SOLIDIFIED

Audit Report for NexusMutual on April 22nd, 2019.

Summary

Audit Report prepared by Solidified for Nexus Mutual covering the insurance smart contracts (and their associated components).

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The debrief took place on April 22nd, 2019, and the final results are presented here.

Audited Files

The following contracts were covered during the audit:

- Claims.sol
- MemberRoles.sol
- Quotation.sol
- ClaimsData.sol
- Migrations.sol
- QuotationData.sol
- ClaimsReward.sol
- NXMToken.sol
- TokenController.sol
- EventCaller.sol
- NXMaster.sol
- TokenData.sol
- Governance.sol
- Pool1.sol
- TokenFunctions.sol
- INXMMaster.sol
- Pool2.sol
- Iupgradable.sol
- PoolData.sol
- MCR.sol
- ProposalCategory.sol

Notes

Audit was performed on commit `945ce6f42024ffbebac477783d827b32fdacb4c6` and pragma version 0.4.24

SOLIDIFIED

Audit Report for NexusMutual on April 22nd, 2019.

Intended Behavior

Nexus Mutual is built on the Ethereum blockchain and uses a modular system for grouping of Ethereum smart contracts, allowing logical components of the system to be upgraded without affecting the other components. Following are the key modules of Nexus Mutual.

1. Token Module: Token contracts maintain details of NXM Members and the NXM Tokens held by each of them. A member of the mutual can buy/sell tokens anytime. NXM tokens can be used to purchase a cover, submit a claim, underwrite smart contracts, assess a claim or transfer tokens to other addresses.

2. Quotation Module: Quotation contracts contain all logic associated with creating and expiring covers. Smart contract cover is the first insurance product supported by the mutual. A member can generate a quotation off-chain, and fund the same via NXM tokens / currency assets(currently ETH and DAI). This creates a cover on-chain. Quotation contracts interact with Token Contracts to lock NXM tokens against a cover which are then used at the time of claim submission.

3. Claim Module: Claim contracts manages the entire claim lifecycle starting from submitting a claim against a cover note to taking part in claims assessment to closing a claim.

4. Claim Reward Module: Claims Reward Contract contains the methods for rewarding or punishing the Claim assessors/Members based on the votes cast and the final verdict. All rewards in Nexus Mutual, commission to stakers, rewards to Claims assessors/members for claims assessment, participants in governance are given via this module.

5. Pool Module: Pool contracts contain all logic associated with calling External oracles through Oraclize and processing the results retrieved from the same. The module also encompasses on-chain investment asset management using 0x-protocol.

6. MCR Module: MCR contracts contain functions for recording the Minimum Capital Requirement (MCR) of the system, each day, thus determining the NXM token price.

7. Governance Module: Governance contracts contain the logic for creating, editing, categorizing and voting on proposals followed by action implementation, code upgradability. These governance contracts are generated in line with the GovBlocks Protocol.

SOLIDIFIED

Audit Report for NexusMutual on April 22nd, 2019.

Issues Found

Critical

1. Non-member can empty quotation contract

When a user `initiateMembershipAndCover()` a cover object is created on the `quotationData` contract and makes the user eligible for a full refund of joining fee plus cover value, before the KYC approval is completed. When a user asks for a full refund, the cover object remains intact in the quotationData contract. A user can then call `payJoiningFee()` because he is not yet a member and refundEligible will be false.

Once that succeeds the user becomes eligible for a refund of the joining fee, but when calling `fullRefund()` it reads once again from the quotation data and pays out the fee and the value of the initial cover, which the user already received back. This can be repeated until the quotation contract is empty.

Recommendation
Ensure that refund values are repeatedly checked and manage a state variable or similar to enforce only a singular return. It's also recommended to streamline the joining process in a single source of truth to avoid conflicting information from multiple sources

Amended [20.05.2019]
Issue was fixed by Nexus Mutual team and is no longer present in commit `341679e5b43774e7db3bdb9ecd8fcea5b2fc9bcf`.

2. `NXMasterToken.sol`: `transferFrom()` does not check whether tokens have been locked for voting

---

The transfer function prohibits locked tokens from being moved. However, the `transferFrom` implementation omits this check. This means that members can get another account to move their locked tokens. This also affects the voting mechanism since two members can vote with the same tokens.

Recommendation
Add check for locked tokens in transferFrom.

Amended [20.05.2019]
Issue was fixed by Nexus Mutual team and is no longer present in commit `341679e5b43774e7db3bdb9ecd8fcea5b2fc9bcf`.

3. `Quotation.sol`: Signature verification is liable to signature replay attacks

---

The signature verifications implemented are through the `verifySign` function is liable to signature replay attacks, as signed messages are not unique.

Recommendation

It is highly recommended to keep track of already used signatures or use a nonce. It is also good practice to include the contract address of the called contract in the message to avoid replay attacks between different networks (eg. replaying testnet calls on mainnet).

Amended [20.05.2019]
Issue was fixed by Nexus Mutual team and is no longer present in commit `341679e5b43774e7db3bdb9ecd8fcea5b2fc9bcf`.

Major

4. Multiple Issues with Governance

4.1 Followers get no protection if leader votes against their interest

Nexus Mutual's response:
A follower shall delegate his vote to a member whose decision he/she trusts. In case of a scenario, where the follower is unhappy with the leader, he/she can undelegate his/her voting rights or transfer delegation to some other leader. We think this should protect followers.

4.2 Users can deny a follower choosing a leader by delegating to them first.

Amended [20.05.2019]
Issue was fixed by Nexus Mutual team and is no longer present in commit `341679e5b43774e7db3bdb9ecd8fcea5b2fc9bcf`.

4.3. It's possible to add new categories with `_memberRoleToVote` values different than the one present in MemberRoles `enum`

Amended [20.05.2019]
Issue was fixed by Nexus Mutual team and is no longer present in commit `341679e5b43774e7db3bdb9ecd8fcea5b2fc9bcf`.

4.4. Delegation only works if it's been made more than 7 days ago.

Nexus Mutual's response:
This has been done on intention. This was the simplest way to prevent double-voting on a proposal by a given address. Had this condition not been there, we would have to write a complicated and gas intensive logic while a leader caste a vote. For each follower, we would have to check, whether he/she has not undelegated vote within 7 days and then cross check whether the previous leader had caste a vote on the proposal or not. There could also be scenarios where the previous leader would have undelegated his right and accepted delegations within this 7 day period, resulting in multilayer checks and still the dissatisfaction of missing out on test cases.

4.5. follower is an unbounded array, so it can grow too large to be able to cast a vote.

Amended [20.05.2019]
Issue was fixed by Nexus Mutual team and is no longer present in commit `341679e5b43774e7db3bdb9ecd8fcea5b2fc9bcf`.

4.6. Voters is incremented even if `member.checkRole` returns false.

Amended [20.05.2019]
Issue was fixed by Nexus Mutual team and is no longer present in commit `341679e5b43774e7db3bdb9ecd8fcea5b2fc9bcf`.

4.7. Adding solution functionality is not present.

Nexus Mutual's response:
This has been left on intention as business requirements of Nexus Mutual require a yes/no approach only. This option makes sense for proposals with multiple options.

4.8. Proposals can never get categorized?

Amended [20.05.2019]
Issue was fixed by Nexus Mutual team and is no longer present in commit `341679e5b43774e7db3bdb9ecd8fcea5b2fc9bcf`.

4.9. `UpdateProposal` does not update categories. It seems to reset the category ID to 0, which means that `categorize()` may need to be called separately.

Nexus Mutuals response:
We do not see this an issue.This has been done on intention. When a proposal is edited, it could result in an altogether different intention/purpose. Hence, requires whitelisting again.

Recommendation
We strongly recommend a refactoring of the governance process, taking into consideration all of the above points. Also, consider using a third party tool to accomplish this, using a tool such as an Aragon DAO could make sense and reduce the effort on developing the governance structure. Keep in mind that on-chain governance is a complex process with a lot of open questions.

5. Multiple issues with upgradability

The upgrade scheme is very loose and not very effective. A few issues:
5.1. Replaced contracts are still active (even if they don't hold Ether, they can receive deposits and operate normally, since they still return true on `onlyInternal` modifier.

Amended [20.05.2019]
Issue was fixed by Nexus Mutual team and is no longer present in commit `341679e5b43774e7db3bdb9ecd8fcea5b2fc9bcf`.

5.2. Governance can upgrade each contract masters individually

Amended [20.05.2019]
Issue was fixed by Nexus Mutual team and is no longer present in commit `341679e5b43774e7db3bdb9ecd8fcea5b2fc9bcf`.

5.3. There's no assurance that even though the address for upgrading is correct that the initialization will be as well (Master have to properly call addNew function).

Amended [20.05.2019]
Issue was fixed by Nexus Mutual team and is no longer present in commit `341679e5b43774e7db3bdb9ecd8fcea5b2fc9bcf`.

Recommendation
We strongly recommend a refactoring of the upgradability process, taking into consideration all of the above points. Present upgradability within the contracts grants much room for potential confusion (due to both, complexity and the above notes), and may result in a faulty upgrade.

Minor

6. `Quotation.sol`: Cover can be initiated with a very large amount

The `initiateMembershipAndCover()` function does not use all `SafeMath` for all operations, making it possible to submit a cover with a massive amount of ETH, by overflowing the operation of cover value plus joining fee.

Recommendation
Enforce usage of `SafeMath` throughout the `initiateMembershipAndCover` function.

Amended [20.05.2019]
Issue was fixed by Nexus Mutual team and is no longer present in commit `341679e5b43774e7db3bdb9ecd8fcea5b2fc9bcf`.

7. Multiple Issues with contract initialization

7.1. On `NXMaster.sol` and `MemberRoles.sol` there's no check to guarantee that the contracts are initialized before functions are callable

Amended [20.05.2019]
Issue was fixed by Nexus Mutual team and is no longer present in commit `341679e5b43774e7db3bdb9ecd8fcea5b2fc9bcf`.

7.2. On `NXMaster.sol` there's no enforcing that the addNewVersion function will receive an array with all the contracts. Consider enforcing that `_contractsAddress length == allContractNames.length`

Amended [20.05.2019]
Issue was fixed by Nexus Mutual team and is no longer present in commit `341679e5b43774e7db3bdb9ecd8fcea5b2fc9bcf`.

Recommendation
Make contracts uncallable if they aren't properly initialized.

8. Unbounded arrays can reach block gas limit on iteration

Most of the arrays used in the system don't have any bound and while most of them are never iterated, some of them are, like `getRewardToBeDistributedByUser` in `ClaimsRewrds.sol`.

Recommendation
Consider implementing loops that can be called in multiple transactions or assert that arrays never grows larger than what fits in a single block.

Amended [20.05.2019]
Issue was fixed by Nexus Mutual team and is no longer present in commit `341679e5b43774e7db3bdb9ecd8fcea5b2fc9bcf`.

9. ERC-20 transfers should be wrapped in require statements

The standard defines that the transfer function should return a boolean and although most implementations throw on failure, this behavior is not required. Therefore it is recommended to wrap transfer calls in require.

Amended [20.05.2019]
Issue was fixed by Nexus Mutual team and is no longer present in commit `341679e5b43774e7db3bdb9ecd8fcea5b2fc9bcf`.

10. `EventCaller` functions can be called by anyone

The functions in the Event Caller contract are callable by anyone in the network, which could confuse off chain operators that rely on events being called.
Also some events seem to be defined in the EventCaller contract, others directly in the contracts. It's not clear if this is intentional or simply an inconsistency

Recommendation
Enforce limitations on whom may call which contract functions (either through marking accounts or by having a whitelist)

Amended [20.05.2019]
Issue was fixed by Nexus Mutual team and is no longer present in commit `341679e5b43774e7db3bdb9ecd8fcea5b2fc9bcf`.

Notes

11. `NXMaster` `delegateCallBack` can be resubmitted

Anyone can resubmit a callback by copying the ID and sending to `NXMaster` again. In most cases, there's no issue but it seems counterintuitive.

Amended [20.05.2019]
While the duplicate does seem unintuitive, the usage to control the relevant locking mechanism has merit and is okay to remain in source.

12. Gas Optimizations

`General`
1. Using `Ints` is highly inefficient
2. Importing interfaces instead of whole contracts should reduce dramatically the bytecode size, making even possible to group contracts, like Pool1 and Pool2, and Claim contracts.
3. Consider organizing variables inside structs to make use of tight packing and save storage gas.
4. Group external calls together, because each call is very expensive.

`Claims.sol`

2. `CheckVoteClosing()` : Inverting else if statements order should save gas due to short circuiting:
`if(status != 12 && dateUpd.add(cd.minVotingTime()) >= now)`
3. `SubmitCAVote()` makes 6 consecutives calls to `ClaimsData`. It is significantly cheaper to call one function in CD that handles all sub operations.

`TokenFuntions.sol`
1. `getTokenPrice()` only forwards a call to another contract. It is significantly cheaper to call contract directly.
2. `getTotalStakedTokensOnSmartContract()` calls `tokenData` on every iteration of for loop unnecessarily.

`ClaimsRewrds.sol`
`getRewardToBeGiven()` calls `cd.getFinalVerdict` twice in two lines;
Duplicate reference to MemberRole in `ClaimsRewards`

`NXMaster.sol`
1. Contracts only have a singular `versionDate` that gets overridden, so there's no need to keep an additional mapping.

13. Some tests contain no assertions

In the test files for `burnStakerStake` most test cases do not contain the usage of any assertions, only using logging in its place. Utilization of assertions allows for invariant testing, and is generally considered more comprehensive.

Recommendation
Utilize a more robust testing framework featuring assertions to validate invariants

Amended [20.05.2019]
Issue was fixed by Nexus Mutual team and is no longer present in commit `341679e5b43774e7db3bdb9ecd8fcea5b2fc9bcf`.

14. Upgrade code to Solidity version > 0.5.7

The Solidity version the contracts are primarily written in is 0.4.24, a now known vulnerable version of Solidity. The contracts should be updated and their compiler version fixed.

Recommendation
Refer to the Ethereum changelog (https://github.com/ethereum/solidity/blob/develop/Changelog.md) for the most up to date Solidity version. Contract elements may need to be updated in order to work with the latest Solidity version.

15. Duplicate token code

Open Zeppelin's ERC20 implementation is included in the code base. However, the `NXMToken` does not inherit from this implementation. Instead, large amounts of the code are copied into the implementation.

Recommendation
Consider building on the imported code base or removing it.

16. Hardcoded constants and types

There are instances where integer identifies are hardcoded into the code. For example, the status of a claim in the claims module is represented by an integer.

Recommendation
For readability purposes, the usage of a constant or Enum type is suggested.

17. Multiplication after division

Solidity operates only with integers. Thus, if the division is done before the multiplication, the rounding errors can increase dramatically.
File: MCR.sol
Lines: 210-210
File: MCR.sol
Lines: 247-247
File: Pool2.sol
Lines: 274-274
File: Pool2.sol
Lines: 275-275

Recommendation
Multiplication before division may increase the rounding precision.

18. Usage of `Assert` over `Revert`

Use assert(x) if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use require(x) if x can be false, due to e.g. invalid input or a failing external component.
Line: Iupgradable.sol:46

Recommendation
Consider using assert in instances in which there are absolute instances of certain situations (e.g. a condition that should never under any circumstances hold true).

19. Mixing of named and unnamed return values should be avoided

To avoid confusion, do not mix named and unnamed return values.

SOLIDIFIED

Audit Report for NexusMutual on April 22nd, 2019.

Closing Summary

NexusMutual's contracts contain various critical, major and minor issues, along with several areas of note.

We recommend the issues are amended, while the notes are up to NexusMutual's discretion, as they mainly refer to improving the operation of the smart contract.

Amended [20.05.2019]
All the critical, major and minor issues raised were properly handled by Nexus Mutual team.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of the NexusMutual platform 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.