SOLIDIFIED

Audit Report for KrawlCat on September 30, 2020

Summary

Audit Report prepared by Solidified covering the KrawCat oracle smart contracts (and their associated components).

Process and Delivery

Two (2) independent Solidified experts performed an unbiased and isolated audit of the code below. The debrief took place on September 30, 2020.

Audited Files

The following contracts were covered during the audit:

- Medianizer.sol
- Pricefeed.sol

Supplied in the repositories:

https://github.com/yesbit/smart-contract-audits

Notes

The audit was based on commit `c1d5dcb80508537ae9ffdc651327497921dced75`.

Intended Behavior

The smart contracts provide oracle solutions the KrawlCat projects, in order to supply pricing information on different token pairing. MakerDAO's Medianizer and PriceFeed contracts are leveraged and provided with a staking functionality.

SOLIDIFIED

Audit Report for KrawlCat on September 30, 2020

Executive Summary

Solidified found that the KrawlCat contracts contain 4 issues and 5 informational notes.

We recommend all issues are amended, while the notes are up to the team's discretion, as they refer to best practices or optimizations.

Issues found:

Critical	Major	Minor	Notes
2	1	1	5

Issues Found

Critical Issues

1. Subscription system can be circumvented easily

The subscription system for `Medianizer.sol` is ineffective, because anyone can use the system without subscribing and paying the fee. Even if the `read()` method was not callable by anyone, there would be no way to protect the values in the smart contract, since even private variables are visible in the blockchain.

Recommendation
Consider re-designing the economic model.

2. Medianizer.sol: Function compute() will always revert after a large number of feeds have been staked to the contract

After a large enough number of feeds have been staked, function `compute()` will always revert, thus crippling the entire functionality of the contract. This is due to the fact that with an enough number of feeds, executing the `for` loop code inside `compute()` will exceed the maximum block gas limit, and thus miners won't be able to execute it.

Since stakers can permissionlessly add feeds, a common attack vector would be for the attacker to keep staking feeds until the block gas limit is exceeded.

Recommendation
Place a limit on the maximum number of feeds that can be staked, so that the `for` loop code will never exceed the maximum block gas limit (currently ~10M).

Major Issues

3. Incomplete Staking Model

The staking system is currently not connected to the price feed reward system. It seems that anyone can stake and receive awards and anyone can submit price information. There is also no slashing or penalization mechanism for not supplying or providing incorrect data. It is not clear what the purpose of staking is. This may be due to an incomplete implementation.

Recommendation
Consider re-designing the staking model.

Minor Issues

4. ERC-20 return values ignored

`PricingFeed.sol` interacts with an external ERC-20 token (not included in the code provided with the audit. One of the comments suggests this to be token agnostic, even though a certain number of decimal places are assumed (see notes below).
However, the return values of ERC-20 calls are never checked. Many tokens revert on failure but this is not required by the standard and has led to many bugs with tokens that return false on failure.

Note, that we have marked this issue as minor because the token is a trusted component set in the constructor.

Recommendation
Consider wrapping all ERC-20 calls in `require` statements.

Notes

5. Double provision of safe math functions

`Medianizer.sol` defines its own safe math library (taken from MakerDAO) but also imports Open Zeppelin's implementation.

Recommendation
Consider removing one of the implementations to simplify the code or reduce dependencies.

6. Use of abstract contract instead of interface

`Pricefeed.sol` defines an abstract contract `Rewards` (implementation not provided for this audit). Since this contract does not implement any of the functions nor declares types and variables, there is no reason to not use an interface instead of an abstract contract, as is the case with the other external contract interfaces.
The missing implementation of this contract in the audited codebase also means that some of the project's functionality cannot be verified.

Recommendation
Consider using an interface instead of an abstract contract for consistency and code clarity.

7. Check token decimal places

`Medianizer.sol` defines a subscription fee of 10 * 10**18. An accompanying comment suggests that this should be interpreted as 10k. This would mean an unusual number of 10**15 decimal places for the token used for fees.
This is marked as a note, rather than an issue since the number of decimals used is impossible to verify without the token. However, there could be a pricing issue.

Recommendation
Verify the intended fee and decimal place before deployment.

8. Medianizer.sol: stakeTime[user] storage isn't freed upon calling unstake()

The `stakeTime[user]` storage that was allocated during `stake()` is not being freed during `unstake()`.

Recommendation
While this issue is not affecting the smart contract functionality in any way, it's best practice and generally good etiquette to free unused Ethereum blockchain storage whenever possible. The calling user will also get partial gas refunds as soon as the storage is freed.

9. Medianizer.sol: Unused member variable: subBlockDuration

Member variable `subBlockDuration` is not being used anywhere inside the Medianizer contract.

Recommendation
Remove the unused member variable.

Audit Report for KrawlCat on September 30, 2020

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of KrawlCat or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Solidified Technologies Inc.