Quantstamp Security Assessment Certificate

September 10th 2021 — Quantstamp Verified

Sherlock

This audit report was prepared by Quantstamp, the leading blockchain security company.

Executive Summary

Type	Insurance Pools
Auditors	Jose Ignacio Orlicki, Senior Engineer Ed Zulkoski, Senior Security Engineer Sebastian Banescu, Senior Research Engineer
Timeline	2021-07-13 through 2021-09-10
EVM	Berlin
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Sherlock Gitbook Test Site
Documentation Quality	High
Test Quality	High

Source Code

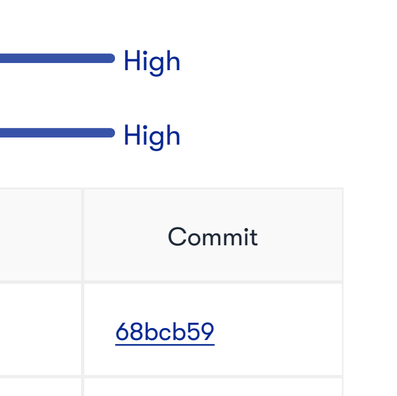
Repository	Commit
sherlock-v1-core	68bcb59
sherlock-v1-core	c9aeaf5 (reaudit)

Goals

• Find issues that could allow attackers to drain user assets.

• Find issues that could lead to fund losses or freeze.

Total Issues	25 (9 Resolved)
High Risk Issues	1 (1 Resolved)
Medium Risk Issues	5 (1 Resolved)
Low Risk Issues	10 (5 Resolved)
Informational Risk Issues	8 (2 Resolved)
Undetermined Risk Issues	1 (0 Resolved)



High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
Undetermined	The impact of the issue is uncertain.
Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
Acknowledged	The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).
Resolved	Adjusted program implementation, requirements or constraints to eliminate the risk.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

We have reviewed the code, documentation, and test suite and found several issues of various severities. Overall, we consider the code to be well-written and with an extensive testing suite, but we suggest adding more inline comments and further tests reflecting current issues displayed. The test suite also includes a gas analysis suite, something that is not very common but highly recommended. We also provide suggestions for improvements to follow the best practices. We recommend addressing all the **26** findings and the rest of the suggestions to harden the contracts for future deployments or contract updates. We recommend against deploying the code as-is.
Update: Quantstamp has audited the changes based on the diffs for the `sherlock-v1-core` repository (4de2ba1...c9aeaf5). Of the original 26 issues, 25 have been either fixed, acknowledged, or mitigated; 1 low impact issue has been removed as a non-issue in further discussions.

ID	Description	Severity	Status
QSP-1	Last Parameter Can Be Spoofed	High	Fixed
QSP-2	Adding/Updating Protocols Should Be Timelocked	Medium	Acknowledged
QSP-3	Insufficient Buffer For Price Fluctuations	Medium	Acknowledged
QSP-4	Protocol May Overpay Premium Fee	Medium	Acknowledged
QSP-5	First Money Out Pool Not Emptied First	Medium	Acknowledged
QSP-6	Non-empty Strategies Can Be Removed Without Sweeping	Medium	Fixed
QSP-7	Privileged Roles	Low	Acknowledged
QSP-8	Unchecked Assumption In `Payout`	Low	Fixed
QSP-9	Unchecked Assumption In `SherX`	Low	Acknowledged
QSP-10	Ignored Return Values	Low	Mitigated
QSP-11	Staker Funds Are Reinvested On Other DeFi Platforms	Low	Acknowledged
QSP-12	Protocol Premium Change Issue	Low	Acknowledged
QSP-13	Unlocked Dependencies	Low	Mitigated
QSP-14	Missing Input Validation	Low	Acknowledged
QSP-15	Gas Concerns	Low	Mitigated
QSP-16	Integer Overflow	Low	Fixed
QSP-17	Transfers of Zero Coins Allowed	Informational	Acknowledged
QSP-18	Cooldown Period Could Be Excessively Long	Informational	Acknowledged
QSP-19	Unlocked Pragma and Different Solidity Versions	Informational	Fixed
QSP-20	Stakeholder Incentives Are Not Fully Aligned	Informational	Acknowledged
QSP-21	Important Contract Operations Cannot Be Easily Monitored	Informational	Acknowledged
QSP-22	Allowance Double-Spend Exploit	Informational	Acknowledged
QSP-23	`initializeSherXERC20` May Be Called Multiple Times	Informational	Fixed
QSP-24	Unused Protocol Manager and Agent Variables	Informational	Acknowledged
QSP-25	Strong Dependency On Block Numbers For Financials	Undetermined	Acknowledged

Quantstamp Audit Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.8.0
- Mythril 0.22.8

Steps taken to run the tools:

Installed the Slither tool: `pip install slither-analyzer` Run Slither from the project directory: `slither .` Installed the Mythril tool from Pypi: `pip3 install mythril` Ran the Mythril tool on each contract: `myth analyze FlattenedContract.sol`

Findings

QSP-1 Last Parameter Can Be Spoofed

Severity: *High Risk*

Status: Fixed

File(s) affected: `PoolBase.sol`, `PoolOpen.sol`, `PoolStrategy.sol`

Description: The following functions contain parameters, which are not used:

1. The `_token` parameter in the `getCooldownFee` function from `PoolBase.sol`
2. The `_token` parameter in the `getSherXWeight` function from `PoolBase.sol`
3. The `_token` parameter in the `getGovPool` function from `PoolBase.sol`
4. The `_token` parameter in the `isPremium` function from `PoolBase.sol`
5. The `_token` parameter in the `isStake` function from `PoolBase.sol`
6. The `_token` parameter in the `getProtocolBalance` function from `PoolBase.sol`

7. The `_token` parameter in the `getProtocolPremium` function from `PoolBase.sol`
8. The `_token` parameter in the `getLockToken` function from `PoolBase.sol`
9. The `_token` parameter in the `isProtocol` function from `PoolBase.sol`
10. The `_token` parameter in the `getProtocols` function from `PoolBase.sol`
11. The `_token` parameter in the `getUnstakeEntry` function from `PoolBase.sol`
12. The `_token` parameter in the `getFirstMoneyOut` function from `PoolBase.sol`
13. The `_token` parameter in the `getTotalPremiumPerBlock` function from `PoolBase.sol`
14. The `_token` parameter in the `getPremiumLastPaid` function from `PoolBase.sol`
15. The `_token` parameter in the `getSherXUnderlying` function from `PoolBase.sol`
16. The `_token` parameter in the `getUnstakeEntrySize` function from `PoolBase.sol`
17. The `_token` parameter in the `getInitialUnstakeEntry` function from `PoolBase.sol`
18. The `_token` parameter in the `getUnactivatedStakersPoolBalance` function from `PoolBase.sol`
19. The `_token` parameter in the `getStakersPoolBalance` function from `PoolBase.sol`
20. The `_token` parameter in the `getUnallocatedSherXStored` function from `PoolBase.sol`
21. The `_token` parameter in the `getTotalSherXPerBlock` function from `PoolBase.sol`
22. The `_token` parameter in the `getSherXLastAccrued` function from `PoolBase.sol`
23. The `_token` parameter in the `LockToToken` function from `PoolBase.sol`
24. The `_token` parameter in the `TokenToLock` function from `PoolBase.sol`
25. The `_token` parameter in the `setCooldownFee` function from `PoolBase.sol`
26. The `_token` parameter in the `activateCooldown` function from `PoolBase.sol`
27. The `_token` parameter in the `cancelCooldown` function from `PoolBase.sol`
28. The `_token` parameter in the `cancelCooldown` function from `PoolBase.sol`
29. The `_token` parameter in the `unstakeWindowExpiry` function from `PoolBase.sol`.

Additionally, there are several functions in `PoolBase.sol`, which use both `_token` and `baseData()`. The latter uses the `bps()`, which is used to retrieve the value of `_token`. This function is overly complex containing inline assembly and assuming that the last parameter of the function where `bps()` is called, is always the `_token`. An attacker could break this assumption by simply passing an arbitrary token when calling any of the aforementioned functions (see enumeration above).
The `bps()` function is also cloned in `PoolOpen.sol` and `PoolStrategy.sol`. Same recommendation applies to those contracts.

Recommendation: Remove the `bps()` function and use the `_token` input parameter instead, e.g.: `function baseData(IERC20 _token) internal view returns (PoolStorage.Base storage ps) { ps = PoolStorage.ps(address(_token)); require(ps.govPool != address(0), 'INVALID_TOKEN'); }` Refactor the related `baseData` function.

Update: Fixed in this PR.

QSP-2 Adding/Updating Protocols Should Be Timelocked

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `Gov.sol`

Description: Staker funds are used as support for all protocols supported by Sherlock. Therefore, adding a new protocol may introduce a risk that one or more *stakers* are not comfortable with. For example, a *staker* might know the team behind a certain protocol that is newly added and believe that the risk of a hack of that protocol is too high to tolerate. Therefore, *stakers* should get a chance to withdraw their funds before a protocol starts being covered by Sherlock.

Recommendation: Add a timelock mechanism for adding and updating protocols such that the *stakers* have ample time to react if needed. It is important to note that *stakers* need to go through a cooldown period before they can *unstake*. Therefore, the timelock period for adding a new protocol should be at least the same length as the cooldown period and ideally, it should be greater or equal to the cooldown period plus the *unstake* period.

Update: Acknowledged by Sherlock team. The detailed response was "*Might implement later. I think the same logic applies to* `updateSolution`*, this should ideally also be time locked as it can be updated to drain all users' funds. For now we will signal the new protocol off chain and make sure the stakers have enough time to unstake.*".

QSP-3 Insufficient Buffer For Price Fluctuations

Severity: *Medium Risk*

Status: Acknowledged

Description: The documentation states that:

> Sherlock implements a buffer amount (20% currently) to account for currency fluctuations. So if Sherlock will cover a protocol for $100M, the staking pools need to be at least $120M in size ($100M we need to cover + 20% buffer).

However, given that we have seen price drops of more than 50% in the past couple of months, this 20% buffer seems insufficient.

Recommendation: Increase the buffer to 50%.

Update: Acknowledged by Sherlock team. The detailed response was "*Will update documentation with "The 20% buffer is based on the ability of protocols to deposit funds within 24 hours, this allows for a XX% drop in price. If the mechanism are not in place, we require a bigger buffer*"".

QSP-4 Protocol May Overpay Premium Fee

Severity: *Medium Risk*

Status: Acknowledged

Description: The Sherlock pool must hold at least the amount necessary to payout the largest protocol, plus a 20% buffer. However, if a large *staker* decides to withdraw their stake at some point and the size of the pool decreases below the amount needed by one or more of the largest platforms, then those platforms will overpay the amount of premium from that point onward until either the premium amount is adjusted by the governance or sufficient funds are staked to restore the pool size to the amount needed to cover the largest platforms.

Recommendation: Automate the process of premium fee adjustment in case the size of the pool goes down.

Update: Acknowledged by Sherlock team. The detailed response was "*We might include this in a redesign. For now; If the pool is 25m$ we only cover protocol up until 15m$ allowing for a large withdrawal of stakers. + We can see the withdrawal of stakers coming as they are subject to a cooldown period. This gives us enough room to operate on a mitigation plan*".

QSP-5 First Money Out Pool Not Emptied First

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `Payout.sol`

Description: When calling the `Payout.payout` function the governance address can choose the amount that is to be transferred from the `firstMoneyOut` pool for each token. This goes against the specification, which indicates that the `firstMoneyOut` pool will be emptied first in case of a valid claim.

Recommendation: Remove the `_firstMoneyOut` input parameter from the `payout` function and only leave the `_amounts`. The First Money Out pool should be emptied and the remaining amount should be subtracted from `stakeBalance`.

Update: Acknowledged by Sherlock team. The detailed response was "*There are still some unknowns around the exact tokenomics. Leaving this freedom to the calling account gives more freedom to execute on a payout.*".

QSP-6 Non-empty Strategies Can Be Removed Without Sweeping

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `PoolStrategy.sol`

Description: As described in one comment of `strategyRemove()` that says that this function `don't check if the current strategy balance = 0`, in case of faulty strategies there can be financial loss of strategy has tokens.

Recommendation: Including sweeping functions to save remaining ETH or ERC20 tokens on a faulty strategy before inside the call to `strategyRemove()`.

Update: Fixed in this PR.

QSP-7 Privileged Roles

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `Gov.sol`, `PoolDevOnly.sol`, `GovDev.sol`

Description:

1. From the protocol overview: "Our claims committee will be the final arbiters in deciding whether a certain hack falls under coverage and therefore should be paid out."
2. Within `Gov.sol`, several protocol parameters may be changed, e.g., `unstakeWindow`.
3. The owner is responsible for setting token prices and protocol premiums. **No oracles are used for pricing.**
4. The diamond pattern would allow the owner to add arbitrary code to the solution as facets via `GovDev.updateSolution`.

Also, other privileged roles should be documented, which can perform a series of actions that could affect end-users:

- The main governance role can do the following:
  - Change the address of the main governance role at any time.
  - Change the Watson's address at any time.
  - Set the length of the unstake window to any value under 10 years, at any time.
  - Set the length of the cooldown period to any value under 10 years, at any time.
  - Add new protocols at any time.
  - Update the protocol agent at any time.
  - Add any ERC20 tokens the protocol is allowed to pay in at any time.
  - Remove protocols that do not have debt.
  - Initialize a new token.
  - Disable any of the added ERC20 tokens with 0 active weight, for staking.
  - Disable any covered protocol if the active premium and active underlying are 0.
  - Sell/swap ERC20 tokens from the pool at any time.
  - Remove ERC20 tokens from the pool if they are not used by stakers or protocols.
  - Set the price of any ERC20 token in USD at any time.
  - Set the premium amounts for a protocol using multiple ERC20 tokens.
  - Change the address of the payout governance, at any time.
  - Set initial SHERX distribution to Watsons, once.
  - Set the SHERX distribution (weights) to different ERC20 tokens at any time.
- The GovDev role can do the following:

-
  - Set the initial main governance address, once.
  - Change the GovDev address, at any time.
  - Delete, update or add functions to the Sherlock smart contract.
  - Set the initial governance payout address, once.
  - Initialize the SherXERC20 token, once.
  - Stake in the `PoolDevOnly` contract. No other role can stake in this pool.

Recommendation: All these privileged actions and their consequences should be made clear via end-user-facing documentation.

Update: Acknowledged by Sherlock team. The detailed response was "*Will be documented*".

QSP-8 Unchecked Assumption In `Payout`

Severity: *Low Risk*

Status: Fixed

File(s) affected: `Payout.sol`

Description: The `LibPool.payOffDebtAll(tokens[i])` function is called on L80 in `Payout.sol` if `amounts[i] > ps.sherXUnderlying`. The assumption here is that the following condition will hold after this function call: `amounts[i] <= ps.sherXUnderlying`. However, there is no proof or clear guarantee that it will hold. This is partially handled by the .sub in the else branch below (ps.sherXUnderlying = ps.sherXUnderlying.sub(amounts[i]);), but missed in the if-branch.

Recommendation: Add an `assert(amounts[i] <= ps.sherXUnderlying)` on L81 inside the `if`-statement after the function call, to verify the assumption holds.

Update: Fixed in this PR.

QSP-9 Unchecked Assumption In `SherX`

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `SherX.sol`

Description: The `SherX.doYield` function allows the caller to specify an `amount` of tokens that should be transferred from the `from` address to the `to` address. However, there is no explicit check inside this function, which ensures that the `amount <= ps.lockToken.balanceOf(from)`. This could lead to an inappropriate amount of tokens being used for computations of the `ineglible_yield_amount`.

Recommendation: Check if `amount > ps.lockToken.balanceOf(from)` and if so the value of the `amount` should be capped to `ps.lockToken.balanceOf(from)` or the execution should throw.

Update: Acknowledged by Sherlock team. The detailed response was "*Calling functions already take care of this. It's zero in case of harvest() call. And between 0 and user balance if called from token transfer.*".

QSP-10 Ignored Return Values

Severity: *Low Risk*

Status: Mitigated

File(s) affected: `AaveV2.sol`

Description: The `AaveV2.withdrawAll` and `AaveV2.withdraw` functions call the Aave lending pool `withdraw` function which returns the final amount withdrawn. Since this is an external contract and we have seen several past attacks where the vulnerable code has made implicit assumptions on calls to external contracts without explicitly checking them, we believe that this poses a risk in case the amount withdrawn from Aave would be much lower than the expected amount.

Recommendation: Check the return values of functions from external contracts and throw or handle appropriately if the result is not as expected.

Update: Mitigated in this PR. Not fixed for the `withdrawAll()` case.

QSP-11 Staker Funds Are Reinvested On Other DeFi Platforms

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `PoolStrategy.sol`

Description: The funds deposited by *stakers* with the purpose of covering platforms in case of a hack are reinvested on other DeFi platforms to increase *staker* yield. This puts the reinvested funds at risk of getting stolen by any hacks or insolvency of the DeFi platforms that are used for investing. At the moment the only platform that is used is Aave and therefore the risk is Low. However, if other (less secure) platforms will be used in the future, then the risk might increase.

Recommendation: Do not place stakeholder funds at risk by depositing them in other DeFi platforms.

Update: Acknowledged by Sherlock team. The detailed response was "*Aave uses the safety module. Protects user funds.*".

QSP-12 Protocol Premium Change Issue

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `Manager.sol`

Description: The `Manager.setProtocolPremium` and `Manager.setProtocolPremiumAndTokenPrice` functions are used to change the premium amount for one or more protocols. These functions all make a call to `LibPool.payOffDebtAll` which pays off the accrued debt of all whitelisted protocols that use a specific token. However, if any of these protocols do not have sufficient balance for the debt to be paid off, it will cause the whole transaction to revert and the protocol premium price will remain unchanged (for any unrelated protocol).

Exploit Scenario: Assuming there are 2 protocols A & B both paying the premium using the same ERC20 token. Protocol B also pays a large percentage of the premium using another ERC20 token that is stable and hence the premium for protocol B does not need to change:

1. Protocol A needs a mid-week premium change because the price of the token has fallen.
2. The governance account calls `Manager.setProtocolPremium`.
3. Protocol B has insufficient balance to pay off their debt and the whole transaction is reverted meaning the premium for protocol A cannot be changed unless protocol B is removed.

Recommendation: This dependency between protocols when it comes to changing the premium amount should be eliminated such that premium amounts can be changed independently of whether any other protocol has sufficient balance to pay off their debt. Remove the calls to `LibPool.payOffDebtAll` inside of both `Manager.setTokenPrice` functions.

Update: Acknowledged by Sherlock team.

QSP-13 Unlocked Dependencies

Severity: *Low Risk*

Status: Mitigated

File(s) affected: `package.json`

Description: All dependency versions inside `package.json` are unlocked due to `^` before the version number. Moreover, the core dependency, namely the `diamond-2 repo` is specified without any version number, which means that the latest commit from the `master` branch will be used when calling `yarn install`. Such a commit might be insecure or unstable since it is not necessarily a release commit.

Recommendation: Fix all dependency versions to stable release versions.

Update: Mitigated in this PR. Some dependencies are still unlocked.

QSP-14 Missing Input Validation

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `GovDev.sol`, `Payment.sol`, `PoolBase.sol`

Description: The following functions are missing input parameter validation:

1. The `GovDev.transferGovDev()` function does not check if the `_govDev` parameter is different from `address(0)`.
2. The `Payment.payment()` method does not check if the same token address is provided twice in the `_tokens` input array parameter.
3. The `_amounts` and `_firstMoneyOut` values are not checked to be less than the available amounts in the pools. There is an implicit check by using SafeMath functions which will fail if these conditions are false, but will not provide a proper error message.
4. The `_amount` input parameter of the `PoolBase.withdrawProtocolBalance` function is not explicitly checked to be less than `ps.protocolBalance[_protocol]`. This could lead to an error in the `sub()` function call on L292, without any error message.
5. The `PoolBase.setCooldownFee` function does not impose an upper limit on the `_fee` parameter value and this fee can be changed at any time by the governance account.

Recommendation: Add input parameter validation to each of the functions indicated in the enumeration above.

Update: A case was fixed in this PR. The other cases were acknowledged.

QSP-15 Gas Concerns

Severity: *Low Risk*

Status: Mitigated

File(s) affected: `Gov.sol`, `Manager.sol`, `Payout.sol`, `PoolBase.sol`, `SherX.sol`, `LibPool.sol`, `LibSherX.sol`

Description: The following smart contract functions contain loops and could lead to an out-of-gas error if the number of supported ERC20 tokens and/or protocols by Sherlock is too large:

1. `Gov.protocolRemove` loops over the number of ERC20 tokens underlying SHERX.
2. `Payout._doSherX` loops over the number of ERC20 tokens underlying SHERX.
3. `SherX.calcUnderlyingInStoredUSD` loops over the number of ERC20 tokens underlying SHERX.
4. `SherX.harvestFor` loops over the number of ERC20 tokens stakers are allowed to stake in.
5. `SherX.setInitialWeight` loops over the number of ERC20 tokens stakers are allowed to stake in.
6. `SherX.redeem` loops over the number of ERC20 tokens stakers are allowed to stake in.
7. `LibPool.payOffDebtAll` loops over the number of protocols registered in the corresponding pool.
8. `LibSherX.calcUnderlying` loops over the number of ERC20 tokens underlying SHERX.
9. `LibSherX.accrueSherX` loops over the number of ERC20 tokens stakers are allowed to stake in.
10. `Manager.setProtocolPremium` loops over many protocols and tokens. In particular, because is a double loop, the inner call to `LibPool.payOffDebtAll(_token[i][j]);` may be invoked many times for the same token.

Note that these are not the only functions with loops. There is a large number of functions with loops in the code base, where the loop boundary is given by the length of one of the input arrays to the corresponding function. Since those functions are external functions we assume that the caller will be notified by their wallet software in case the length of the input array is too large. However, all functions with loops should be in scope for this issue.

Recommendation: Enforce a maximum number of:

1. ERC20 tokens underlying SHERX.
2. ERC20 tokens stakers are allowed to stake in.
3. Protocols registered in a pool.

Such that none of the aforementioned functions would lead to an out-of-gas error.

Update: Mitigated in this PR. Still not fully confirmed that 256 tokens and pools do not lead to an out-of-gas error.

QSP-16 Integer Overflow

Severity: *Low Risk*

Status: Fixed

File(s) affected: `SherX.sol`

Description: The `_watsons` and `_weights` input parameters of the `SherX.setWeights` function are `uint256` and `uint256[]` respectively. However, both input parameters are cast to `uint16` on L221 and L215, respectively. This can cause an integer overflow/truncation of the actual value passed in as input.

Recommendation: Change the input parameter types to `uint16` and remove the casts.

Update: Fixed by Sherlock team. The detailed response was *"Changing to uint16 doesn't allow to check for uint256.max, this variable is used to not change the watsons address."*.

QSP-17 Transfers of Zero Coins Allowed

Severity: *Informational*

Status: Acknowledged

File(s) affected: `SherXERC20.sol`

Description: Functions `transferFrom()` and `transfer()` of `SherXERC20` allows for a valid transfer of a zero amount of token from any account to any account. This has not a severe impact but can generate false alerts on transferences from wallets, pollute logging systems, and generate wash trading to collect possible future rewards. This is allowed by the ERC20 specification. From EIP-20: "Note Transfers of 0 values MUST be treated as normal transfers and fire the Transfer event."

Update: Acknowledged by Sherlock team.

QSP-18 Cooldown Period Could Be Excessively Long

Severity: *Informational*

Status: Acknowledged

File(s) affected: `Gov.sol`

Description: The governance can set the cooldown period to 25 million blocks, which (assuming a block time of 13 seconds) is over 10 years. This is an excessive amount of time for stakers to wait before they can withdraw their funds.

Recommendation: The maximum cooldown period should be set to a reasonable amount of time.

Update: Acknowledged by Sherlock team.

QSP-19 Unlocked Pragma and Different Solidity Versions

Severity: *Informational*

Status: Fixed

File(s) affected: `all contracts except for IAaveDistributionManager.sol and IAveIncentivesController.sol`

Description: All affected contracts have an unlocked pragma, but removing the `^` is not sufficient to fix the problem, because different files use different versions. The following versions were observed:

- 0.7.0
- 0.7.1
- 0.7.4
- 0.7.6

Recommendation: Either use the latest version from the list of versions in the description (i.e., 0.7.6) or use one of the latest versions i.e., 0.8.6, which allows removing all calls to SafeMath and leads to lower gas costs.

Update: Fixed in this PR.

QSP-20 Stakeholder Incentives Are Not Fully Aligned

Severity: *Informational*

Status: Acknowledged

Description: There is a conflict of interest between the protocols and the security team because the security team is paid a percentage out of the total premium that the protocol pays to Sherlock AND the security team determines the premium price as well. This means that the security team is in theory incentivized to provide a high premium price for a protocol, which would also be beneficial for the Sherlock treasury and for the stakers. However, in practice the premium prices need to be competitive on the DeFi insurance market, otherwise, protocols will not pay it.

Recommendation: Provide a transparent actuarial model where protocols can easily use to compute the premium themselves. It is also not clear how the premium increases (mathematical

function) if the Sherlock security team finds an issue later on which it flags to the protocol, but the protocol does not fix this issue within a reasonable time frame.

Update: Acknowledged by Sherlock team. The detailed response was "*Core team + internal standard for pricing will regulate security experts who seem to be deliberately pricing too low or too high. Eventually this regulation may be managed by SHER tokenholders.*".

QSP-21 Important Contract Operations Cannot Be Easily Monitored

Severity: *Informational*

Status: Acknowledged

File(s) affected: `All contracts are affected.`

Description: The contract does not declare any events, which makes it difficult for 3rd party integrators to monitor the actions performed by the smart contracts.
The only exception is the `SherX.sol` file that defines and emits the `Harvest` event.

Recommendation: Declare and emit meaningful events in all smart contract functions that perform state changes.

Update: Acknowledged by Sherlock team.

QSP-22 Allowance Double-Spend Exploit

Severity: *Informational*

Status: Acknowledged

File(s) affected: `SherXERC20.sol`, `NativeLock.sol`

Description: As it presently is constructed, the contract is vulnerable to the allowance double-spend exploit, as with other ERC20 tokens.

Exploit Scenario:

1. Alice allows Bob to transfer `N` amount of Alice's tokens (`N>0`) by calling the `approve()` method on `Token` smart contract (passing Bob's address and `N` as method arguments)
2. After some time, Alice decides to change from `N` to `M` (`M>0`) the number of Alice's tokens Bob is allowed to transfer, so she calls the `approve()` method again, this time passing Bob's address and `M` as method arguments
3. Bob notices Alice's second transaction before it was mined and quickly sends another transaction that calls the `transferFrom()` method to transfer `N` Alice's tokens somewhere
4. If Bob's transaction will be executed before Alice's transaction, then Bob will successfully transfer `N` Alice's tokens and will gain an ability to transfer another `M` tokens
5. Before Alice notices any irregularities, Bob calls `transferFrom()` method again, this time to transfer `M` Alice's tokens.

Recommendation: The exploit (as described above) is mitigated through use of functions that increase/decrease the allowance relative to its current value, such as `increaseAllowance()` and `decreaseAllowance()`.
Pending community agreement on an ERC standard that would protect against this exploit, we recommend that developers of applications dependent on `approve()` / `transferFrom()` should keep in mind that they have to set allowance to 0 first and verify if it was used before setting the new value. Teams who decide to wait for such a standard should make these recommendations to app developers who work with their token contract.
Note that `SherXERC20` already mitigates this by implementing the `increaseApproval` and `decreaseApproval` functions. However, these functions are not called using the de-facto standard names of `increaseAllowance` and `decreaseAllowance`. Moreover, end-users that do not know about this vulnerability could directly use the `approve` function.

Update: Acknowledged by Sherlock team.

QSP-23 `initializeSherXERC20` May Be Called Multiple Times

Severity: *Informational*

Status: Fixed

File(s) affected: `SherXERC20.sol`

Description: This would re-instantiate `name` and `symbol`. It is not clear why this is allowed.

Recommendation: Revert if the existing `name` or `symbol` have already been set.

Update: Fixed in this PR.

QSP-24 Unused Protocol Manager and Agent Variables

Severity: *Informational*

Status: Acknowledged

File(s) affected: `Gov.sol`

Description: Functionality for `protocolManagers` and `protocolAgents` are set in L161-162 but never used. This makes the audit and maintenance of the code more difficult.

Recommendation: Remove all variables and functionality that is never used.

Update: Acknowledged by Sherlock team.

QSP-25 Strong Dependency On Block Numbers For Financials

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `Gov.sol`, `SherX.sol`, `Manager.sol`, `PoolBase.sol`, `LibSherX.sol`, `LibPool.sol`

Description: There is a strong dependency on `block.number` for financial calculations in at least 31 places inside the code. Current trends of Dapps observe them migrated to Layer2 platforms compatible with Solidity contracts for speed and gas costs. Due to block times and `block.number` speed changes being totally different (usually much faster) in Layer2 chains, this can result in financials extremely disrupted in Layer2 if contracts are migrated in the current state to L2. A more certain impact, but of smaller size, will impact the deployed contracts on the future automated migration to Ethereum 2.0, as current block time is approximately 13.34 seconds but Ethereum 2.0 is designed for a block time of 12 seconds. This can result in tokenomics distorted for Ethereum 2.0 or malicious users leveraging a Layer2 deployments to obtain an unfair advantage over Layer1 users.

Recommendation: Consider using `block.timestamp` instead of `block.number` for financial calculations, the calculations are very similar but with a different scale considering that 1 second can be considered like a virtual block accruing financial results.

Update: Acknowledged by Sherlock team.

Automated Analyses

Slither

Slither did not report any significant issues.

Mythril

Mythril did not report any significant issues.

Adherence to Specification

1. It is a bit difficult to follow the architecture. The way the facets break up the system makes it unclear which users need to interact with which functions. Putting the docstrings in the interfaces instead of the contracts themselves actually makes the code harder to follow, and some internal/private functions do not appear documented because of this. Certain functions names are unclear. For example, `Manager.sol` has a function `_updateData` that is a view function. `Payout._doSherX` lacks a description.

2. If any of the protocol's default on their premium payments, seems that it is the responsibility of the governance to remove that protocol before doing anything else. Otherwise many functions will fail, such as anything that invokes `LibPool.payOffDebtAll(_token[i]);`. May want to document this.

3. In `PoolStorage.sol`, we have these two lines:

```
    // How much token (this) is available for SherX holders
    uint256 sherXUnderlying;
```

Why is "(this)" used there? Alternatively, if this variable does correspond to "this" contract, why is the name "underlying"?

1. In `LibPool.stake`, we have the following snippet:

```
    if (totalLock == 0) {
      // mint initial lock
      lock = 10**18;
    }
```

Would it make more sense to this to mint `lock = _amount * 10 ** 18`? I'm not sure if this would significantly change the math, but it might mitigate some rounding issues if the initial stake is huge.

1. Why does `PoolDevOnly.stake` exist? I don't see how this does anything beyond `PoolOpen.stake`. Is the idea to first add the restricted `PoolDevOnly.stake` function via the diamond pattern so the Sherlock team could pre-stake, and then later replace it with `PoolOpen.stake`? This could use documentation.

2. The function `tokenUnload` has several parts that require clarification:
    1. It describes some token "swap" logic that occurs through `_native.swap`, which implements the `IRemove` interface. It is not entirely clear why this is needed or how it will be implemented, as there is no implementation of `swap` outside of a mock contract.

    2. It is unclear why the `ps.unallocatedSherX` is aliased as `totalFee`, which is sent to an arbitrary `_remaining` address as specified by governance. Will stakers be able to claim their SherX from this `_remaining` address?

    3. It is unclear why the new `ps.firstMoneyOut` is deleted at the end of the function, but the amount is not transferred out; only `ps.stakeBalance.sub(ps.firstMoneyOut)` is transferred to `_remaining`.

3. In `Payout._doSherX`, the comment states: `/// @return sherUsd Total amount of USD of the underlying tokens that are being transferred`. Does this conform to the implementation? I only see the return value of `sherUsd` changed here:

```
    if (address(tokens[i]) == _exclude) {
        // Return USD value of token that is excluded from payout
        sherUsd = amounts[i].mul(sx.tokenUSD[tokens[i]]);
      }
```

Code Documentation

1. It is unclear what the following comment on L18 in `Manager.sol` is indicating: // Once transaction has been mined, protocol is officially insured.
2. Typo on L349 in `Manager.sol`: "Dont change usdPool is prices are equal" -> "Don't change usdPool if prices are equal".
3. In `SherX.sol`, the variable `ineglible_yield_amount` should be "ineligible".

Adherence to Best Practices

1. Error messages in `require` statements are unclear, e.g. `ZERO_GOV`, `MAX`, `DEBT`, `WHITELIST`. It is recommended to add code comments describing each error in more detail and/or to add a docs page with a table where each error message is mapped to a comprehensive description.

2. Magic numbers should be replaced by named constants where the name of the constants should provide a semantic meaning of the value for that constant:

    . `25000000` appears twice in `Gov.sol`

    . `10**18` appears numerous times in `Manager.sol`, `Payout.sol`, `PoolBase.sol`, `SherX.sol`, `LibPool.sol`

    . `10e17` appears once in `Payout.sol`

3. Commented code should be removed:

    . L341-342 in `Gov.sol` contain commented code.

4. Unused code features should be removed. This refers to code that is not commented-out, but which has a "//NOTE: UNUSED" comment associated with it. We have noticed that in the Sherlock protocol the `protocolMangers` feature should not be used, but the code facilitates its use.

    . L89, L161 in `Gov.sol`

    . L17 in `GovStorage.sol`

5. Keep the code as simple (readable) as possible. The following are examples where this best practice is not respected:

    . The `onlyValidToken` modifier takes 2 input parameters i.e., `ps` and `_token`. However, `ps` is always derived from `_token` before `onlyValidToken` is called:

```
PoolStorage.Base storage ps = PoolStorage.ps(_token);
onlyValidToken(ps, _token);
LibPool.payOffDebtAll(_token);
```

    . Simplify and optimize the gas consumption of complex arithmetic expressions by replacing divisions with multiplications, e.g. on L168 in `Payment.sol` the following expression: `excludeUsd.div(curTotalUsdPool.div(SherXERC20Storage.sx20().totalSupply())).div(10e17)` may be simplified in the following way: `excludeUsd.mul(SherXERC20Storage.sx20().totalSupply).div(curTotalUsdPool*10e17)`.

6. Error messages in `require` statements are ambiguous, e.g. "AMOUNT". We recommend providing more context like the function where the error occurs and ideally a message indicating why it occurs.

7. It is not clear why `Gov.getProtocolManager` exists if "UNUSED".

8. All functions (including private/internal ones) should have descriptive names and documentation. Functions such as `_doSherX` are unclear. State variables should also be documented.

9. In `Manager.updateData`, the variable names `sub` and `add` should not be used as they shadow the SafeMath functions, making expressions such as `usdPerBlock.sub(sub.sub(add).div(10**18))` unnecessarily complicated.

10. Some functions are unnecessarily duplicated. For example, `SherX.getUnmintedSherX` is the same as `LibPool.getTotalUnmintedSherX`.

11. One example of naming convention issues -- `PoolStorage` defines `stakeBalance` as follows:

```
    // The total amount staked by the stakers in this pool, **including value of `firstMoneyOut`**
    // if you exclude the `firstMoneyOut` from this value, you get the actual amount of tokens staked
    // This value is also excluding funds deposited in a strategy.
    uint256 stakeBalance;
```

But then `LibPool.sol` defines a function called `stakeBalance` that takes the `PoolStorage.stakeBalance` and removes `firstMoneyOut`.

1. Review compiler warnings and remove useless ones with pragma directives to have a clean output. For example, for interface usage there are usually unused variables warning, remove these onew with `// solhint-disable-next-line` for compiler linter or using the variables without effects.

```
function swapUnderlying(uint256 _underlying) private returns (uint256) {
  _underlying = 0; // void assignment to avoid warnings
  return underlying;
}
```

1. We recommend adding more inline comments and adding new tests to the test suite to reflect new findings in this report.

Update: all best practices discussed or fixed in this PR.

Test Results

Test Suite Results

The test suite is excellent with a total of 442 tests passing and 0 tests failing (on `coverage` run, on `test` run there are some test failing designed to test stuff live on Ethereum Mainnet).

```
Gov
  setInitialGovMain()
     ✓ Initial state
     ✓ Do (49ms)
  transferGovMain()
     ✓ Initial state
     ✓ Do
  setWatsonsAddress()
     ✓ Initial state
     ✓ Do
     ✓ Do again fail
     ✓ Do again
  setUnstakeWindow()
     ✓ Initial state
     ✓ Do
  setCooldown()
     ✓ Initial state
     ✓ Do
  protocolAdd()
     ✓ Initial state (52ms)
```

```
    ✓ Do (71ms)
  protocolUpdate()
    ✓ Do
  protocolDepositAdd()
    ✓ Initial state
    ✓ Do max (60ms)
    ✓ Do
  protocolRemove()
    ✓ Do
  protocolRemove() - Debt
    ✓ Remove fail
    ✓ Remove fail, not removed from pool (62ms)
    ✓ Remove success (71ms)
  tokenInit(), exceed limits
    ✓ Exceed staker limit (48ms)
    ✓ Exceed premium limit
  tokenInit()
    ✓ Do (61ms)
    ✓ Do 2 (54ms)
    ✓ Do 3 (50ms)
  tokenInit() - reinit
    ✓ Do stake reenable (63ms)
    ✓ Do protocol reenable (63ms)
    ✓ Do gov reinit
  tokenDisableStakers()
    ✓ Do (40ms)
  tokenDisableStakers() - Active weight
    ✓ Do
  tokenDisableStakers() - Staker Withdraw
    ✓ Do
    ✓ Cooldown
    ✓ Unstake (46ms)
  tokenDisableStakers() - Harvest
    ✓ Do
    ✓ Harvest fail
    ✓ Harvest success (56ms)
  tokenDisableProtocol()
    ✓ Do (38ms)
  tokenDisableProtocol() - Active Premium
    ✓ Do
  tokenDisableProtocol() - Active SherX
    ✓ Do
  tokenUnload() - Active premium
    ✓ Do
  tokenUnload() - Active strategy
    ✓ Do
  tokenUnload() - Active balances (+activate cooldown)
    ✓ Initial state
    ✓ Do swap, token not added (44ms)
    ✓ Add token (66ms)
    ✓ Do (84ms)
    ✓ Remove
    ✓ Harvest fail
  tokenRemove()
    ✓ Do
  tokenRemove() - Premiums set
    ✓ Do
  tokenRemove() - Active protocol
    ✓ Do
  tokenRemove() - Balance & FMO
    ✓ Balance
  tokenRemove() - SherX
    ✓ Do
  tokenRemove() - Readd, verify unstake
    ✓ Initial state
    ✓ Do
    ✓ Readd
    ✓ Stake (121ms)
  setMaxTokensSherX()
    ✓ Initial state
    ✓ Do
    ✓ Do again
  setMaxTokensStaker()
    ✓ Initial state
    ✓ Do
    ✓ Do again
  setMaxProtocolPool()
    ✓ Initial state
    ✓ Do
    ✓ Do again

GovDev
  ✓ Initial state
  transferGovDev()
    ✓ Do
    ✓ Do again
  renounceGovDev()
    ✓ Do
    ✓ Do again

Manager - Clean
  ✓ Initial state (73ms)
  setTokenPrice(address,uint256)
    ✓ Do (111ms)
    ✓ Do again (105ms)
  setTokenPrice(address[],uint256[])
    ✓ Do (111ms)
    ✓ Do again (111ms)
  setPPm(bytes32,address,uint256)
    ✓ Do (101ms)
    ✓ Do again (106ms)
  setPPm(bytes32,address[],uint256[])
    ✓ Do (125ms)
    ✓ Do again (118ms)
  setPPm(bytes32[],address[][],uint256[][])
    ✓ Do (128ms)
    ✓ Do again (133ms)
  setPPmAndTokenPrice(bytes32,address,uint256,uint256)
    ✓ Do (108ms)
    ✓ Do again (109ms)
  setPPmAndTokenPrice(bytes32,address[],uint256[],uint256[])
    ✓ Do (118ms)
    ✓ Do again (125ms)
  setPPmAndTokenPrice(bytes32[],address,uint256[],uint256)
    ✓ Do (109ms)
    ✓ Do again (108ms)
  setPPmAndTokenPrice(bytes32[],address[][],uint256[][],uint256[][])
    ✓ Do (138ms)
    ✓ Do again (146ms)

Manager - Active
  ✓ Initial state (76ms)
  setTokenPrice(address,uint256)
    ✓ Do (109ms)
  setTokenPrice(address[],uint256[])
    ✓ Do (118ms)
  setPPm(bytes32,address,uint256)
    ✓ Do (106ms)
  setPPm(bytes32,address[],uint256[])
    ✓ Do (119ms)
  setPPm(bytes32[],address[][],uint256[][])
    ✓ Do (136ms)
  setPPmAndTokenPrice(bytes32,address,uint256,uint256)
    ✓ Do (111ms)
  setPPmAndTokenPrice(bytes32,address[],uint256[],uint256[])
    ✓ Do (119ms)
  setPPmAndTokenPrice(bytes32[],address,uint256[],uint256)
    ✓ Do (111ms)
  setPPmAndTokenPrice(bytes32[],address[][],uint256[][],uint256[][])
    ✓ Do (137ms)
```

```
Manager - No Weights
   ✓ Initial state (55ms)
   ✓ Set premium (87ms)
   ✓ Set premium again (84ms)
   ✓ Set watsons and premium (111ms)
   ✓ Set premium again (86ms)

Payout
   ✓ Initital state
  First Money Out
     ✓ Do (58ms)
  Stake Balance
     ✓ Do (60ms)

Payout - SherX
   ✓ Initital state (63ms)
  Not excluding
     ✓ Do complete depletion
     ✓ Do premium token
     ✓ Do (125ms)
  Excluding tokenC
     ✓ Do (113ms)
  With unpaid premium
     ✓ Initital state
     ✓ Do (91ms)

Payout - Non active
   ✓ Initial state
   ✓ Do (55ms)

Payout - Using unallocated SHERX
   ✓ Initital state (57ms)
   ✓ Do (108ms)

Pool
  setCooldownFee()
     ✓ Initial state
     ✓ Do
  depositProtocolBalance()
     ✓ Initial state
     ✓ Do
     ✓ Do twice
  withdrawProtocolBalance()
     ✓ Initial state
     ✓ Do (40ms)
     ✓ Do exceed
     ✓ Do full (38ms)
  stake()
     ✓ Initial state
     ✓ Do (71ms)
     ✓ Do bob (80ms)
  activateCooldown()
     ✓ Initial state
     ✓ Do (75ms)
     ✓ Expired
  activateCooldown() - Fee
     ✓ Initial state (43ms)
     ✓ Do too much
     ✓ Do (102ms)
     ✓ Expired
  cancelCooldown()
     ✓ Initial state
     ✓ Do (57ms)
     ✓ Do twice
  cancelCooldown() - Expired
     ✓ Do
  unstakeWindowExpiry()
     ✓ Initial state
     ✓ Not expired, t=1
     ✓ Not expired, t=2
     ✓ Do (58ms)
     ✓ Do twice
  unstake()
     ✓ Initial state (45ms)
     ✓ Do (84ms)
     ✓ Do twice
  unstake() - Expired
     ✓ Cooldown active, t=1
     ✓ Window of opportunity, t=2
     ✓ Expired, t=3
  unstake() - First Money Out
     ✓ Initial state
     ✓ Do (53ms)
  payOffDebtAll()
     ✓ Initial state
     ✓ t=1
     ✓ Do (46ms)
  cleanProtocol()
     ✓ Initial state
     ✓ Do (63ms)
  cleanProtocol() - Accrued debt (no force)
     ✓ Initial state
     ✓ Do (61ms)
  cleanProtocol() - Accrued debt (force)
     ✓ Initial state
     ✓ Do (58ms)
  cleanProtocol() - Accruing debt
     ✓ Do (69ms)
  cleanProtocol() - No Debt
     ✓ Do
  Exchange rates
     ✓ Initial state (48ms)
     ✓ Initial user stake (70ms)
     ✓ payout
     ✓ After payout exchange rates (38ms)
    Exchange rates, non 18 decimals
       ✓ Initial state (39ms)
       ✓ Initial user stake (67ms)

PoolStrategy
  strategyRemove()
     ✓ Initial state
     ✓ Do
     ✓ Do again
  strategyRemove(), sweep
     ✓ Initial state
     ✓ Do
     ✓ Do again
  strategyUpdate()
     ✓ Initial state
     ✓ Do
     ✓ Do again
     ✓ Do again (active balance) (78ms)
  strategyDeposit()
     ✓ Initial state
     ✓ Do
  strategyWithdraw()
     ✓ Initial state
     ✓ Do
  strategyWithdrawAll()
     ✓ Initial state
     ✓ Do

Production
   ✓ Verify non-dev fail
   ✓ Verify dev success (51ms)

SherX
  setInitialWeight()
     ✓ Do unset
     ✓ Do
```

```
      ✓ Do twice
      ✓ Do twice, with move (62ms)
    setWeights()
      ✓ Initial state
      ✓ Do
      ✓ Do disabled
      ✓ Do wrong
      ✓ Do tokenB, exceed sum (57ms)
      ✓ Do tokenB, beneath sum
      ✓ Do tokenB, single
      ✓ Do overflow
      ✓ Do overflow with watsons
      ✓ Do 30/70 (43ms)
      ✓ Do watsons, exceed sum
      ✓ Do watsons, below sum
      ✓ Do watsons, 10/20/70 (49ms)
      ✓ Do watsons, 20/10/70 (43ms)
    harvestFor(address,address)
      ✓ Initial state (69ms)
      ✓ Setup (115ms)
      ✓ Do (105ms)
      ✓ Do wait (79ms)
      ✓ Do again (103ms)
    harvest calls
      ✓ harvest()
      ✓ harvest(address)
      ✓ harvest(address[])
      ✓ harvestFor(address)
      ✓ harvestFor(address,address)
      ✓ harvestFor(address,address[])
    redeem() - stale
      ✓ Initial state (93ms)
      ✓ Do (119ms)
    redeem() - moving
      ✓ Initial state (103ms)
      ✓ Do (156ms)
    calcUnderlying()
      ✓ Initial state (51ms)
      ✓ t=1 (65ms)
      ✓ t=2 (67ms)
      ✓ t=3 (66ms)
      ✓ t=4, update (112ms)
      ✓ t=5 (64ms)
      ✓ t=6 (66ms)
    accrueSherX(address)
      ✓ Initial state (75ms)
      ✓ Do (93ms)
    accrueSherX()
      ✓ Initial state (78ms)
      ✓ Do (98ms)
    accrueSherXWatsons()
      ✓ Initial state
      ✓ Do (39ms)

  SherXERC20
    ✓ Initial state
    ✓ increaseAllowance()
    ✓ decreaseAllowance()
    ✓ approve()

  SherXERC20 - Active
    ✓ Initial state
    transfer()
      ✓ Do
    transferFrom()
      ✓ Fail
      ✓ Fail again
      ✓ Do
      ✓ Do again
      ✓ Do fail burn

  Stateless
    Gov - State Changing
      setInitialGovMain()
        ✓ Invalid sender
        ✓ Invalid gov
        ✓ Success
      transferGovMain()
        ✓ Invalid sender
        ✓ Invalid gov
        ✓ Invalid gov (same)
        ✓ Success
      setWatsonsAddress()
        ✓ Invalid sender
        ✓ Invalid watsons
      setUnstakeWindow()
        ✓ Invalid sender
        ✓ Invalid window (zero)
        ✓ Invalid window
        ✓ Success
      setCooldown()
        ✓ Invalid sender
        ✓ Invalid cooldown (zero)
        ✓ Invalid cooldown
        ✓ Success
      protocolAdd()
        ✓ Invalid sender
        ✓ Invalid protocol
        ✓ Invalid protocol (zero)
        ✓ Invalid agent (zero)
        ✓ Invalid manager (zero)
        ✓ Invalid token
        ✓ Success
      protocolUpdate()
        ✓ Invalid sender
        ✓ Invalid protocol
        ✓ Invalid protocol (zero)
        ✓ Invalid agent (zero)
        ✓ Invalid manager (zero)
        ✓ Success
      protocolDepositAdd()
        ✓ Invalid sender
        ✓ Invalid protocol
        ✓ Invalid protocol (zero)
        ✓ Invalid lengths (zero)
        ✓ Already added
        ✓ Invalid token
      protocolRemove()
        ✓ Invalid sender
        ✓ Invalid protocol
        ✓ Invalid protocol (zero)
        ✓ ~ state helper ~
        ✓ Success
      tokenInit()
        ✓ Invalid sender
        ✓ Invalid lock
        ✓ Invalid token (zero)
        ✓ Invalid stake (owner)
        ✓ Invalid govpool (zero)
        ✓ Invalid supply
        ✓ Invalid underlying
        ✓ Invalid stakes
        ✓ Invalid premiums
        ✓ Success
      tokenDisableStakers()
        ✓ Invalid sender
        ✓ Invalid index
        ✓ Invalid token
        ✓ Success
      tokenDisableProtocol()
        ✓ Invalid sender
```

```
      ✓ Invalid index
      ✓ Invalid token
      ✓ Success
    tokenUnload()
      ✓ Invalid sender
      ✓ Invalid token
      ✓ Invalid native
      ✓ Invalid sherx
      ✓ Stakes set
      ✓ Success
    tokenRemove()
      ✓ Invalid sender
      ✓ Invalid token
      ✓ Invalid token (zero)
      ✓ Not disabled
      ✓ Success
    setMaxTokensSherX()
      ✓ Invalid sender
    setMaxTokensStaker()
      ✓ Invalid sender
    setMaxProtocolPool()
      ✓ Invalid sender
  GovDev — State Changing
    transferGovDev()
      ✓ Invalid sender
      ✓ Invalid gov (zero)
      ✓ Invalid gov (same)
    renounceGovDev()
      ✓ Invalid sender
    updateSolution()
      ✓ Invalid sender
      ✓ Success
  Manager — State Changing
    setTokenPrice(address,uint256)
      ✓ Invalid sender
      ✓ Invalid token (sherx)
      ✓ Invalid token
    setTokenPrice(address[],uint256[])
      ✓ Invalid sender
      ✓ Invalid length
      ✓ Invalid token (sherx)
      ✓ Invalid token
    setPPm(bytes32,address,uint256)
      ✓ Invalid sender
      ✓ Invalid protocol
      ✓ Invalid token (sherx)
      ✓ Invalid token
    setPPm(bytes32,address[],uint256[])
      ✓ Invalid sender
      ✓ Invalid length
      ✓ Invalid protocol
      ✓ Invalid token (sherx)
      ✓ Invalid token
    setPPm(bytes32[],address[][],uint256[][])
      ✓ Invalid sender
      ✓ Invalid length 1
      ✓ Invalid length 2
      ✓ Invalid length 3
      ✓ Invalid protocol
      ✓ Invalid token (sherx)
      ✓ Invalid token
    setPPmAndTokenPrice(bytes32,address,uint256,uint256)
      ✓ Invalid sender
      ✓ Invalid protocol
      ✓ Invalid token (sherx)
      ✓ Invalid token
    setPPmAndTokenPrice(bytes32,address[],uint256[],uint256[])
      ✓ Invalid sender
      ✓ Invalid length 1
      ✓ Invalid length 2
      ✓ Invalid protocol
      ✓ Invalid token (sherx)
      ✓ Invalid token
    setPPmAndTokenPrice(bytes32[],address,uint256[],uint256)
      ✓ Invalid sender
      ✓ Invalid length
      ✓ Invalid protocol
      ✓ Invalid token (sherx)
      ✓ Invalid token
    setPPmAndTokenPrice(bytes32[],address[][],uint256[][],uint256[][])
      ✓ Invalid sender
      ✓ Invalid length 1
      ✓ Invalid length 2
      ✓ Invalid length 3
      ✓ Invalid length 4
      ✓ Invalid length 5
      ✓ Invalid protocol
      ✓ Invalid token (sherx)
      ✓ Invalid token
  Payout — State Changing
    setInitialGovPayout()
      ✓ Invalid sender
      ✓ Invalid gov
      ✓ Success
    transferGovPayout()
      ✓ Invalid sender
      ✓ Invalid gov
      ✓ Invalid gov (same)
      ✓ Success
    payout()
      ✓ Invalid sender
      ✓ Invalid payout (zero)
      ✓ Invalid payout (this)
      ✓ Invalid length 1
      ✓ Invalid length 2
      ✓ Invalid length 3
      ✓ Invalid token
      ✓ Success
  Payout — View Methods
    getGovPayout()
      ✓ Do
  Pool — State Changing
    setCooldownFee()
      ✓ Invalid sender
      ✓ Invalid fee
      ✓ Invalid token
      ✓ Success
    depositProtocolBalance()
      ✓ Invalid protocol
      ✓ Invalid amount
      ✓ Invalid token
      ✓ Invalid token (disabled)
      ✓ Success
    withdrawProtocolBalance()
      ✓ Invalid sender
      ✓ Invalid protocol
      ✓ Invalid amount
      ✓ Invalid receiver
      ✓ Invalid token
      ✓ Success
    stake()
      ✓ Invalid amount
      ✓ Invalid receiver
      ✓ Invalid token
      ✓ Invalid token (disabled)
      ✓ Success
    activateCooldown()
      ✓ Invalid amount
      ✓ Invalid token
      ✓ Success
    cancelCooldown()
```

```
        ✓ Invalid id
        ✓ Invalid token
        ✓ Success
    unstakeWindowExpiry()
        ✓ Invalid account/id
        ✓ Invalid token
    unstake()
        ✓ Invalid id
        ✓ Invalid receiver
        ✓ Invalid token
        ✓ Success
    payOffDebtAll()
        ✓ Invalid token
        ✓ Success
    cleanProtocol()
        ✓ Invalid sender
        ✓ Invalid receiver
        ✓ Invalid token
        ✓ Invalid token (zero)
        ✓ Invalid index (zero)
        ✓ Success
  ISherX — State Changing
    _beforeTokenTransfer()
        ✓ Invalid sender
    setInitialWeight()
        ✓ Invalid sender
        ✓ Success
    setWeights()
        ✓ Invalid sender
        ✓ Invalid token
        ✓ Invalid lengths
        ✓ Success
    harvest()
        ✓ Success
    harvest(address)
        ✓ Success
    harvest(address[])
        ✓ Success
    harvestFor(address)
        ✓ Success
    harvestFor(address,address)
        ✓ Success
    harvestFor(address,address[])
        ✓ Success
    redeem()
        ✓ Invalid amount
        ✓ Invalid receiver
        ✓ Success
  ISherXERC20 — State Changing
    initializeSherXERC20()
        ✓ Invalid sender
        ✓ Invalid name
        ✓ Invalid symbol
        ✓ Success
    increaseAllowance()
        ✓ Invalid spender
        ✓ Invalid amount
        ✓ Success
    decreaseAllowance()
        ✓ Invalid spender
        ✓ Invalid amount
        ✓ Success
    approve()
        ✓ Invalid spender
    transferFrom()
        ✓ Invalid from
  PoolStrategy — State Changing
    strategyRemove()
        ✓ Invalid sender
        ✓ Invalid token
    strategyUpdate()
        ✓ Invalid sender
        ✓ Invalid token
        ✓ Invalid strategy
    strategyDeposit()
        ✓ Invalid sender
        ✓ Invalid token
        ✓ Invalid amount
        ✓ No strategy
    strategyWithdraw()
        ✓ Invalid sender
        ✓ Invalid token
        ✓ Invalid amount
        ✓ No strategy
    strategyWithdrawAll()
        ✓ Invalid sender
        ✓ Invalid token
        ✓ No strategy


  474 passing (47s)
```

Code Coverage

The code coverage is excellent.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	100	100	100	
ForeignLock.sol	100	100	100	100	
NativeLock.sol	100	100	100	100	
contracts/facets/	100	100	100	100	
Gov.sol	100	100	100	100	
GovDev.sol	100	100	100	100	
Manager.sol	100	100	100	100	
Payout.sol	100	100	100	100	
PoolBase.sol	100	100	100	100	
PoolDevOnly.sol	**100**	**100**	**100**	**100**	
PoolOpen.sol	100	100	100	100	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
PoolStrategy.sol	100	100	100	100	
SherX.sol	100	100	100	100	
SherXERC20.sol	100	100	100	100	
contracts/interfaces/	100	100	100	100	
IGov.sol	100	100	100	100	
IGovDev.sol	100	100	100	100	
ILock.sol	100	100	100	100	
IManager.sol	100	100	100	100	
IPayout.sol	100	100	100	100	
IPoolBase.sol	100	100	100	100	
IPoolStake.sol	100	100	100	100	
IPoolStrategy.sol	100	100	100	100	
IRemove.sol	100	100	100	100	
ISherX.sol	100	100	100	100	
ISherXERC20.sol	100	100	100	100	
ISherlock.sol	100	100	100	100	
IStrategy.sol	100	100	100	100	
contracts/interfaces/aaveV2/	100	100	100	100	
DataTypes.sol	100	100	100	100	
IAToken.sol	100	100	100	100	
IAaveDistributionManager.sol	100	100	100	100	
IAaveGovernanceV2.sol	100	100	100	100	
IAaveIncentivesController.sol	100	100	100	100	
IExecutorWithTimelock.sol	100	100	100	100	
IGovernanceV2Helper.sol	100	100	100	100	
ILendingPool.sol	100	100	100	100	
ILendingPoolAddressesProvider.sol	100	100	100	100	
IProposalValidator.sol	100	100	100	100	
IStakeAave.sol	100	100	100	100	
MockAave.sol	100	100	100	100	
contracts/libraries/	100	100	100	100	
LibPool.sol	100	100	100	100	
LibSherX.sol	100	100	100	100	
LibSherXERC20.sol	100	100	100	100	
contracts/storage/	100	100	100	100	
GovStorage.sol	100	100	100	100	
PayoutStorage.sol	100	100	100	100	
PoolStorage.sol	100	100	100	100	
SherXERC20Storage.sol	100	100	100	100	
SherXStorage.sol	100	100	100	100	
contracts/util/	100	100	100	100	
Import.sol	100	100	100	100	
All files	**100**	**100**	**100**	**100**	

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

```
f2dbee5fa5fb14e73d545dd529c2b2b3ad7036a02c1460003fb5f1494f666e1d ./contracts/ForeignLock.sol
b5a698ab27ad97d34397b8930c6b34a244daccc44b179bb866ccadc30965230b ./contracts/NativeLock.sol
2293e575ff2138e8873b213c44700304f06b48ffb52ef7fdf4a3b40643144b5c ./contracts/interfaces/IPoolBase.sol
cebf303928a31b507bcabe1a037be97bcbc8af5b41341706f49395ab9d0457e2 ./contracts/interfaces/IPayout.sol
0cace961edcd0d2d402f400548f1e271c624dd0fc3dc41e5462f1bb81a8637e6 ./contracts/interfaces/IPoolStrategy.sol
0bb87f0288fed9e6cf031ae3ca0b56b614c524311e6d4160a107d9f827539652 ./contracts/interfaces/ISherX.sol
8a4d708bb947be23a547dcb30a734fe909e1d2facc7bfe7d02cbd181f74fd1f2 ./contracts/interfaces/IPoolStake.sol
523813d994c2d24e801c150fa1e411e1d0c09c87eacbd24b8e80f2b3489b4735 ./contracts/interfaces/ISherlock.sol
b7726ab896c1c95ae8be81df16790c8af8a43a66c20cd24173fe4e1cf647ae26 ./contracts/interfaces/IRemove.sol
c62cba55cbf39372531c53888e5d3f67b109c37c0f821c4417e6d8f79b5161f5 ./contracts/interfaces/IStrategy.sol
2abd721b3c95fad0fc856c178108514b9ee5442926cfe76c0f331230742e9f5d ./contracts/interfaces/IGovDev.sol
9467af8564e28508c5c0f1c60abbc1d1187b9b52b029f9e67d05062a1a1de1df ./contracts/interfaces/IManager.sol
e9e9b2b73da5c5c299a314004109916ccf5b597a84a46acf7216e9ec4d059cb2 ./contracts/interfaces/ISherXERC20.sol
238992aa04e7f1b3489853b208201cb134d6e8b1c376d14485b5ca4cc14c9dc6 ./contracts/interfaces/ILock.sol
4118ab267d414eae024a28e2105e9ca6d2b68616296c9db2aab21ae78ba79a35 ./contracts/interfaces/IGov.sol
b216db3959d5d3991f21f635a137c5689c85cfd5d615443eb65547bfbe29d488 ./contracts/interfaces/aaveV2/IProposalValidator.sol
f653ba8e59d5daa59e4faf4609fefac32a50c769f06c895372587e04bfc69781 ./contracts/interfaces/aaveV2/IGovernanceV2Helper.sol
be0f6a8de524761e59926a0576105c6a090df89236456ad29e1f90065c5a9e4b ./contracts/interfaces/aaveV2/IAaveIncentivesController.sol
7b2bf14c8516fb4b747a954d3091ec8e9959be4fdc0fd8e5f3926a6f58835e79 ./contracts/interfaces/aaveV2/IAaveDistributionManager.sol
0812fefddf1e8936a7076d60824bf64da7ce0f9b604b56c620a19aaf0a9fb505 ./contracts/interfaces/aaveV2/IAaveGovernanceV2.sol
30c02e2519b95b3a933f5624fe752fa1485903492639e1920c50cefeb67e91bb ./contracts/interfaces/aaveV2/ILendingPool.sol
e4bfb2e3224e1d125444a55f00efb8c2ff0cf3c4e28c1749acf7dab0f378c5d1 ./contracts/interfaces/aaveV2/ILendingPoolAddressesProvider.sol
34bb1abaa341c68de831caabc0ddb23fadcd07d69e29c50b66a471ddd0cb00a1 ./contracts/interfaces/aaveV2/DataTypes.sol
f85413659ce7a6f1392c96ea959e300517ad2be10aa0243101231001ff948e21 ./contracts/interfaces/aaveV2/IAToken.sol
b49d3af2879fad4312c2922714b60c34ea9a272f60ac422769e96bd40a0304b0 ./contracts/interfaces/aaveV2/IExecutorWithTimelock.sol
5a9e1420a32f718b145a4a811264fd303f999f2b1ddd002eafa13f4dc1d71083 ./contracts/interfaces/aaveV2/IStakeAave.sol
fe83babe5fcbeec4d8ce7222294d487d66a04796f84ef2b12927fd0e48accd9b ./contracts/interfaces/aaveV2/MockAave.sol
4bb3828068a8312d8788debb28ce58f6eb1358cdc7eabe2a2074aa849c2d2d80 ./contracts/storage/GovStorage.sol
a61fd24b909bf81176682340ef7562e5333a741255ad736e847daed29665e20a ./contracts/storage/PayoutStorage.sol
4b8e537df205d0adc56b2d3d688a82fcd4c65a840a11411b645eb567a256957d ./contracts/storage/SherXStorage.sol
55f3598395d03a47142f4fb005c8f2812c6742a5b6a3ce105c6c90b6edd0de64 ./contracts/storage/PoolStorage.sol
e54b7312e39429cbd5d4babf77e7b0817c075bd06a82cc034ee98dec8e4dc422 ./contracts/storage/SherXERC20Storage.sol
ef3949df59fbdf491020749b56554f9d06831987ddfae707baa0b09a812cce6f ./contracts/libraries/LibSherX.sol
87c99e1e464483d61e830981007b026caee5cff484e74f48611b56bcd8ed5e08 ./contracts/libraries/LibPool.sol
dd113e3810cdf8892c9d7a12a552a146fff0dda263d8bcaf31148481219a2a36 ./contracts/libraries/LibSherXERC20.sol
0ef035d0fd1f00bcfdf8b428dea511f4b6332962145069877bd4db3cc6c12f64 ./contracts/strategies/AaveV2.sol
5ca0b7abe31d8d65afdeee6ecc157687012242e26721710df32e3c22e6c7fcfb ./contracts/util/Import.sol
c9d81ff5c7215710ef5ad0ab08a8491f51d85fb11c34caa2538b4b32a6625200 ./contracts/util/RemoveMock.sol
440551a2078f00a21befbb94c7af08d1571d5b5ef681523f5451fa477e225ab8 ./contracts/util/ERC20Mock.sol
a19fbda17d8ceb026d8ebbdb5e18094585f30947eafaaf700d586f8d1b3e3642 ./contracts/util/StrategyMock.sol
49feef4dc0a32dcd2d6d581c5a7ceea29172b52e03dc6fdf651da99d61832fbc ./contracts/facets/PoolStrategy.sol
befcefaa01bf20cde836c209ff00991185ba5b9818ba3a7ecb019748b0a4185b ./contracts/facets/SherXERC20.sol
b21d5ac57be593881bfdc2c66fb200c6d076474cce1b006248b73cef53e7effb ./contracts/facets/PoolDevOnly.sol
746006ca5e016b2e465cdc0838fd77c85fff17429d0ed064ef8722f58d6b8946 ./contracts/facets/Payout.sol
992dcdc01603be03e4b1d7d9c84725e57f036412d33f6521378a1b29eafac3b6 ./contracts/facets/Gov.sol
20cb5012ae1a613967a3c10ca4f58f9a0013c6ff3f2570c7db1c4bba7b11b8f1 ./contracts/facets/SherX.sol
75d704c0d8d839d45e6b824928e93a2820a67586c8c55ef2bb83d37206907d1d ./contracts/facets/Manager.sol
55c3608d6075a43ad3ca6c88b44888694c39c83035307e11f1d18bcbcbdda155 ./contracts/facets/PoolOpen.sol
81a7e76e40af311c6d4cd3e547f5944093be11afbe8cf05a7534a14ef98679ec ./contracts/facets/GovDev.sol
218212551a42f22c0415e1001183f74be2571beef1ed764c57ff960f968a452e ./contracts/facets/PoolBase.sol
```

Tests

```
aeda6f8ee3d8e424baebd59627bf6f721743b592248755e46862784e6198d854 ./test/Payout.js
6bd6715ac108611497b24f17034a20442cf6a8029bb36d3ee83eb1aacce97b0b ./test/SherXERC20.js
02367cf04d41d177f74837c8cf5a25d8f8275c8c303adb844794247fb758c81e ./test/SherX.js
```

```
a55033b6bcff32ee08699cd199ec1cee75d34cc44ea4a4f08697fb098dd790a2 ./test/GovDev.js
1235d86371893370bbf3751588f0a8ae04abd661a53bce432caad0a762fe1412 ./test/Stateless.js
43207a38cd32dafc5abed30fc18c433eb388343235f13c4b71a8b05e2b7cf899 ./test/Pool.js
9df3febb3563d1cda8ab791195bcbe5adadc57416f277c181aeb7bd0ab415e84 ./test/Manager.js
0e25d01a91919999b46667d42294e020a495107bdc35b5b2badb2287fe7f971e ./test/Gov.js
3b5ae6fc53927ee385f988e71c3683aabfeee73b02ec90d05b6a830446458096 ./test/Production.js
0418923dff3411cdce8d8f5c79d426af31fde94d42b875de44e720496773d206 ./test/PoolStrategy.js
0a06273c516a267cfe8c3926d926a92f87326499a5abc2a7c7137db32f912538 ./test/utilities/index.js
d67031622c046ee35827603ffc15e590efdf18492a47dab3afe23c98e80383d7 ./test/utilities/snapshot.js
e1b73d3d51fef58ad338b5dfc92bf0b3a9d83539b739de51a729f62a65abbdd3 ./test/mainnet-test/AaveV2.js
```

Changelog

- 2021-08-10 - Initial report
- 2021-09-10 - Reaudit report (c9aeaf5)

About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure blockchain platforms at scale using computer-aided reasoning tools, with a mission to help boost the adoption of this exponentially growing technology.

With over 1000 Google scholar citations and numerous published papers, Quantstamp's team has decades of combined experience in formal verification, static analysis, and software verification. Quantstamp has also developed a protocol to help smart contract developers and projects worldwide to perform cost-effective smart contract security scans.

To date, Quantstamp has protected $5B in digital asset risk from hackers and assisted dozens of blockchain projects globally through its white glove security assessment services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Quantstamp's collaborations with leading academic institutions such as the National University of Singapore and MIT (Massachusetts Institute of Technology) reflect our commitment to research, development, and enabling world-class blockchain security.