SHERLOCK

Sherlock Coverage Overview

Details

Protocol Customer: Opyn
Total Coverage Amount: 10,000,000 USDC
TVL Coverage Premium: 2.0%, per year
Current Covered TVL: The lesser of the Total Coverage Amount, the TVL of Protocol Customer, or 67% of Sherlock Staking Pool
TVL Coverage Provided for the Following Chain(s): Ethereum Layer 1
Bug Bounty Coverage Amount: 10% of Total Coverage Amount
Bug Bounty Coverage Premium: 10%, per year
Claims to be paid in: USDC
Protocol Code Commit hash (contracts folder):
38cecd032a23d4d1c547b78dc228ea4deb1be5db
Covered contracts (contracts folder):
https://github.com/opynfinance/squeeth-monorepo/tree/main/packages/hardhat/contracts
Protocol Code Commit hash (crab-netting folder):
21573d3f70a7250cb6c135fc520a378dc872e5bf
Covered contracts (crab-netting folder):
https://github.com/opynfinance/squeeth-monorepo/tree/packages/crab-netting/src

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 all claims are made and payments in connection with successful claims are received by the Protocol Customer
Current TVL - The total value locked in the Protocol Customer’s Covered Contract(s) (as defined above) as measured monthly by Sherlock
Staking Pool - The total value of all tokens (USDC, etc.) held in Sherlock’s V2 staking

Sherlock TVL Coverage

If Sherlock has agreed to provide coverage on the Protocol Customer's Current Covered TVL, 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 views as 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 funds up to the stated Coverage Amount. Please see the sections *"Claim Validity"*, *"Sherlock Claims Process"*, *"Paying a Claim"*, and *"Deciding on Claims"*, below for more detail.

In the event that Sherlock's Staking Pool TVL drops below 150% of the Coverage Amount agreed upon with the Protocol Customer, the Current Covered TVL will be temporarily decreased to the lesser of 67% of the Staking Pool TVL and the Protocol Customer TVL until the situation is remedied. This means the Protocol Customer will never overpay for their current coverage.

Coverage Premium Pricing

Coverage premium pricing is defined above in the "Details" section. Sherlock reserves the right to not provide coverage and refund any up-front coverage payments if the auditors don't feel comfortable with the codebase after the audit has been completed.

Sherlock only calculates the TVL Coverage Premium on the *lesser* of a Protocol Customer's Current TVL and Coverage Amount.

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. Sherlock will cover the payouts associated with valid critical submissions. 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”, “Paying a Claim”, 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 3 months worth of payment assuming the Protocol Customer’s TVL reaches the maximum 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 monthly to ensure the Protocol Customer doesn’t overpay for coverage.

Over the course of 6 months, the Protocol Customer will need to increase their deposit to the Sherlock payment account if the difference between the “active balance” of funds and the accumulated premium debt is less than $500, 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 is still valid and Sherlock *will* properly assess and pay out that claim when necessary.

At the end of the 6 month coverage period, the Protocol Customer can remove any “unused” funds in their deposit, or request a refund from Sherlock, which will be paid back to the Protocol Customer at an address provided. Alternatively, if a Protocol Customer will be extending coverage for another 6 months, the funds deposited may remain and rolled into the next coverage period.

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 sufficiently large helps to avoid the Protocol Customer from running the risk of a spike up in TVL, requiring the Protocol Customer team to quickly increase their “active balance”, so they are not temporarily removed from coverage until they fund their account.

The Protocol Customer is prohibited from making material changes to the Protocol Code which have not been approved and audited by either Sherlock's Watsons (see section above on "Supplemental Audit Needs"), or an approved external auditor. If the changes are approved, coverage will automatically extend to the new contracts and Sherlock will follow-up with a revised Audit + Coverage Overview, 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 contract is added somewhere in the system and unaudited. Basically, this just means that Sherlock is not "off the hook" on all the covered contracts if the protocol deploys one incremental contract that has not been approved by Sherlock.

In the event Sherlock or an approved auditor 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 is a valid claim and will be covered by Sherlock. In the latter situation, where the exploit happened because of the unaudited changes, this would not be covered by Sherlock.

Payment Process

All Protocol Customer payments to Sherlock will be made in USDC to the following Ethereum smart contract address: 0x666B8EbFbF4D5f0CE56962a25635CfF563F13161

Claim Validity

A Protocol Customer will bring a possible covered exploit in their protocol to the attention of Sherlock's security team. It is likely the security experts at Sherlock in charge of that Protocol Customer will be involved in this process of discovering a possible exploit and understanding its nature. If there's a possibility that the exploit would be covered, the Protocol Customer will be tasked with deciding the amount of the claim. It is likely the security experts at Sherlock in charge of that Protocol Customer will also be heavily involved in advising the proper amount to create a claim for.

Once a possible exploit and the amount claimed by 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 official advisors to Sherlock. 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. All of the members of the SPCC will have a stake in Sherlock (likely in the form of tokens or equity) and will 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. 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.

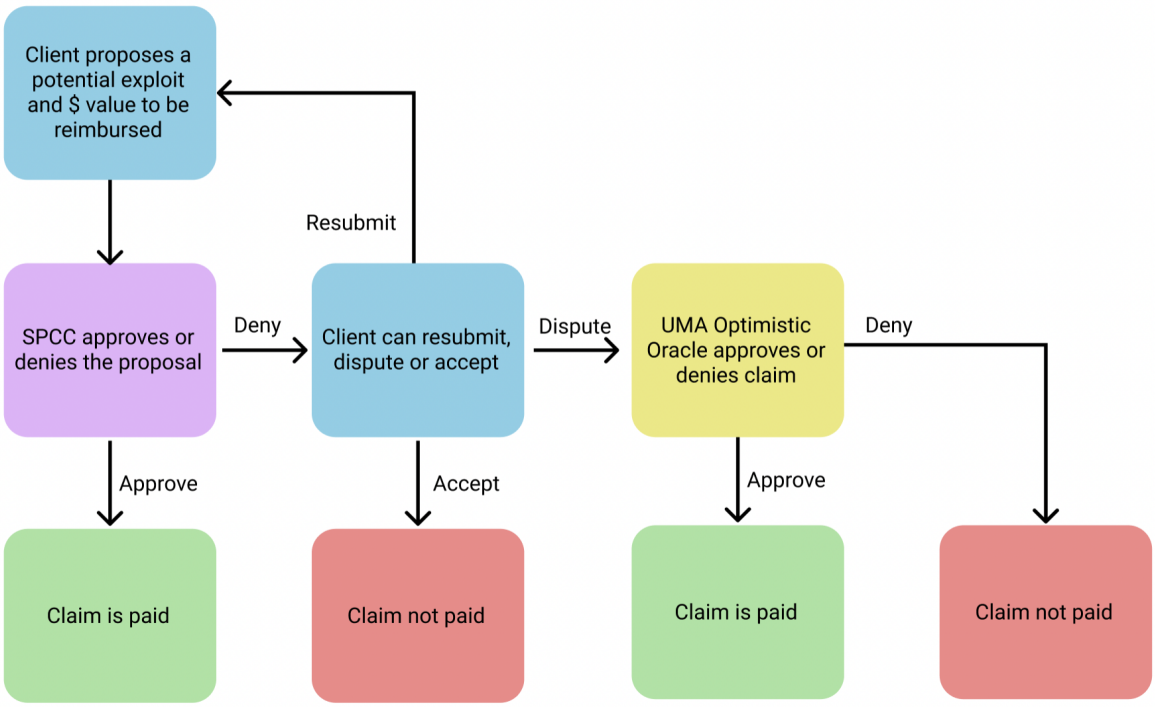
The decision made by the SPCC will be binary (either a claim will be accepted or not). Once a decision is made on a claim by the SPCC, there are a few possible paths. The first path for a Protocol Customer is to accept the decision.

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 (currently 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, then the amount specified by their claim will 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 three main questions to ask (which will be explained in detail in the following pages):

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

If 1) is true, meaning an exploit did occur, and 2) and 3) are false, then it is likely that this event should be covered and 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.

Utilization of Total Coverage Amount

TVL Coverage Utilization

The Protocol Customer will be eligible to submit claims up to their Total Coverage Amount for an exploit that occurred at any time when they maintained active coverage. If a claim is paid out by Sherlock, the Protocol Customer's Total Coverage Amount is reduced by the amount that was paid out. Premiums will continue to be calculated using the pre-determined rate above in *"Details"*, on the lesser of the Current TVL and new Total Coverage Amount. Sherlock and the Protocol Customer will collaboratively discuss whether the necessary steps have been taken to "replenish" their Total Coverage Amount back to the original amount agreed upon. During this collaborative discussion, Sherlock may refuse the right to "replenish" the Total Coverage Amount back to the original amount, and / or adjust the price at which the "replenished" coverage will be calculated.

Said simply, a Protocol Customer may submit claims for exploits until they have used the entirety of their Total Coverage Amount, at which point they will need to discuss replenishing their Total Coverage Amount with Sherlock.

After an exploit payout has occurred, Sherlock will provide an updated version of this document, which explicitly lays out the current agreed-upon Total Coverage Amount.

Bug Bounty Coverage Utilization

Similarly, if a covered bug bounty submission is paid out by Sherlock, the remaining Bug Bounty Coverage Amount is the original amount, *minus* the bug bounty paid out. At which point, Sherlock and the Protocol Customer will discuss raising the bug bounty back to the original size of the Bug Bounty Coverage.

After an exploit payout has occurred, Sherlock will provide an updated version of this document, which explicitly lays out the current agreed-upon Bug Bounty Coverage Amount.

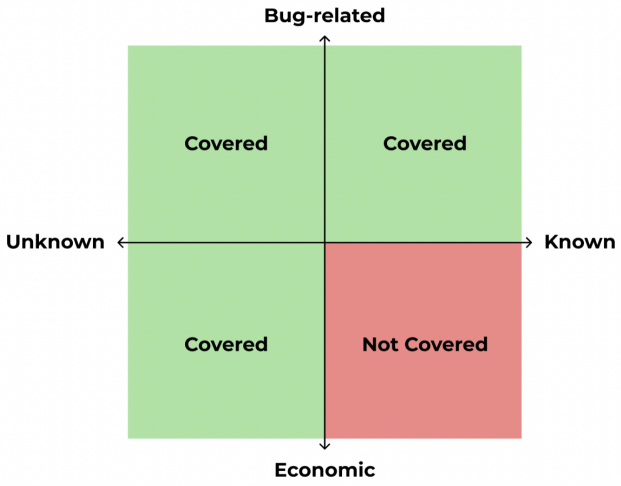
The Spirit of Sherlock Exploit Protection

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, 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 is known or unknown, an incident related to a smart contract bug should generally be covered. The onus is on Sherlock's security team to price known bugs properly or fix them. 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 developer / team understands the implications of a known bug-related risk, but deems it an "acceptable risk" for their protocol, it should still be paid out as long as the team disclosed it to (or at least did not make efforts to conceal it from) the Sherlock smart contract team. As long as the Sherlock team knows about the "acceptable" risk around the code, it can be priced properly in Sherlock's model.

However, if a team makes considerable effort to conceal (or obfuscate) a certain "acceptable risk" or known risk, Sherlock may have grounds to not pay out. This clause exists mainly to disincentivize protocol teams from concealing as many bugs/vulnerabilities as they can from the Sherlock smart contract team in order to get a lower rate for coverage.

With that in mind, 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. 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 or other approved security teams 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.

In some instances, exploits that involve token price fluctuations (among other things) can be covered. See the "Oracle Manipulations" section below.

Collateral Shortfalls (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 of 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. This type of collateral shortfall would not be covered by Sherlock.

Unavailability of Funds (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 (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 (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.

Counterparty Risk (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

These types of attacks involve malicious addresses (often 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 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. The biggest reason that Sherlock cannot cover these types of attacks is because the potential for fraud is too high. If a user or protocol "loses" funds because their transaction is front-run in the mempool, it is very difficult for Sherlock to know that the address doing the frontrunning is not also controlled by the same user or protocol.

However, in certain cases, these types of events would be covered by Sherlock. If, for whatever reason, a protocol tries to pass private or randomness-reliant information through the mempool, this should be covered by Sherlock (see "Specific Known Bug-Related Attacks" below) because the Sherlock security team should catch these types of bugs and in those cases it is 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.

Another area where this would be covered is simply bad logic in the protocol which doesn't check for certain conditions. The specific example here is the ERC20 *approve* race-condition exploit.

Specific Other 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 which 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 exorbitantly high approval value will not be reimbursed by Sherlock.

Rug Pulls / Admin Rights / Off-limits functionality

The unauthorized accessing of any function or contract where access is white-listed or entirely disallowed is NOT covered. 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.

Mistakes in Deployment

If a vulnerability becomes possible due to a poorly executed deployment of smart contracts, this is generally not something Sherlock would cover. However, Sherlock can provide services to check the accuracy/effectiveness of a deployment and then cover deployment-related risks. Right now, this is seen as an "add-on" to normal security services provided by Sherlock because it needs to be done at deployment time instead of during an audit.

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.

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, we ask that the team to reach out to us.

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 a blockchain and frontend code does not meet this criteria.

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 (oracle manipulation, etc.) which are exploits. And, of course, if a flash loan is a part of a broader unknown economic attack, then the event should be covered. 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

Oracles 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 and other DEX-derived price feeds.

However, the Sherlock team has decided that Chainlink oracles have a strong enough history of reliably providing accurate prices and so any oracle manipulation that happens due to a malfunctioning Chainlink oracle WILL be covered by Sherlock. All other oracles outside of Chainlink do not have a strong enough history for Sherlock to comfortably cover their risks.

Specific Events Covered by Sherlock

Specific Known “Bug-related” Attacks

- 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
- Absence of required participants
- 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
- Floating pragma / outdated compiler version / compiler-related bugs
- Missing checks / callable initialization function
- Missing variables / using the wrong variable
- Proxy/upgradability-related attacks (such as the OpenZeppelin UUPS bug)
- External dependencies (such as OpenZeppelin libraries)

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. Any attack that can be classified as bug-related but is not listed

under “Specific Events NOT Covered By Sherlock” should inherently be covered by Sherlock.

Events that Combine Different Attacks

Many exploits combine multiple types of events to disrupt a protocol. As long as just one of the events in the combined attack is determined to be covered by Sherlock, then the entire aggregated attack should be covered.

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 slightly risk-averse 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 any integration that a protocol could work with. So, Sherlock is taking an incremental approach, where the covered integrations will include a whitelisted set of protocols, as well as any protocol that has previously been audited by Sherlock.

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

If the uncovered protocol has coverage from Sherlock or another coverage provider, and a payout takes place (or is scheduled to take place) for that 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 double their loss amount.

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

Sherlock seeks to provide some degree of protection for user funds that reside on non-Ethereum Layer 1 and Layer 2 chains. Sherlock and the Protocol Customer should expressly state in the “Details” section which chains are covered.

If a bug is discovered in the L2/sidechain code and NOT the protocol code, then the maximum amount of payout for that bug across all covered protocols is 50% of the staking pool funds. Of course, the maximum payout per-protocol will also be limited by the size of the coverage amount for that specific protocol.

Bugs in actual L1 or L2 blockchain code should not be paid out if they simply cause the chain to freeze (i.e. not produce new blocks) for a certain period of time. Any protocol building on top of an L1 or L2 should be resilient to long periods of time where no new blocks are created. Unfortunately, these freezes are extremely common on L1s and L2s. However, if the bug in the L1 or L2 code results in a freeze that is expected to be permanent (i.e. funds are trapped or effectively lost forever) then Sherlock would consider this to be a bug that SHOULD get paid out according to the section paragraph in this section.

Events Specific to Opyn

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:

Because Opyn is a derivatives platform, there are specific risks that should be covered or not covered by Sherlock.

Opyn relies on Uniswap V3 oracles for pricing within Squeeth. There are known manipulation risks relating to Uniswap V3, namely the ability for liquidity to be much thinner than expected over certain price ranges. Sherlock considers the possibility of low liquidity across Uniswap V3 pairs to be a known economic risk, and therefore this risk would not, by itself, constitute grounds for a payout. The idea is that a protocol should be able to handle any token price (whether the price has been manipulated or not) without breaking.

Example: Rari's Fuse market exploit on pool #23. This is an example of an exploit made possible due to manipulation in the price of a collateral asset in a lending pool. While the behavior from the Uniswap V3 oracle was unexpected, at the end of the day the exploit became possible because of an inflated value of a token that was used as collateral in the Fuse pool. Sherlock does not want to rely on the prices of certain tokens staying within certain ranges to define payouts.

Other events that are considered known risks within Squeeth are liquidations and impermanent loss. Sometimes these events can be triggered by real smart contract bugs and thus should be

paid out, but other times they naturally result from intended protocol functionality and thus should not be paid out by Sherlock.