SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Contest type:	Public
Prepared for:	xKeeper
Prepared by:	Sherlock
Lead Security Expert:	lllllll
Dates Audited:	April 10 - April 13, 2024
Prepared on:	June 10, 2024

Introduction

xKeeper is a keeper network aggregator which aims to decentralise the on-chain automation of DeFi. xKeeper, facilitates the use of multiple keeper networks, such as Keep3r Network, Gelato, or others. xKeeper is a fully modular framework, designed to be the backbone of future on-chain automation.

Scope

Repository: defi-wonderland/xkeeper-core

Branch: dev

Commit: 615834b896f9d2067a90477612c3e7fbb71cd323

For the detailed scope, see the contest details.

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

Issues found

Medium	High
2	0

Security experts who found valid issues

Kose
Kirkeelee
LTDingZhen
s1ce
lllllll

Issue M-1: L1 data fees are not reimbursed

Source: https://github.com/sherlock-audit/2024-04-xkeeper-judging/issues/57

The protocol has acknowledged this issue.

Found by

lllllll, Kirkeelee, Kose, LTDingZhen, s1ce

Summary

L1 data fees are not reimbursed, and they are often orders of magnitude more expensive than the L2 gas fees being reimbursed. Not reimbursing these fees will lead to jobs not being executed on L2s.

Vulnerability Detail

While the contest README says that the protocol is interested in all EVM-compatible chains, its not clear to what extent they need compatibility. For instance, many chains have workarounds for the fact that they need to publish data from the L2 to the L1 via calldata, and therefore charge for those operations directly. Such caveats indicate that the compatibility is not 100% and therefore we can't really assume that any random chain will be supported. However, the README explicitly mentions Optimism as a target chain, so its idiosyncracies are in-scope.

The Gelato protocol properly handles L1 data fees, but neither Keep3r nor OpenRelay do. It could be argued that for Keep3r, it's possible to modify a job's fee rate dynamically and therefore it's not that big a risk, but for OpenRelay, the reimbursement formula is hard-coded, which means there's no workaround.

Looking at a transaction from this vault, the transaction cost was 0.000305237594921218 ETH $(1.17) but only 0.000000089296942 56 ETH (<$0.01) was reimbursed. The L1 data fee was 0.000304676890061656 Eth, whereas the gas fee was 0.000000111805735391.

Impact

As is shown in this analysis for another contest, the two fees do not have a fixed ratio, and the L1 data fee is frequently much larger than the L2 gas fee, for extended periods of time. This essentially means that the protocol is broken on L2s for any use case which requires timely executions. The contest README states that the sponsor is interested in issues where future integrations would be negatively impacted, and one such case would be where an exchange is trying to use xkeeper to handle customer operations, as is outlined at the end of the linked-to comment

above. Operations will appear to hang for multiple hours at a time, causing loss of funds for customers trying to close their orders.

Code Snippet

Only the L2 gas is reimbursed, not any of the L1 data fees:

```
// File: solidity/contracts/relays/OpenRelay.sol : OpenRelay.exec()   #1

28        // Execute the automation vault counting the gas spent
29        uint256 _initialGas = gasleft();
30        _automationVault.exec(msg.sender, _execData, new
 ↪  IAutomationVault.FeeData[](0));
31        uint256 _gasSpent = _initialGas - gasleft();
32
33        // Calculate the payment for the relayer
34        uint256 _payment = (_gasSpent + GAS_BONUS) * block.basefee *
 ↪  GAS_MULTIPLIER / BASE;
35
36        // Send the payment to the relayer
37        IAutomationVault.FeeData[] memory _feeData = new
 ↪  IAutomationVault.FeeData[](1);
38        _feeData[0] = IAutomationVault.FeeData(_feeRecipient, _NATIVE_TOKEN,
 ↪  _payment);
39:       _automationVault.exec(msg.sender, new IAutomationVault.ExecData[](0),
 ↪  _feeData);
```

https://github.com/sherlock-audit/2024-04-xkeeper/blob/main/xkeeper-core/solidity/contracts/relays/OpenRelay.sol#L28-L39

Tool used

Manual Review

Recommendation

Every L2 has its own formula for calculating the L1 data fee, so different versions of the code will have to be written for each L2. This is the description for Optimism. Note that the Ecotone upgrade has occurred, so be sure to implement based on those or these instructions.

Discussion

sherlock-admin2

1 comment(s) were left on this issue during the judging contest.

Kose commented:

> medium

realfugazzi

Why is this an issue? This is not the responsibility of the OpenRelay contract.

adam-idarrha

@realfugazzi because that contract is in scope , and written by xkeeper .

realfugazzi

@lllllll000 how is this not a feature request, according to your logic?

You are stating that L1 fees are not reimbursed, but there is no place where this is posted as protocol requirement. Job executioners will simulate the op and see if reimbursement is fit for them. Looks like a feature request then, correct?

ashitakah

I agree, it is true that the payments in L2 would be too low to incentivize the work and we are working on improving the system, but it is not a vulnerability and continues to pay although in a residual way.

lllllll000

@ashitakah isn't "too low to incentivize the work" a future integration issue for any project looking to use xkeeper on an L2?

BoRonG0d

Contest readme explicitly states that the contract will be deployed on OP mainnet, and ALL information watson received during the competition did not indicate that OpenRelay would be used as future integration. So this is a valid issue.

And, this was judged as high in the past:

https://github.com/sherlock-audit/2023-07-perennial-judging/issues/91

Issue M-2: Keep3r Relay Implementations are Not Compatible with Keep3r in Optimism and Executions Will Always Revert

Source: https://github.com/sherlock-audit/2024-04-xkeeper-judging/issues/132

Found by

Kose

Summary

`Keep3rRelay` and `Keep3rBondedRelay` uses deprecated function for sidechains and `exec` calls will always revert.

Vulnerability Detail

Keep3r takes different arguments for function `worked()` in sidechains in order to estimate gas usage and rewards for keepers properly:

```
/// @dev Sidechain implementation deprecates worked(address) as it should come
 ↪  with a usdPerGasUnit parameter
function worked(address) external pure override {
  revert Deprecated();
}

/// @notice Implemented by jobs to show that a keeper performed work
/// @dev Uses a USD per gas unit payment mechanism
/// @param _keeper Address of the keeper that performed the work
/// @param _usdPerGasUnit Units of USD (in wei) per gas unit that should be
 ↪  rewarded to the keeper
function worked(address _keeper, uint256 _usdPerGasUnit) external override {
```

The snippet above taken from Keep3rSidechain.sol that is live in optimism currently. We can also see that this contract is exact contract that will be interacted as it is the address of `KEEPER_V2` in deployed Keep3r Relays by xKeeper in Optimism. Deployed addresses can checked from here But relay contracts implemented by xKeeper uses the `worked()` function that is deprecated:

```
// Inject the final call which will issue the payment to the keeper
_execDataKeep3r[_execDataLength + 1] = IAutomationVault.ExecData({
  job: address(KEEP3R_V2),
```

```
  jobData: abi.encodeWithSelector(IKeep3rV2.worked.selector, msg.sender)
});
```

Hence in chains other than mainnet, Keep3r calls will always revert.

Impact

Current Keep3r Relay contracts are not compatible with Keep3r in Optimism (Keeper is only deployed to Mainnet and Optimism currently). Although vault creation will succeed, `exec()` called by keepers will always revert.

Code Snippet

Keep3rRelay.sol Keep3rBondedRelay.sol

Tool used

Manual Review

Recommendation

Either implement a compatible version for Keep3rSideChain, or don't use Keep3r in sidechains.

Discussion

sherlock-admin3

1 comment(s) were left on this issue during the judging contest.

Kose commented:

> medium

Hash01011122

Borderline Low/Med issue

kosedogus

Escalate

The core contract functionality is broken for `Keep3rRelay.sol` and `Keep3rBondedRelay.sol` in all L2's. All `exec()` calls via these contracts will **always** revert, rendering contracts useless. This should be valid medium.

sherlock-admin2

> Escalate
>
> The core contract functionality is broken for `Keep3rRelay.sol` and `Keep3rBondedRelay.sol` in all L2's. All `exec()` calls via these contracts will **always** revert, rendering contracts useless. This should be valid medium.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

ashitakah

The issue is right, in the L2 version of keep3r worked was deprecated, we have to generate versions of the keep3r relays in L2 with the updated worked.

Hash01011122

As I mentioned, above this issue is borderline low/medium issue. In earlier contest of sherlock, issues like this was considered as low severity issue, because of which I considered it as low. I will let head of judge decide the severity.

cvetanovv

For me this issue is Medium and fits the rule:

> Breaks **core** contract functionality, rendering the contract useless.

I am planning to accept the escalation and make this issue Medium.

Evert0x

Result: Medium Unique

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- kosedogus: accepted

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits: https://github.com/xkeeper-framework/xkeeper-core/commit/df7b21882a1073b334625a1be394e38a2bba1cb9

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.