Sherlock Coverage Agreement

DO NOT EDIT THIS TEMPLATE FILE. MAKE A COPY

Details

Protocol Customer: Ajna
Audit Report(s): Report here
Maximum Desired Coverage Amount: 2,000,000 USDC
Active Coverage Amount: The lesser of
1) the Maximum Desired Coverage Amount
2) the Current TVL of Protocol Customer (if opted into)
3) 80% of Sherlock Staking Pool

Is Current TVL relevant and included in Active Coverage Amount calculation? Yes
TVL Coverage Provided for the Following Chain(s): Ethereum mainnet, Arbitrum, Optimism, Binance Smart Chain, Polygon, Fantom, Tron, Avalanche
Bug Bounty Coverage Amount: 10% of Active Coverage Amount
Bug Bounty Coverage Premium: 0.0% per year
TVL Coverage Deductible: 5% of Active Coverage Amount (before exploit event)
Bug Bounty Deductible: 25% of Bug Bounty Coverage Amount
Claims to be paid in: USDC/DAI
Covered Contract(s):

Repo at Commit:
https://github.com/ajna-finance/contracts/commit/febd576143005b0930e829aa3bff526d83618e76

Contracts at Commit:
1. ajna-core/src/ERC20Pool.sol
2. ajna-core/src/ERC20PoolFactory.sol
3. ajna-core/src/ERC721Pool.sol
4. ajna-core/src/ERC721PoolFactory.sol

5. ajna-core/src/PoolInfoUtils.sol
6. ajna-core/src/PositionManager.sol
7. ajna-core/src/RewardsManager.sol
8. ajna-core/src/base/FlashloanablePool.sol
9. ajna-core/src/base/PermitERC721.sol
10. ajna-core/src/base/Pool.sol
11. ajna-core/src/base/PoolDeployer.sol
12. ajna-core/src/interfaces/pool/IERC3156FlashBorrower.sol
13. ajna-core/src/interfaces/pool/IERC3156FlashLender.sol
14. ajna-core/src/interfaces/pool/IPool.sol
15. ajna-core/src/interfaces/pool/IPoolFactory.sol
16. ajna-core/src/interfaces/pool/commons/IPoolBorrowerActions.sol
17. ajna-core/src/interfaces/pool/commons/IPoolDerivedState.sol
18. ajna-core/src/interfaces/pool/commons/IPoolErrors.sol
19. ajna-core/src/interfaces/pool/commons/IPoolEvents.sol
20. ajna-core/src/interfaces/pool/commons/IPoolImmutables.sol
21. ajna-core/src/interfaces/pool/commons/IPoolInternals.sol
22. ajna-core/src/interfaces/pool/commons/IPoolKickerActions.sol
23. ajna-core/src/interfaces/pool/commons/IPoolLPActions.sol
24. ajna-core/src/interfaces/pool/commons/IPoolLenderActions.sol
25. ajna-core/src/interfaces/pool/commons/IPoolSettlerActions.sol
26. ajna-core/src/interfaces/pool/commons/IPoolState.sol
27. ajna-core/src/interfaces/pool/commons/IPoolTakerActions.sol
28. ajna-core/src/interfaces/pool/erc20/IERC20Pool.sol
29. ajna-core/src/interfaces/pool/erc20/IERC20PoolEvents.sol
30. ajna-core/src/interfaces/pool/erc20/IERC20PoolBorrowerActions.sol
31. ajna-core/src/interfaces/pool/erc20/IERC20PoolFactory.sol
32. ajna-core/src/interfaces/pool/erc20/IERC20PoolLenderActions.sol
33. ajna-core/src/interfaces/pool/erc20/IERC20PoolImmutables.sol
34. ajna-core/src/interfaces/pool/erc20/IERC20Taker.sol
35. ajna-core/src/interfaces/pool/erc721/IERC721Pool.sol
36. ajna-core/src/interfaces/pool/erc721/IERC721PoolBorrowerActions.sol
37. ajna-core/src/interfaces/pool/erc721/IERC721PoolErrors.sol
38. ajna-core/src/interfaces/pool/erc721/IERC721PoolEvents.sol
39. ajna-core/src/interfaces/pool/erc721/IERC721PoolFactory.sol
40. ajna-core/src/interfaces/pool/erc721/IERC721PoolImmutables.sol
41. ajna-core/src/interfaces/pool/erc721/IERC721PoolLenderActions.sol
42. ajna-core/src/interfaces/pool/erc721/IERC721PoolState.sol
43. ajna-core/src/interfaces/pool/erc721/IERC721Taker.sol

44. ajna-core/src/interfaces/position/IPositionManager.sol
45. ajna-core/src/interfaces/position/IPositionManagerDerivedState.sol
46. ajna-core/src/interfaces/position/IPositionManagerEvents.sol
47. ajna-core/src/interfaces/position/IPositionManagerErrors.sol
48. ajna-core/src/interfaces/position/IPositionManagerOwnerActions.sol
49. ajna-core/src/interfaces/position/IPositionManagerState.sol
50. ajna-core/src/interfaces/rewards/IRewardsManager.sol
51. ajna-core/src/interfaces/rewards/IRewardsManagerDerivedState.sol
52. ajna-core/src/interfaces/rewards/IRewardsManagerErrors.sol
53. ajna-core/src/interfaces/rewards/IRewardsManagerEvents.sol
54. ajna-core/src/interfaces/rewards/IRewardsManagerOwnerActions.sol
55. ajna-core/src/interfaces/rewards/IRewardsManagerState.sol
56. ajna-core/src/libraries/external/BorrowerActions.sol
57. ajna-core/src/libraries/external/KickerActions.sol
58. ajna-core/src/libraries/external/LPActions.sol
59. ajna-core/src/libraries/external/LenderActions.sol
60. ajna-core/src/libraries/external/PoolCommons.sol
61. ajna-core/src/libraries/external/PositionNFTSVG.sol
62. ajna-core/src/libraries/external/SettlerActions.sol
63. ajna-core/src/libraries/external/TakerActions.sol
64. ajna-core/src/libraries/helpers/RevertsHelper.sol
65. ajna-core/src/libraries/helpers/PoolHelper.sol
66. ajna-core/src/libraries/helpers/SafeTokenNamer.sol
67. ajna-core/src/libraries/internal/Buckets.sol
68. ajna-core/src/libraries/internal/Deposits.sol
69. ajna-core/src/libraries/internal/Loans.sol
70. ajna-core/src/libraries/internal/Maths.sol

Glossary

Protocol Code - The contract(s) listed under “Covered Contract(s)” above
Protocol Agent Address - The smart contract address provided to Sherlock, which is the address from which claims are made and from which a Protocol Customer’s payout address can be changed
Current TVL - The total value locked in the Protocol Customer’s Covered Contract(s) (as defined above).
Staking Pool - The total value of all tokens (USDC, etc.) held in Sherlock’s V2 staking pool

Maximum Desired Coverage Amount - Please note: The Maximum Desired Coverage Amount is NOT necessarily the amount of funds owed to a Protocol Customer during a payout. It is the desired amount of coverage that the Protocol Customer is willing to pay for. Please refer to the Active Coverage Amount to determine the maximum amount the protocol team could be entitled to in the case of a successful payout.
Token - Any cryptocurrency or form of value that exists on a blockchain (including NFTs, etc.)
Commit - The commit hash in GitHub (or other code management software provider) which is the final group of changes made related to a Sherlock audit/fix review. The changes made in this commit hash must explicitly be signed off on by the Lead Senior Watson of the Sherlock audit. Any subsequent/additional commit hashes or deviations from the above process can result in voided coverage.
Deductible - The amount the Protocol Team must pay before Sherlock will contribute USDC/DAI to a claim. The deductible amount is always deducted from the Active Coverage Amount and Bug Bounty Coverage Amount, meaning that if an Active Coverage Amount is $1M and the deductible comes out to $50k, Sherlock can pay out a maximum of 950k USDC/DAI.

Sherlock TVL Coverage

If Sherlock has agreed to provide coverage on the Protocol Customer's Active Coverage Amount, this coverage will begin on the date the protocol has decided, provided all criteria in "Initiating and Maintaining Active Coverage" have been met.

Please see the section below, "The Spirit of Sherlock Exploit Protection", for a more detailed discussion around the types of smart contract and economic risks Sherlock intends to consider covered events.

In the event of a covered smart contract exploit or economic exploit, the Protocol Customer can submit a claim and be reimbursed for lost on-chain funds up to the stated Active Coverage Amount minus the Deductible at the start block of the exploit. If for any reason the affected user(s) or affected protocol(s) see a partial or full return of the exploited funds before or after a payout from Sherlock, the returned funds must be deducted from the Sherlock payout and any amount paid out by Sherlock relating to the returned funds that had been lost (as part of a claim) must be returned to Sherlock. Please see the sections "Claim Validity", "Sherlock Claims Process", "Payment Process", and "Deciding on Claims", below for more detail.

In the event that Sherlock's Staking Pool TVL decreases, the Current Covered TVL will be decreased to the lesser of the three prongs of the Active Coverage Amount. This means the Protocol Customer will never overpay for their current coverage. Sherlock's Staking Pool TVL can drop for many reasons, including whitehat payouts for Covered Protocols, claims submitted by Covered Protocols for blackhat exploits, a blackhat exploit of Sherlock's staking contracts, etc.

Coverage Premium Pricing

Coverage premium pricing can be found by making reference to Sherlock's documentation and Sherlock's Twitter. Sherlock reserves the right to not provide coverage and refund any up-front coverage payments if the auditors and/or Sherlock don't feel comfortable with the codebase after the audit has been completed.

Bug Bounty Coverage

At the completion of the audit, the Protocol Customer will implement a bug bounty program with a bounty valued at "Bug Bounty Coverage Amount" defined in the "Details" section on page 1 through Immunefi, or an alternative platform agreed upon by Sherlock and the Protocol Customer. Bug bounty pricing is defined in "Details". Typically, if a protocol purchases TVL Coverage + Bug Bounty Coverage, the Bug Bounty pricing is baked into the cost of the Coverage Premium. Covered bug bounty claims are generally characterized by vulnerabilities that, if executed on mainnet, would have resulted in a payout as defined by the sections "Claim Validity", "Sherlock Claims Process", "Payment Process", and "Deciding on Claims" below.

Initiating and Maintaining Active Coverage

To initiate coverage, a Protocol Customer should send a deposit to Sherlock's smart contract address defined in the "Payment Process" section, for at least 1 months worth of payment assuming the Protocol Customer's TVL reaches the maximum Active Coverage Amount.

The amount of the deposit that can be withdrawn by the Protocol Customer will continuously decrease while coverage is active. This is the premium amount a Protocol Customer is "charged." Sherlock updates the Protocol Customer's Current TVL periodically to ensure the Protocol Customer doesn't overpay for coverage.

In order to keep coverage active, the Protocol Customer will need to increase their balance in the Protocol Payment Portal if the "Active Balance" is less than 500 USDC or "Coverage Left" is less than 12 hours, as they run the risk of being temporarily removed from coverage. The coverage will end at the first block after the protocol is removed from coverage. For the avoidance of doubt, even if the Protocol Customer is not currently under coverage, an exploit that occurred during a block before the coverage ended could still be valid and Sherlock will properly assess and pay out that claim when relevant.

Sherlock designed this payment philosophy to help Protocol Customers stay capital efficient and avoid "overpaying" for coverage they don't use. Submitting a premium deposit that is sufficiently large helps the Protocol Customer avoid the risk of a spike up in TVL (up to the Maximum Desired Coverage Amount), requiring the Protocol Customer to quickly increase their "active balance", so they are not temporarily removed from coverage until they fund their account.

The Protocol Customer are advised against making material changes to the Protocol Code which have not been approved and audited by Sherlock. If the changes are approved, coverage will automatically extend to the new contracts and Sherlock will follow up with a revised coverage agreement (this document), noting the new "Covered Contract(s)" in the section "Details".

Sherlock will continue to provide active coverage on all contracts that were originally reviewed by Sherlock and deployed, even if a new unaudited contract(s) is added somewhere in the system. Basically, this just means that Sherlock is not "off the hook" on all the covered, audited contracts if the protocol deploys one incremental contract (or contracts) that has not been approved by Sherlock. Of course, the unapproved incremental contract(s) will not be under coverage and any exploit(s) that wouldn't have been possible without this contract (or contracts) will not be covered.

In the event Sherlock does not review the new code changes and an exploit occurs, Sherlock's two primary claims systems, the SPCC and UMA Optimistic Oracle (see section "Claim Validity" below), will be used to assess whether the exploit would have happened regardless of the unaudited changes, or whether the exploit was caused because of the unaudited changes. In the former situation, where the exploit would have happened regardless, this could constitute a valid claim. In the latter situation, where the exploit happened because of the unaudited changes, this will not constitute a valid claim.

Payment Process

All initial Protocol Customer payments to Sherlock will be made in USDC to the following Ethereum smart contract address: 0x666B8EbFbF4D5f0CE56962a25635CfF563F13161. After the initial coverage payment, subsequent payments should be made through Sherlock's Protocol Payment Portal.

Claim Validity

The process of determining claim validity starts with a Protocol Customer bringing a potentially covered exploit in their protocol to the attention of Sherlock. It is likely Sherlock will recommend bringing in the security experts who performed the audit of that Protocol Customer to understand the nature of the exploit (and how to fix/mitigate it quickly). If there's a possibility that the exploit could be covered, the Protocol Customer can work with Sherlock to decide the amount of the claim.

The method for determining the claim amount should be: the dollar value of all tokens lost/stolen on-chain as a direct result of the exploit, and valued at the timestamp when the exploit first began (using CoinGecko or another price aggregation service). Any off-chain losses or costs associated with attempting to recover the funds are not eligible to be included in the claim amount. Any tokens recovered by the Protocol Customer (e.g. if the exploiter returned the funds) will be valued based on the dollar amount of the tokens returned at the timestamp they were sent back. And these recoveries must be netted against the claim amount (if the claim has not been submitted yet) or must be repaid to Sherlock (if the claim has already been submitted). Sherlock has the right to counsel with other coverage providers so that the aggregate payout amount from multiple providers does not exceed 100% of the loss amount.

Once a possible exploit and the amount claimed by the Protocol Customer is brought to the attention of Sherlock, the process of deciding the validity of the claim begins. The first step is to bring the exploit and amount of the claim to the attention of the Sherlock Protocol Claims Committee ("SPCC") via the Protocol Agent Address. The SPCC is made up of members of the core team of Sherlock as well as well-known security experts in the crypto space. These members will be well-versed in the general nature of exploits and events covered by Sherlock as detailed in this statement of coverage. This committee will be composed of some of the foremost security experts in the DeFi space. Members of the SPCC may have a stake in Sherlock (likely in the form of tokens) and may have an interest in doing what is best for the long-term wellbeing of Sherlock. They will also have reputations and public identities existing outside of Sherlock that they will want to uphold.

Sherlock believes these factors will make it very likely that the members of the SPCC will see it in their best interest to make the most accurate claims decision possible.

The decision made by the SPCC will be binary (either a claim will be accepted or not) and it will be made in 7 days or less from the time the claim was submitted. Once a decision is made on a claim by the SPCC, there are a few possible paths. If the claim is denied by the SPCC, the first path for a Protocol Customer is to accept the SPCC's denial and take no further action.

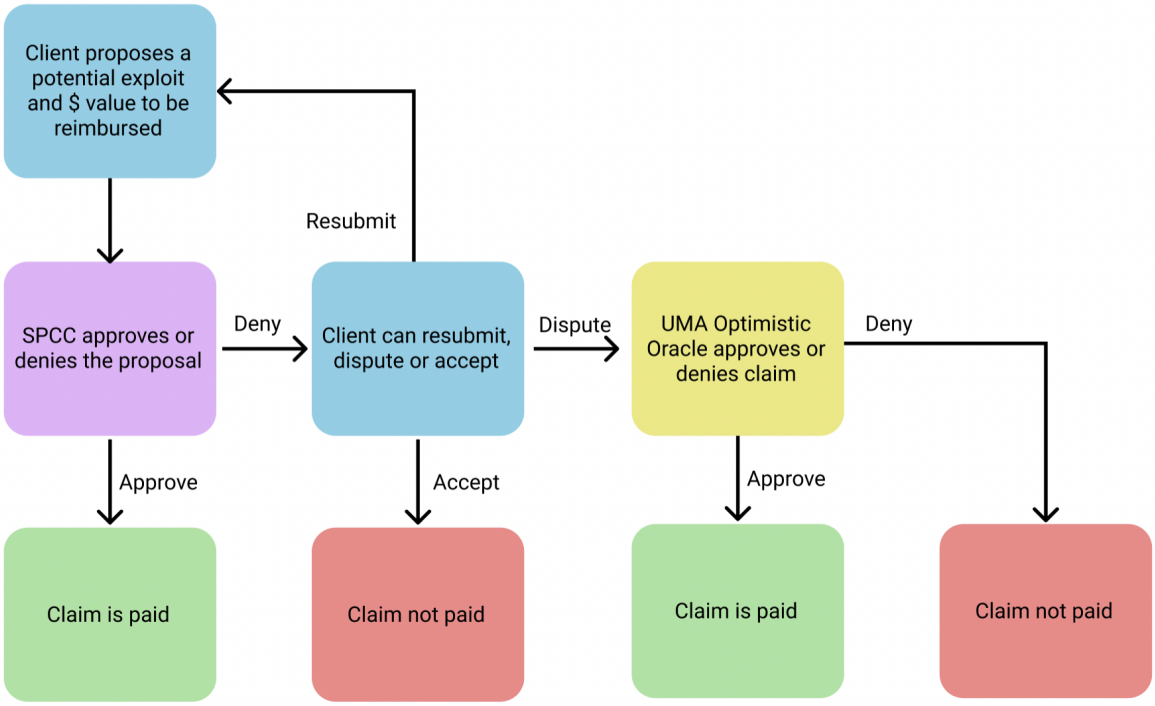
Note: If a Protocol Customer has an existing relationship with a member (or members) of the SPCC and that relationship could constitute a conflict of interest, the member (or members) are required to abstain from participating in any claim votes associated with the Protocol Customer.

The second path is to revise the claim (usually the amount of the claim) and re-submit. A Protocol Customer is limited to 3 submissions for each potential exploit (to be defined by the block number timestamp at which the potential exploit began).

The third and last path for the Protocol Customer is to escalate to arbitration. This would require the Protocol Customer to "stake" the current fee charged by UMA to use their Optimistic Oracle (last priced at ~22k USDC). The escalation would move the claim decision from the SPCC's hands into the hands of UMA's Optimistic Oracle, more specifically UMA's Data Verification Mechanism. The Protocol Customer will have 4 weeks to escalate the claim to UMA's Optimistic Oracle once it has been denied by the SPCC. When escalated to UMA's Optimistic Oracle, the claims decision will be voted on by UMA tokenholders and the resolution of that vote will be the final claim decision (overruling the SPCC).

If the Protocol Customer is proven correct by the UMA Optimistic Oracle, then the amount specified by their claim can be paid out. They will also receive their stake back, minus the fee charged by UMA for using the Optimistic Oracle. If the Protocol Customer's escalation proves to be unsuccessful, then the amount specified by the claim is not paid out and the stake is not returned. Further reading related to UMA's Optimistic Oracle and Data Verification Mechanism can be found here.

Sherlock Claims Process



Deciding on Claims

When trying to decide if a claim falls under coverage or not, there are four main questions to ask (which will be explained in detail in the following pages):

1) Was there an unintended loss of user funds in the protocol? Basically did an exploit occur?
2) Was the exploit due to a flaw/oversight in contracts that were under active coverage with Sherlock?
3) Does this exploit fall into the category of a “Known Economic Risk” explained below?
4) Does this exploit fall into a category under “Specific Events NOT Covered by Sherlock” listed below?

If 1) and 2) are true, meaning an exploit did occur due to a flaw in covered contracts, and 3) and 4) are false, then it is conceivable that this event should be paid out by Sherlock. The reason for approaching the decision in this manner is that Sherlock provides some possibility for “unknown unknown” exploits occurring. And if this event is indeed an exploit, but Sherlock has not provided language around handling it in the letter or spirit of

this document (specifically whether it should NOT be covered), then this new form of exploit should likely be covered by Sherlock.

For claims related to both exploits and bug bounties, sometimes it can be hard to know if a certain exploit is under coverage or not, and if it's a bug bounty it can be hard to determine if the bug report is severe enough for Sherlock to cover it (Sherlock only covers Critical severity bug reports). In these cases, it is the responsibility of the Protocol Customer to pay a mutually agreeable auditor (or auditors) to review the exploit cause or bug report. If Sherlock and the Protocol Customer struggle to mutually agree on an auditor, then the best-performing auditors from the Protocol Customer's most recent Sherlock audit contest should be reached out to, in order of performance in the contest and offered a reasonable hourly rate for reviewing the bug/exploit. If the bug bounty or exploit should be paid out (according to this auditor) then Sherlock will reimburse the Protocol Customer for 50% of the payment made to the auditor who made the determination as part of the official payout.

Utilization of Coverage

TVL Coverage Utilization

The Protocol Customer will be eligible to submit claims up to the Active Coverage Amount for an exploit that occurred at any time when they maintained active coverage (and kept a balance above 0 USDC). At any time, Sherlock may adjust the rate at which the premium will be calculated, although Sherlock endeavors to give (and has historically given) at least 2-weeks notice whenever a rate change will go into effect.

Whenever a claim is paid out by Sherlock to a Protocol Customer (including exploit, bug bounty payouts, etc.), that Protocol Customer is no longer considered to be under active coverage until the on-chain coverage is removed and a new agreement with Sherlock has been reached for new on-chain coverage.

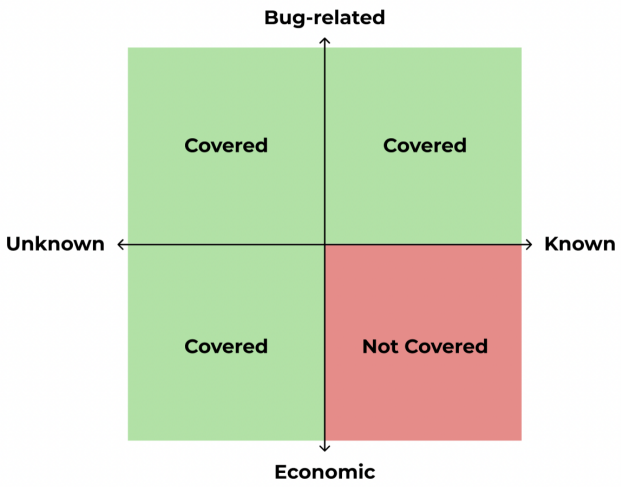
Zero Balance Coverage

Technically, once a Protocol Customer's balance falls below 500 USDC or 12 hours worth of coverage, the covered protocol can be removed from coverage by anyone on-chain. Once a protocol is removed from coverage, they can no longer create claims for exploits that occurred on or after the timestamp at which they were removed from coverage. If a covered protocol's balance hits 0 USDC, the protocol's coverage is considered to be inactive, even if the protocol has not been technically removed from coverage in the smart contracts yet. A Protocol Customer would need to be removed and re-added to

coverage at that point, they cannot simply replenish the balance once it gets to 0 USDC. If a protocol experiences an exploit during the time at which the protocol's balance is 0 USDC (or after an unauthorized replenishment has occurred), the exploit should not be paid out and the SPCC and UMA Optimistic Oracle should vote accordingly.

The Spirit of Sherlock Exploit Protection

This document will outline in detail all of the areas of coverage by Sherlock against which claims can be made (or not made). Because there are always bound to be gaps in explicit wording, Sherlock also attempts to explain the "spirit" of what the later paragraphs will convey, so that unforeseen exploits can still be handled well.



Known Economic Risks

There are two important categories of coverage at Sherlock. The first is bug-related coverage. If a smart contract has a syntax error or otherwise fails to execute its logic as intended due to a mistake related to code being written improperly, that would likely be considered a bug-related incident. However, if there is still a loss of funds despite the code being technically correct in what it intended to do (as a third-party would observe), this would likely fall more in the category of an economic incident. The latter (economic incidents) are not so much a failure of code or syntax as they are a failure of economic design. The difference can be subtle and there are definitely gray areas, but generally if the literal code functions in the way a developer intended (and in the way it was conveyed to Sherlock/auditors), that is likely an economic error. If the literal code does not function as intended, that is likely a bug-related error.

We can create a quadrant of coverage with four types of errors: unknown bug-related errors, known bug-related errors, unknown economic errors, and known economic errors. An unknown error is simply something that the developers/auditors are unaware of (until it surfaces). This means a common bug (in a known class of bugs) can still be an unknown bug in a specific contract because it was not identified in that contract. Whether a bug-related error was previously known or unknown, an incident related to a

smart contract bug should generally be covered. The onus is on Sherlock's security team to find known bugs and help the Protocol Customer fix them securely. And unknown bugs are inherently unforeseeable so they should be covered.

For unknown economic risks, the sentiment is the same. Because it was unknown (and therefore unforeseeable), it should be covered.

But known economic risks are a bit different. Almost every protocol has some set of known economic risks. For example, if the value of Maker collateral falls below the value of the deposits made into the protocol, the depositors are at risk of losing funds. Same goes for almost anything related to token price volatility. If a token price goes down, holders of the token or parties who interact with that token are at risk of losing funds related to the price drop. These are examples of known economic risks. Sometimes, these risks are a large part of the reason APYs are so high for certain opportunities. These are not risks Sherlock intends to cover. Please see the "Specific Economic Events Not Covered By Sherlock" section below which includes some well-known economic risks for various protocol types.

Known Bug-Related Risks

If a Protocol Customer understands the implications of a known bug-related risk, but deems it an "acceptable risk" for their protocol (normally indicated with a "Won't Fix" or "Sponsor Disputed" label in a Sherlock audit repository), it is no longer considered to be covered by Sherlock.

If a Protocol Customer is warned/notified about a bug/vulnerability PRIOR to initiating coverage with Sherlock, this bug/vulnerability would not be considered covered by Sherlock. If the protocol that the Protocol Customer has forked has been notified publicly (or put out a notification) about a bug/vulnerability before the Protocol Customer initiated coverage this would also not be covered.

Sherlock attempts to enumerate, in as clear terms as possible, the events that will or will not be covered by Sherlock coverage:

Specific Economic Events NOT Covered by Sherlock

Token Price Fluctuations

Any loss of funds due to a change in token price or stablecoin depegging should almost certainly not be covered by Sherlock. Any protocol covered by Sherlock must know exactly which tokens it could have the opportunity to interact with (and will be audited

based on this info). Any new token that a protocol intends to interact with should be treated just like any other new integration: with a security review by Sherlock before being executed on mainnet. And any protocol should have contingencies in their code for the price of all of these tokens dropping to zero or approaching infinity. The volatility of a token price is a perfect example of a “known economic risk” as recounted in the preceding section.

Changes in token price especially apply on the user side. The risk of a token’s price going down (or up in the case of short-selling) should always be considered a known risk and thus a loss of funds caused by a change in the price of a token alone should not be a claimable event.

Collateral Shortfalls (example: Lending Protocols)

This section is especially applicable to lending protocols and related protocols. Any lending protocol is well aware that one of the known economic risks is a shortfall in collateral, which would leave depositors unable to collect some or all of their principal. Of course, these collateral shortfalls could be caused by a bug in a smart contract, in which case Sherlock should cover the event. But a common, known economic risk of lending protocols is collateral shortfalls related to rapid and/or large changes in the price of tokens being used as collateral (or the oracles being relied upon for those token prices). This type of collateral shortfall would not be covered by Sherlock.

Unavailability of Funds (example: Lending Protocols)

This section is especially applicable to lending protocols and related protocols. There may be situations where a depositor’s tokens are not available to be withdrawn due to high utilization (on the borrowing side) of the depositor’s tokens. This is a known economic risk related to lending protocols and thus would not be covered.

Impermanent Loss (example: AMM Protocols)

This section is especially relevant for AMM-style protocols. Any LP in an AMM-style protocol should be aware of the risk of impermanent loss when providing liquidity to that protocol.

Slippage (example: AMM Protocols)

This section is especially relevant for AMM or exchange-style protocols. Any user who is doing transactions with an AMM or exchange-style protocol should be aware that losses due to slippage are possible. Slippage losses should generally not be covered by Sherlock.

Counterparty Risk (example: Undercollateralized Protocols)

This section is especially relevant for uncollateralized or undercollateralized lending protocols. If a counterparty or intermediary who received a loan does not repay the loan or has some other payment-related issue, this should not be covered by Sherlock.

Transaction Ordering Attacks / Frontrunning / Sandwich Attacks / MEV-Related Attacks / Chain Re-Orgs

Many of these types of attacks involve malicious addresses (usually controlled by bots) that spot profitable transactions in the mempool and then execute the transaction themselves in order to capture the profit. Or the malicious address sees a certain profitable state change that will be caused by a transaction in the mempool, and calls a function or executes a transaction to take advantage of that state change. Sherlock does not cover these types of attacks because they exist outside the bounds of the covered smart contracts. If a user gets front-run, it means that anyone in the world had a right to call the transaction that the user called, so the user did not necessarily have priority on the value gained from the transaction, which means the "loss" would not be covered.

However, in certain cases, related events would be covered by Sherlock. If, for example, a function was meant to be whitelisted but the modifier was missing, this could be covered because the Sherlock audit process should catch these types of bugs and it should be fairly clear that unsound logic was being used in the code. In other cases, it's not always clear what the intentions of the developers were and therefore Sherlock cannot cover those cases.

Other Specific Events NOT Covered by Sherlock

Social Engineering

If a user is tricked or psychologically manipulated into performing an action or divulging confidential information that results in a loss of assets, Sherlock does not intend to provide reimbursement.

Approve Max / Approve Unlimited

The expectation for protocols covered by Sherlock is that they should discourage (or prevent) approving amounts (of tokens) to a contract above and beyond what is necessary for a specific transaction. Sometimes, it is not possible to entirely prevent this in the smart contracts, but it should at least be made impossible through the covered protocol's sponsored frontend/UI. Sherlock's goal is to protect end-users who may not be sophisticated users of crypto. Any user who goes against the recommendation of the

sponsored UI and “approves unlimited” anyways can be thought of as a sophisticated user according to Sherlock. Users who approve more than they need for a specific transaction and then experience an exploit that drains funds held in their wallet (not at the covered protocol) will not be covered by Sherlock. To be clear, the user’s funds that were in the protocol and lost due to an exploit would be reimbursed. Any funds taken from the user’s wallet due to an excessively high approval value will not be reimbursed by Sherlock.

Rug Pulls / Admin Rights / Off-limits functionality

The unauthorized accessing of any function where access is white-listed or entirely disallowed is NOT covered (assuming the function’s access control is properly implemented in the code). Sherlock strongly recommends multi-signature admin functionality for all accounts and admin contracts.

The risk of funds being lost in a single signature setup is too high for Sherlock to cover. And in a multi-signature setup, the preponderance of evidence related to a loss of funds points to a rug pull (or extremely poor key management), which is a situation Sherlock does not intend to cover. Therefore Sherlock cannot cover ANY “admin”-related exploits. Sherlock is not able to accurately assess these types of exploits currently and so the price of premiums would be far too high if these risks were covered by Sherlock. Sherlock is working to expand its coverage in this area. But for now, any exploit related to privileged access (without an accompanying covered exploit), will not be covered by Sherlock.

This also applies to any governance-induced loss of funds. If a majority of token holders decides to vote maliciously in any way, that cannot be covered. For example, if a majority of token holders decide to transfer a minority’s share of tokens to themselves, this would not be covered. And if a malicious party somehow acquires enough tokens to make a malicious change through governance, this also should not be covered.

Note: A bug related to a missing (or incorrect) access control check (such as a missing modifier) would be covered. This is a mistake in the code, not a “rug pull” necessarily.

Contract / Admin Address Blocklisting / Blacklisting / Freezing

If a protocol’s smart contracts or admin addresses get added to a “blocklist” and the functionality of the protocol is affected by this blocklist, Sherlock will not cover this. A specific example could be a USDC or USDT blocklist. If a protocol uses USDC but the protocol’s contract addresses or admin addresses get blocklisted by USDC and the protocol can no longer function properly as a result, this should not be covered by Sherlock.

Mistakes in Deployment

If a vulnerability becomes possible due to a poorly executed deployment of smart contracts, this is not something Sherlock would cover. However, Sherlock can provide services to check the accuracy/effectiveness of a deployment. Right now, this is seen as an “add-on” to normal security services provided by Sherlock.

Phishing attacks

Users affected by phishing attacks related to their wallet (Metamask, etc.) would not be covered by a specific protocol’s policy. Even if the tokens involved were tokens related to or distributed by a specific protocol that has a policy with Sherlock, this holds true.

Phishing attacks related to “fake” websites (i.e. websites hosted at domains other than the protocol’s sponsored website/app) would also not be covered. The onus is on the user to ensure they are actually interacting with a covered protocol, not a duplicate, replica, or look-alike website or protocol.

Phishing attacks spawning from a covered protocol’s sponsored website/app are also not covered (such as hijacking a DApp’s DNS). Sherlock currently does not have the resources to ensure and monitor the security of website / frontend-related vulnerabilities, but this may change in the future. If getting coverage for this kind of attack is a very high priority for a protocol team, please reach out to Sherlock.

Front-end bugs

In the same vein as phishing attacks, Sherlock currently does not have the resources to ensure and monitor the security of website / frontend-related vulnerabilities, but this may change in the future. So Sherlock cannot cover any unintended loss of funds resulting from an exploit/bug in the frontend (defined as non-Solidity code or code that is not deployed on a blockchain) of a Protocol Customer. This means that code related to libraries like Web3.js or Ethers.js cannot be covered even if it is interacting with smart contracts. The code covered must be deployed on the covered blockchain and frontend code does not meet this criteria.

Keepers

If the protocol is set up in a way where it relies on “keepers” or some external address to execute a job or trigger an action, Sherlock will not be responsible for exploits/freezing/malfunctions caused by the action/inaction/misuse of the keeper by this party or a pause/bug in the blockchain itself. However, if the keeper’s code was in scope, reviewed and signed off on by Sherlock, and an exploit occurs as a result of a bug

in the keeper's code (NOT the action or inaction of the party responsible for calling/maintaining the keeper), then, when taking into account the other facts of the situation, this could qualify as a covered exploit.

Coverage of Layer 2, Sidechain, and Other Chain-Related Risk

Sherlock, unfortunately, cannot cover exploits caused by a bug in the blockchain/L2 code itself, but Sherlock does strive to help make protocols resilient against common scenarios (such as the chain freezing) experienced by blockchains during the audit phase.

Specific Events That Should Not Be Relied on for Decisions

Flash Loan

A flash loan by itself is simply a way to acquire more tokens. Any attack that can be accomplished with a flash loan can also be accomplished without a flash loan (by a whale, etc.). Therefore, the presence of a flash loan does not necessarily mean that an exploit has occurred. However, flash loans are often accompanied by other events that can be covered exploits. If the flash loan is simply taking advantage of a known economic attack (liquidation may occur if a token price drops), then it would not be covered by Sherlock. The presence of flash loans by themselves in a potential exploit event are not good indicators of whether an event should be covered or not.

Oracle Manipulations

Oracle manipulations are well-known events that have caused the loss of tokens in the past. Unfortunately, some oracles (a.k.a. price feeds) like Uniswap V3 are nearly impossible to perfectly protect against manipulation. The only way to protect a Uniswap V3 oracle against manipulation is for the protocol team or Sherlock to LP a certain amount of funds in the pool, across the entire tick range, and never move them. Because Sherlock and most protocol teams are not able or willing to provide this liquidity, Uniswap V3 oracles must always be used "at your own risk." For this reason, Sherlock cannot cover oracle manipulations on Uniswap V3 price feeds or similar price feeds.

For a similar reason, off-chain oracle providers such as Chainlink are also not covered by Sherlock. Sherlock does not cover Protocol Customers against any oracle/price feed risks, such as stale prices, manipulated prices or other malfunctions. However, Sherlock does strive to point out areas during the audit where these risks can be mitigated/handled.

Specific Events Covered by Sherlock

Specific Known “Bug-related” Attacks

Note: All must result in a loss of funds. A temporary pause in the availability of funds (or losses caused by a temporary pause) would not be covered.

- Integer underflow/overflow
- Reentrancy including cross-function reentrancy
- Silent failing sends / unchecked sends / unchecked low-level calls / delegatecall to untrusted callee
- Unbound loops
- Self-destruct-related exploits / forcibly sending Ether to a contract
- Denial-of-service due to fallback function, gas limit reached, unexpected throw, unexpected kill
- False randomness / reliance on “private” information being sent through the mempool
- Time manipulation / timestamp dependence
- Short address attacks
- Insufficient gas griefing
- Authorization through tx.origin
- Uninitialized storage pointer
- Missing checks / callable initialization function
- Missing variables / using the wrong variable
- Proxy/upgradability-related attacks (such as the OpenZeppelin UUPS bug)
- External dependencies if they were in scope of the audit

Known “bug-related” attacks not listed here

The list of specific, known bug-related attacks above is surely incomplete, but is provided mainly for convenience. It is not an exhaustive list of exploit types that Sherlock could pay out and just because an attack is not listed above doesn’t mean that it couldn’t be covered.

Events that Combine Different Attacks

Many exploits combine multiple types of events. As long as just one of the events in the combined attack is determined to be covered by Sherlock, then the attack could be covered. In the opposite direction, if one or more of the events in the combined attack is explicitly not covered, it doesn’t mean that the aggregated attack couldn’t be covered. Losses due specifically to uncovered events always remain uncovered, even in the event of being combined with an incident involving a covered event.

Composability

Sherlock recognizes that composability and a protocol's need to integrate with other projects is a valuable part of the DeFi ecosystem. However, for the security of Sherlock's staking pool, and consequently its covered customers, Sherlock needs to take a careful approach to how integrations are covered.

Both extremes seem unrealistic to expect. It's unreasonable to *only* cover integrations that are done with protocols previously audited by Sherlock. However, it's equally unreasonable to expect Sherlock to cover an integration with any protocol under the sun. So, Sherlock is taking an incremental approach, where the covered integrations will include a whitelisted set of protocols, as well as any protocol whose code has been previously audited by Sherlock (but only the contracts in scope of the Sherlock audit and at the commit hash Sherlock signed off on). Of course, the Protocol Customer still needs to have the integration code go through Sherlock's full audit + fix review + coverage process.

The current list (which will update as Sherlock adds more protocols to coverage) can be found here:
https://github.com/sherlock-protocol/integrations-whitelist/commit/b343edd5d6d1f44e11abfc75c63240c6fc546081

If there is an exploit situation and a covered integration also has coverage from Sherlock or another coverage provider, and a payout takes place (or is scheduled to take place) for that integrated protocol, and the Protocol Customer receives reimbursement from that payout, then the total amount possible to be paid out to the Protocol Customer will be netted against the first payout. This is to ensure that Sherlock doesn't pay the same affected party more than their loss amount.

Events Specific to Ajna

Any issues found in the audit report which were acknowledged by the protocol team, or not fixed, will be excluded from coverage.

Other specific events for the protocol include:
N/A

This coverage agreement and any associated addendums (whitelisted protocol list, bug bounty description) constitute the entire coverage terms and no participant, or Sherlock

as a whole, shall be liable in any manner by any warranties, representations or covenants outside this coverage agreement wording.