **Mitigation**: Check if `this.balance > 0` before making the call.
63. **`Sherlock._verifyUnlockableByOwner` always returns `msg.sender`**
    a. **Summary**: `Sherlock._verifyUnlockableByOwner` either returns `msg.sender` or reverts. So the return value is not capturing anything different.
    b. **Details**: There is no need to create an additional named return variable `nftOwner` in `redeemNFT()` as this will always be `msg.sender`. It reduces readability and costs gas.
    c. **Github Permalinks**: `_verifyUnlockableByOwner`, `redeemNFT`
    d. **Mitigation**: Remove the return value of `_verifyUnlockableByOwner` and use `msg.sender` at the caller.
64. **Unrestricted access to `AaveStrategyV2.deposit`**
    a. **Summary**: Functions need to have restrictive access control.
    b. **Details**: `AaveStrategyV2.deposit` deposits the full balance of USDC held in `AaveV2Strategy` contract into Aave's lending pool and also changes the allowance of USDC for the contract. There is no obvious reason or incentive to grant this function's access to all external actors as this is designed to be called only by the Sherlock Core contract.
    c. **Github Permalinks**: `AaveV2Strategy`
    d. **Mitigation**: Consider restricting access to `AaveStrategyV2.deposit` using `onlySherlockCore`.
65. **`SherlockClaimManager.escalate` transfers/approves more than necessary**
    a. **Summary**: `SherlockClaimManager` claims that the bond amount is approved, but actually `_amount` can be more than this.
    b. **Details**: There is no reason to transfer the potentially larger `_amount` or approve this larger value, and then transfer the leftover back to the sender, because that should not be necessary.
    c. **Github Permalinks**: `SherlockClaimManager`
    d. **Mitigation**: `_amount` could be removed from the function entirely or, if that is not desirable for user experience reasons, then `_amount` could be checked against `BOND` and `BOND` could be used.
66. **Missing `_nonStakers` percentage input validation**
    a. **Summary**: Perform input validation for non-zero nonstakers percentage of premiums, unless this can be zero.

b. **Details**: In the case that percentage of premiums can be 0%, it should be specified. If not, perform input validation for a non-zero percentage of premiums.
c. **Github Permalinks**: `protocolUpdate`
d. **Mitigation**: Perform input validation for non-zero nonstakers percentage of premiums or specify if that is allowed.

67. **Avoid hardcoded values**
    a. **Summary**: There are hardcoded addresses or identifiers.
    b. **Details**: There are hardcoded addresses or identifiers that may be better initialised via the constructor. This helps with integration tests, testing in testnet deployment and also allows better configuration.
    c. **Github Permalinks**: `UMA_IDENTIFIER`, `UMA`, `TOKEN`, `DEPLOYER`
    d. **Mitigation**: Change them from constant to immutable and set their values in the constructor.
68. **Missing input validation of `_tokenID`**
    a. **Summary**: `tokenBalanceOf` is missing input validation of its parameter `_tokenID`.
    b. **Details**: Functions `lockupEnd` and `sherRewards` that like `tokenBalanceOf` take a parameter `_tokenID` perform input validation using `_exists` to check if `_tokenID` exists. However, `tokenBalanceOf` is missing this validation.
    c. **Github Permalinks**: `tokenBalanceOf`, `lockupEnd`, `sherRewards`
    d. **Mitigation**: Add input validation using `_exists`.
69. **Missing cleanup of data structures**
    a. **Summary**: Protocol data structure is not cleaned up on removal.
    b. **Details**: When a protocol is removed from the system, its data structures are cleaned up. However, there is on data structure that does not appear to be cleaned up which may cause stale values to persist and affect accounting if/when the protocol is added again later. Specifically, the mapping of `nonStakersClaimableByProtocol` is not cleaned up.
    c. **Github Permalinks**: `nonStakersClaimableByProtocol`, `_forceRemoveProtocol`
    d. **Mitigation**: Clean up all protocol data structures as soon as a protocol is removed.
70. **Backrunning can prevent new protocols from funding positions**
    a. **Summary**: An attacker can grief protocol addition by forcing its removal before its balance is deposited.
    b. **Details**:The intended order of functions for adding a new protocol is: `protocolAdd`—`depositToActiveBalance`—`setProtocolPremium`. Immediately after `protocolAdd`, an attacker could grief by calling `forceRemoveByActiveBalance` and removing the protocol's coverage position and agent. This will cause DoS for the protocol by preventing it from being added to Sherlock.
    c. **Github Permalinks**: `protocolAdd`, `depositToActiveBalance`, `setProtocolPremium`, `forceRemoveByActiveBalance`

d. **Mitigation**: Document that `protocolAdd` and `depositToActiveBalance` should be executed in a single transaction.

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.