A CONSENSYS DILIGENCE AUDIT REPORT

The LAO

Date	January 2020
Lead Auditor	Shayan Eskandari
Co-auditors	Sergii Kravchenko, Daniel Luca

1 Summary

ConsenSys Diligence conducted a security audit on Moloch DAO v2 (The LAO, MolochLAO) in January 2020. Moloch DAO v2, is similar to Moloch DAO v1 with additional features such as: Multi-Token Support, Tokens Whitelisting, ~~Emergency Processing~~, Guild Kick, non-voting shares (Loot), and many others (Full change log can be found here).

2 Audit Scope

This audit covered the following files, based on the frozen commit 4bc443f:

File	SHA-1 hash
contracts/Moloch.sol	8f55cc17fcf0488acdc9dd2261dca1e08f42c4ac
contracts/GuildBank.sol	d4329bc7836a1800eb2376da05f76f1783700b9a

UPDATE: *Due to the fact that most of the found issues had a similar remediation, Moloch team decided to start the fixes, in pull-pattern branch, earlier than the scheduled mitigation phase. We are working with Moloch team to review most of the fixes (mainly on GitHub and Telegram chat), and will try to include as much as possible in this report. It should be noted that the system summary, workflow charts, and tool-based analysis were done based on the original commit hash prior to the redesign, and partly updated to reflect the new changes. These changes include but not limited to merging* `Moloch.sol` *and* `GuildBank.sol` *into one contract, and using Pull patterns for withdrawal rather push for fund transfers.*

The final commit hash reviewed in this report is pull-pattern/72de553.

The audit activities can be grouped into the following three broad categories:

1. **Security:** Identifying security related issues within the contract.
2. **Architecture:** Evaluating the system architecture through the lens of established smart contract best practices.
3. **Code quality:** A full review of the contract source code. The primary areas of focus include:
   - Correctness
   - Readability
   - Scalability
   - Code complexity
   - Quality of test coverage

2.1 Documentation

The following documentation was available to the project team:

- Inline code documentation
- The LAO—Moloch v2 Smart Contract Audit Kickoff Call Documentation

- MetaCartel Ventures Whitepaper

3 System Overview

3.1 Detailed Design

MolochDAO consists of two main smart contracts:

- Moloch
- GuildBank

The Actors and their permissions in the system are described in Security specification section. The following figure is a visualization of the actors and the overview of Moloch DAO functionality **before the start of the audit**:

Update: The following is the system overview visualization of the DAO **after the audit**. As seen in the chart, changes are resulted in a much simpler codebase and workflow. Some of these changes are as following:

- Removal of the GuildBank smart contract
- Removal of `bailOut()` , `safeRagequit` , Emergency Processing
- Replacing most of the external calls for ERC20 transfers with internal token balance changes (`userTokenBalances`)
- Replacing external ERC20 transfers within the pull pattern (Withdraw)

Proposal in MolochDAO

Proposals in the MolochDAO use a state machine, which depending on the type of proposal can fall into different states. They must pass requirements, mainly specified in `Moloch._validateProposalForProcessing()` and `Moloch._didPass()` functions, to be processed. The following sequence diagram shows the overall lifecycle of the proposals:

4 Security Specification

This section describes, **from a security perspective**, the expected behavior of the system under audit. It is not a substitute for documentation. The purpose of this section is to identify specific security properties that were validated by the audit team.

4.1 Actors

The relevant actors are as follows:

- **Summoner**
  - Specified by the actor deploying the DAO
  - The sole initial member and share holder (1 share)
  - ~~The summoner cannot be GuildKick~~
  - ~~Receives loots on all bailouts~~
- **Member**
  - The member can sponsor proposals
    - These proposals can be any of the following:
      - Add new member
      - Whitelist Tokens
      - Guild Kick other members
      - Ask for more share/loots in a proposal
      - Make a payment (/trade)
      - Ceremonial Votes (`details` field for off-chain decision)
  - Can rage quit from the DAO and withdraw her funds
  - Can assign a secondary delegate key for her membership
  - Can Vote on proposals
- **Anyone**
  - Can submit a new proposal to the DAO

- Can cancel their own proposals
- Can rage kick jailed members (given that the requirements are satisfied)
- ~~Can bail out jailed members (given that the requirements are satisfied)~~
- Can call to process proposals (given that the requirements are satisfied)
- Withdraw their token balance in the DAO

4.2 Trust Model

MolochDAO uses a social approach to mitigate many of the common decentralize attacks on anonymous stake-based DAOs. Membership Admission is Permissioned, The new member proposals are only sponsored by current members, hence any member of the DAO should be known by at least one of the existing members to join.

Continuing Membership is Permissioned and Community-Policed, meaning that any member can quit the DAO (`ragequit`) or can propose another member to be kicked out of the DAO (`ragekick`), conditioned to votes from sufficient members.

The overall motto of Moloch DAO is that false negatives are better than false positives. Meaning that if for any reason something is going wrong with a proposal, it's better to fail and refund all the participants than to accept a faulty proposal and stake members' funds. This has been implied by the functionalities such as `ragequit` , that if a DAO does not follow ones ideology, they are free to quit the DAO and get a refund.

Most of the trust model and assumptions of the Moloch DAO are well documented in the MetaCartel Ventures Whitepaper.

4.3 Security Considerations

The following is a non-exhaustive list of security considerations:

- Summoner is the sole member in the start of the DAO and is responsible to sponsor new member proposals.
- `periodDuration` is simply the unit of time in Moloch DAO. It is defaulted to 4.8 hours (can be changed in deploy time only), meaning there are 5 periods per day. This period is used to determine time length in the DAO (e.g. `votingPeriod` = 35 * `periodDuration` which is 7 days, same for `gracePeriodLength`). These two periods would mitigate most of the possible attacks (e.g. Burning man attack), however, given enough incentive to attack the DAO, the attacker could potentially congest the network for one period (4.8 hours), and prevent members from rage quiting before the grace period. This attack is really unlikely to happen due to high cost of performing this attack and also considering the social aspect of the DAO that each member is sponsored by another existing member.
- Tribute Token transfers for 0 amount is possible where the proposal does not include a payment but needs to be passed. This can be optimized by a check for the amount before the transfer.

5 Issues

Each issue has an assigned severity:

- **Minor** issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- **Medium** issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- **Major** 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.
- **Critical** issues are directly exploitable security vulnerabilities that need to be fixed.

5.1 `safeRagequit` makes you lose funds Critical ✓ Fixed in Pull Pattern

Resolution
`safeRagequit` no longer exists in the Pull Pattern update. `ragequit` is considered safe as there are no longer any ERC20 transfers in its code flow.

Description

`safeRagequit` and `ragequit` functions are used for withdrawing funds from the LAO. The difference between them is that `ragequit` function

tries to withdraw all the allowed tokens and `safeRagequit` function withdraws only some subset of these tokens, defined by the user. It's needed in case the user or GuildBank is blacklisted in some of the tokens and the transfer reverts. The problem is that even though you can quit in that case, you'll lose the tokens that you exclude from the list.

To be precise, the tokens are not completely lost, they will belong to the LAO and can still potentially be transferred to the user who quit. But that requires a lot of trust, coordination, time and anyone can steal some part of these tokens.

Recommendation

Implementing pull pattern for token withdrawals should solve the issue. Users will be able to quit the LAO and burn their shares but still keep their tokens in the LAO's contract for some time if they can't withdraw them right now.

5.2 Creating proposal is not trustless Critical ✓ Fixed in Pull Pattern

Resolution
this issue no longer exists in the Pull Pattern update, due to the fact that emergency processing and in function ERC20 transfers are removed.

Description

Usually, if someone submits a proposal and transfers some amount of tribute tokens, these tokens are transferred back if the proposal is rejected. But if the proposal is not processed before the emergency processing, these tokens will not be transferred back to the proposer. This might happen if a tribute token or a deposit token transfers are blocked.

code/contracts/Moloch.sol:L407-L411

```
if (!emergencyProcessing) {
    require(
        proposal.tributeToken.transfer(proposal.proposer, proposal.tributeOffered),
        "failing vote token transfer failed"
    );
```

Tokens are not completely lost in that case, they now belong to the LAO shareholders and they might try to return that money back. But that requires a lot of coordination and time and everyone who ragequits during that time will take a part of that tokens with them.

Recommendation

Pull pattern for token transfers would solve the issue.

5.3 Emergency processing can be blocked Critical ✓ Fixed in Pull Pattern

Resolution
Emergency Processing no longer exists in the Pull Pattern update.

Description

The main reason for the emergency processing mechanism is that there is a chance that some token transfers might be blocked. For example, a sender or a receiver is in the USDC blacklist. Emergency processing saves from this problem by not transferring tribute token back to the user (if there is some) and rejecting the proposal.

code/contracts/Moloch.sol:L407-L411

```
if (!emergencyProcessing) {
    require(
        proposal.tributeToken.transfer(proposal.proposer, proposal.tributeOffered),
        "failing vote token transfer failed"
    );
```

The problem is that there is still a deposit transfer back to the sponsor and it could be potentially blocked too. If that happens, proposal can't be processed and the LAO is blocked.

Recommendation

Implementing pull pattern for all token withdrawals would solve the problem. The alternative solution would be to also keep the deposit tokens in the LAO, but that makes sponsoring the proposal more risky for the sponsor.

5.4 Token Overflow might result in system halt or loss of funds Major ✓ Fixed

Resolution

Fixed in fd2da6, and 32ad9b by allowing overflows in most balance calculations (e.g. `unsafeSubtractFromBalance` and `unsafeAddToBalance`). This is to prevent system halt, however as mentioned above, in case of overflow the token balance will be incorrect for token holders and members should take that into account when approving future proposals.

Description

If a token overflows, some functionality such as `processProposal`, `cancelProposal` will break due to safeMath reverts. The overflow could happen because the supply of the token was artificially inflated to oblivion.

This issue was pointed out by Heiko Fisch in Telegram chat.

Examples

Any function using `internalTransfer()` can result in an overflow:

contracts/Moloch.sol:L631-L634

```
function max(uint256 x, uint256 y) internal pure returns (uint256) {
    return x >= y ? x : y;
}
```

Recommendation

We recommend to allow overflow for broken or malicious tokens. This is to prevent system halt or loss of funds. It should be noted that in case an overflow occurs, the balance of the token will be incorrect for all token holders in the system.

`ragekick`, `rageQuit` were fixed by not using safeMath within the function code, however this fix is risky and not recommended, as there are other overflows in other functions that might still result in system halt or loss of funds.

One suggestion is having a function named `unsafeInternalTransfer()` which does not use safeMath for the cases that overflow should be allowed. This mainly adds better readability to the code.

It is still a risky fix and a better solution should be planned.

5.5 Whitelisted tokens limit Major ✓ Fixed

Resolution

mitigated by having separate limits for number of whitelisted tokens (for non-zero balance and for zero balance) in 486f1b3 and follow up commits. That's helpful because it's much cheaper to process tokens with zero balance in the guild bank and you can have much more whitelisted tokens overall.

```
uint256 constant MAX_TOKEN_WHITELIST_COUNT = 400; // maximum number of whitelisted tokens
uint256 constant MAX_TOKEN_GUILDBANK_COUNT = 200; // maximum number of tokens with non-zero balance in guildbank
uint256 public totalGuildBankTokens = 0; // total tokens with non-zero balance in guild bank
```

It should be noted that this is an estimated limit based on the manual calculations and current OP code gas costs. DAO members should consider splitting the DAO into two if more than 100 tokens with non-zero balance are used in the DAO to be safe.

Description

`_ragequit` function is iterating over all whitelisted tokens:

contracts/Moloch.sol:L507-L513

```
for (uint256 i = 0; i < tokens.length; i++) {
    uint256 amountToRagequit = fairShare(userTokenBalances[GUILD][tokens[i]], sharesAndLootToBurn, initialTotalSharesAndLoot);
    // deliberately not using safemath here to keep overflows from preventing the function execution (which would break ragekicks)
    // if a token overflows, it is because the supply was artificially inflated to oblivion, so we probably don't care about it anyways
    userTokenBalances[GUILD][tokens[i]] -= amountToRagequit;
    userTokenBalances[memberAddress][tokens[i]] += amountToRagequit;
}
```

If the number of tokens is too big, a transaction can run out of gas and all funds will be blocked forever. Ballpark estimation of this number is around 300 tokens based on the current OpCode gas costs and the block gas limit.

Recommendation

A simple solution would be just limiting the number of whitelisted tokens.

If the intention is to invest in many new tokens over time, and it's not an option to limit the number of whitelisted tokens, it's possible to add a function that removes tokens from the whitelist. For example, it's possible to add a new type of proposals, that is used to vote on token removal if the balance of this token is zero. Before voting for that, shareholders should sell all the balance of that token.

5.6 Summoner can steal funds using `bailout` Major ✓ Fixed in Pull Pattern

Resolution

`bailout` no longer exists in the Pull Pattern update. Note that in case the member loses their private key the funds will be lost.

Description

Currently, there are 2 major reasons for using the `bailout` function:

1. Kick someone out of the LAO. If the shareholders vote for kicking somebody, the kicked user goes to jail at first. If the LAO kicks someone, it's important not to steal user's funds, but remove them from profit-sharing as soon as possible. Currently, because the user can potentially block some token transfers, funds can't be transferred and the user is still having loot and is participation in a profit-sharing. In order to avoid that, `bailout` function was introduced. It allows anyone to transfer kicked user's funds to the summoner if the user does not call `safeRagequit` (which forces the user to lose some funds). The intention is for the summoner to transfer these funds to the kicked member afterwards. The issue here is that it requires a lot of trust to the summoner on the one hand, and requires more time to kick the member out of the LAO.
2. "lost private key" problem. If someone's private key was lost, shareholders can allow summoner to transfer funds from any user whose keys were lost. The problem is that any member's funds can be stolen by the LAO members and the summoner like that. So every member should keep track of that kind of proposal and is forced to do the ragequit if that proposal passes. That decreases trustlessness because if a user is not tracking the system for some time, the user's money can possibly be stolen.

:ommendation

1. To solve these issues, these 2 intentions should be split into 2 different mechanisms. By implementing pull pattern for token transfers, kicked member won't be able to block the `ragekick` and the LAO members would be able to kick anyone much quicker. There is no need to keep the `bailout` for this intention.
2. If "lost private key" problem should be addressed in the LAO, the time period for the funds recovery should be big because there is no need to do the recovery asap. Recovery can be done without a preliminary kick and can even cover not only the `shares` and `loot`, but also tokens that should be withdrawn (if pull pattern is implemented)

5.7 Sponsorship front-running Major ✓ Fixed in Pull Pattern

Resolution
this issue no longer exists in the Pull Pattern update with Major severity, as mentioned in the recommendation, the front-running vector is still open but no rationale exist for such a behaviour.

Description

If proposal submission and sponsorship are done in 2 different transactions, it's possible to front-run the `sponsorProposal` function by any member. The incentive to do that is to be able to block the proposal afterwards. It's sometimes possible to block the proposal by getting blacklisted at `depositToken`. In that case, the proposal won't be accepted and the emergency processing is going to happen next. Currently, if the attacker can become whitelisted again, he might even not lose the deposit tokens. If not, it will block the whole system forever and everyone would have to ragequit (but that's the part of another issue).

Recommendation

Pull pattern for token transfers will solve the issue. Front-running will still be possible but it doesn't affect anything.

5.8 Delegate assignment front-running Medium Won't Fix

Description

Any member can front-run another member's delegateKey assignment.

if you try to submit an address as your delegateKey, someone else can try to assign your delegate address tp themselves. While incentive of this action is unclear, it's possible to block some address from being a delegate forever. `ragekick` and `ragequit` do not free the delegate address and the delegate itself also cannot change the address.

The possible attack could be that a well-known hard-to-replace multisig address is assigned as a delegateKey and someone else take this address to block it. Also, if the malicious member is about to ragequit or be kicked, it's possible to do this attack without losing anything.

The only way to free the delegate is to make it a member, but then it can never be a delegate after.

Recommendation

Make it possible for a delegateKey to approve delegateKey assignment or cancel the current delegation. And additionally, it may be valuable to clear the delegate address in the `_ragequit` function.

Commit-reveal methods can also be used to mitigate this attack.

5.9 `No` votes are still valid after the ragequit/ragekick Medium Won't Fix

Description

Shareholders can vote for the upcoming proposals 2 weeks before they can be executed. If they ragequit or get ragekicked, their votes are still considered valid. And while the LAO does not allow anyone to ragequit before the last proposal with `Yes` vote is processed, it's still possible to quit the LAO and having active `No` votes on some proposals.

It's not naturally expected behaviour because by that time a user ragequits, they are not part of the LAO and do not have any voting power. Moreover, there is no incentive not to vote `No` just to fail all the possible proposals, because the user won't be sharing any consequences of the result of these proposals. And even incentivized to vote `No` for every proposal just as the act of revenge for the ragekick.

Recommendation

The problem is mitigated by the fact that all rejected proposals can be submitted again and be processed a few weeks after.

It's possible to remove all the `No` votes from the proposals after user's ragekick/ragequit.

5.10 Dilution bound should be a fixed-point number Minor Won't Fix

Resolution
a per-proposal dilution bound was considered for the v1, but kept it global in the interest of code simplicity.

Description

The dilution bound is designed to mitigate an issue where a proposal is passed, then many users ragequit from the DAO and the remaining members have to pay more than they initially intended to. Because of that, the proposal will be automatically rejected if the total amount of shares becomes `dilutionBound` times less than it was before. The problem is that `dilutionBound` is an integer value and it's impossible to configure it to decimal values such as 1.2, for example.

Recommendation

Make `dilutionBound` a fixed-point number.

5.11 Whitelist proposal duplicate Minor Won't Fix

Description

Every time when a whitelist proposal is sponsored, it's checked that there is no other sponsored whitelist proposal with the same token. This is done in order to avoid proposal duplicates.

code/contracts/Moloch.sol:L277-L281

```
// whitelist proposal
if (proposal.flags[4]) {
    require(!tokenWhitelist[address(proposal.tributeToken)], "cannot already have whitelisted the token");
    require(!proposedToWhitelist[address(proposal.tributeToken)], 'already proposed to whitelist');
    proposedToWhitelist[address(proposal.tributeToken)] = true;
```

The issue is that even though you can't sponsor a duplicate proposal, you can still submit a new proposal with the same token.

Recommendation

Check that there is currently no sponsored proposal with the same token on proposal submission.

5.12 Moloch - bool[6] flags can be changed to a dedicated structure Minor Won't Fix

Resolution
The Moloch team decided to leave the `flags` structure as is, and added comments to all the usage of the boolean list values to increase readability and mitigate introduction of bugs due to future updates.

Description

The Moloch contract uses a structure that includes an array of bools to store a few flags about the proposal:

code/contracts/Moloch.sol:L88

```
bool[6] flags; // [sponsored, processed, didPass, cancelled, whitelist, guildkick]
```

This makes reasoning about the correctness of the code a bit complicated because one needs to remember what each item in the flag stands for. The make the reader's life simpler a dedicated structure can be created that incorporates all of the required flags.

Examples

```
        bool[6] memory flags; // [sponsored, processed, didPass, cancelled, whitelist, guildkick]
```

Recommendation

Based on the provided examples change the `bool[6] flags` to the proposed examples.

Flags as bool array with enum (proposed)

This second contract implements the `flags` as a defined structure with each **named** element representing a specific flag. This method makes clear which flag is accessed because they are referred to by the name, not by the index.

This third contract has the least amount of changes to the code and uses an enum structure to handle the index.

```
pragma solidity 0.5.15;

contract FlagsEnum {
    struct Proposal {
        address applicant;
        uint value;
        bool[3] flags; // [sponsored, processed, kicked]
    }

    enum ProposalFlags {
        SPONSORED,
        PROCESSED,
        KICKED
    }

    uint proposalCount;

    mapping(uint256 => Proposal) public proposals;

    function addProposal(uint _value, bool _sponsored, bool _processed, bool _kicked) public returns (uint) {
        Proposal memory proposal = Proposal({
            applicant: msg.sender,
            value: _value,
            flags: [_sponsored, _processed, _kicked]
        });

        proposals[proposalCount] = proposal;
        proposalCount += 1;

        return (proposalCount);
    }

    function getProposal(uint _proposalId) public view returns (address, uint, bool, bool, bool) {
        return (
            proposals[_proposalId].applicant,
            proposals[_proposalId].value,
            proposals[_proposalId].flags[uint(ProposalFlags.SPONSORED)],
            proposals[_proposalId].flags[uint(ProposalFlags.PROCESSED)],
            proposals[_proposalId].flags[uint(ProposalFlags.KICKED)]
        );
    }
}
```

6 Tool-Based Analysis

Several tools were used to perform automated analysis of the reviewed contracts. These issues were reviewed by the audit team, and relevant issues are listed in the Issue Details section.

6.1 MythX

MythX is a security analysis API for Ethereum smart contracts. It performs multiple types of analysis, including fuzzing and symbolic execution, to detect many common vulnerability types. The tool was used for automated vulnerability discovery for all audited contracts and libraries. More details on MythX can be found at mythx.io.

6.2 Ethlint

Ethlint is an open source project for linting Solidity code. Only security-related issues were reviewed by the audit team.

Below is the raw output of the Ethlint vulnerability scan:

```
contracts/GuildBank.sol
  13:8     warning    Provide an error message for require()                                                  error-reason
  23:12    warning    Provide an error message for require()                                                  error-reason
  34:8     warning    Provide an error message for require()                                                  error-reason
  36:27    warning    There should be no whitespace or comments between the opening brace '{' and first item.    whitespace
  36:37    warning    There should be no whitespace or comments between the last item and closing brace '}'.     whitespace

contracts/Moloch.sol
  34:4      warning    Line exceeds the limit of 145 characters                                 max-len
  41:4      warning    Line exceeds the limit of 145 characters                                 max-len
  42:4      warning    Line exceeds the limit of 145 characters                                 max-len
  76:8      warning    Line exceeds the limit of 145 characters                                 max-len
  169:24    warning    Avoid using 'now' (alias to 'block.timestamp').                          security/no-block-members
  262:8     warning    Line exceeds the limit of 145 characters                                 max-len
  272:49    error      String literal must be quoted with double quotes.                        quotes
  280:74    error      String literal must be quoted with double quotes.                        quotes
  285:57    error      String literal must be quoted with double quotes.                        quotes
  362:8     warning    Line exceeds the limit of 145 characters                                 max-len
  517:8     warning    Line exceeds the limit of 145 characters                                 max-len
  540:13    warning    Assignment operator must have exactly single space on both sides of it.  operator-whitespace
  559:8     warning    Error message exceeds max length of 76 characters                        error-reason
  583:8     warning    Error message exceeds max length of 76 characters                        error-reason
  641:15    warning    Avoid using 'now' (alias to 'block.timestamp').                          security/no-block-members

contracts/oz/ERC20.sol
  128:8    warning    Provide an error message for require()    error-reason
  143:8    warning    Provide an error message for require()    error-reason
  157:8    warning    Provide an error message for require()    error-reason
  171:8    warning    Provide an error message for require()    error-reason
  172:8    warning    Provide an error message for require()    error-reason

contracts/oz/SafeMath.sol
  10:8    warning    Provide an error message for require()    error-reason
  17:8    warning    Provide an error message for require()    error-reason
  24:8    warning    Provide an error message for require()    error-reason
  32:8    warning    Provide an error message for require()    error-reason

contracts/test-helpers/Submitter.sol
  9:2     error    Only use indent of 4 spaces.    indentation
  11:2    error    Only use indent of 4 spaces.    indentation
  13:2    error    Only use indent of 4 spaces.    indentation
  15:0    error    Only use indent of 4 spaces.    indentation
  17:2    error    Only use indent of 4 spaces.    indentation
  39:0    error    Only use indent of 4 spaces.    indentation
  41:2    error    Only use indent of 4 spaces.    indentation
  51:0    error    Only use indent of 4 spaces.    indentation
  53:2    error    Only use indent of 4 spaces.    indentation
  63:0    error    Only use indent of 4 spaces.    indentation

contracts/tokens/ClaimsToken.sol
  95:1     error      Only use indent of 4 spaces.    indentation
  98:1     error      Only use indent of 4 spaces.    indentation
  100:1    error      Only use indent of 4 spaces.    indentation
  102:1    error      Only use indent of 4 spaces.    indentation
  105:1    error      Only use indent of 4 spaces.    indentation
  112:0    error      Only use indent of 4 spaces.    indentation
  121:1    error      Only use indent of 4 spaces.    indentation
  129:0    error      Only use indent of 4 spaces.    indentation
  140:1    error      Only use indent of 4 spaces.    indentation
  148:0    error      Only use indent of 4 spaces.    indentation
  154:1    error      Only use indent of 4 spaces.    indentation
  160:0    error      Only use indent of 4 spaces.    indentation
  167:1    error      Only use indent of 4 spaces.    indentation
  173:0    error      Only use indent of 4 spaces.    indentation
  180:1    error      Only use indent of 4 spaces.    indentation
  184:0    error      Only use indent of 4 spaces.    indentation
  190:1    error      Only use indent of 4 spaces.    indentation
  197:0    error      Only use indent of 4 spaces.    indentation
  203:1    error      Only use indent of 4 spaces.    indentation
  208:0    error      Only use indent of 4 spaces.    indentation
  216:1    error      Only use indent of 4 spaces.    indentation
  226:0    error      Only use indent of 4 spaces.    indentation
  232:1    error      Only use indent of 4 spaces.    indentation
  234:1    error      Only use indent of 4 spaces.    indentation
  237:0    error      Only use indent of 4 spaces.    indentation
  239:1    error      Only use indent of 4 spaces.    indentation
```

```
  243:2    warning    Provide an error message for require()    error-reason
  246:0    error      Only use indent of 4 spaces.              indentation
  251:1    error      Only use indent of 4 spaces.              indentation
  260:0    error      Only use indent of 4 spaces.              indentation
  268:1    error      Only use indent of 4 spaces.              indentation
  276:0    error      Only use indent of 4 spaces.              indentation

contracts/tokens/Token.sol
  25:8    warning    Provide an error message for require()    error-reason

✖ 44 errors, 28 warnings found.
```

6.3 Surya

Surya is a utility tool for smart contract systems. It provides a number of visual outputs and information about the structure of smart contracts. It also supports querying the function call graph in multiple ways to aid in the manual inspection and control flow analysis of contracts.

Below is a complete list of functions with their visibility and modifiers:

Files Description Table

File Name	SHA-1 Hash
contracts/GuildBank.sol	d4329bc7836a1800eb2376da05f76f1783700b9a
contracts/Moloch.sol	8f55cc17fcf0488acdc9dd2261dca1e08f42c4ac
contracts/oz/ERC20.sol	6db943e86683ce536b8e75e79d7fb80a02b855ae
contracts/oz/IERC20.sol	f249341b598ed60fdb987fc6dd05b6cd15da7b6b
contracts/oz/ReentrancyGuard.sol	115a19532af141450ea30ad141aecb76b79035b4
contracts/oz/SafeMath.sol	b86ab5a6679fd597c3a0412d31080893beeb653a
contracts/test-helpers/Submitter.sol	7b29e3178cb4c7848851a8c92661a0e12fee7489
contracts/tokens/ClaimsToken.sol	11bb8b648de195efbca13df15e10b3e6a75fcab6
contracts/tokens/Token.sol	7c193d22ad069e368aba4fa9bc3d4c28e8e1973b

Contracts Description Table

Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
GuildBank	Implementation			
└	<Constructor>	Public ❗	🛑	
└	withdraw	Public ❗	🛑	onlyOwner
└	withdrawToken	Public ❗	🛑	onlyOwner
└	fairShare	Internal 🔒		
Moloch	Implementation	ReentrancyGuard		
└	<Constructor>	Public ❗	🛑	
└	submitProposal	Public ❗	🛑	nonReentrant
└	submitWhitelistProposal	Public ❗	🛑	nonReentrant
└	submitGuildKickProposal	Public ❗	🛑	nonReentrant
└	_submitProposal	Internal 🔒	🛑	
└	sponsorProposal	Public ❗	🛑	nonReentrant onlyDelegate
└	submitVote	Public ❗	🛑	nonReentrant

Contract	Type	Bases		
				onlyDelegate
└	processProposal	Public ❗	🛑	nonReentrant
└	processWhitelistProposal	Public ❗	🛑	nonReentrant
└	processGuildKickProposal	Public ❗	🛑	nonReentrant
└	_didPass	Internal 🔒	🛑	
└	_validateProposalForProcessing	Internal 🔒		
└	_returnDeposit	Internal 🔒	🛑	
└	ragequit	Public ❗	🛑	nonReentrant onlyMember
└	safeRagequit	Public ❗	🛑	nonReentrant onlyMember
└	_ragequit	Internal 🔒	🛑	
└	ragekick	Public ❗	🛑	nonReentrant
└	bailout	Public ❗	🛑	nonReentrant
└	cancelProposal	Public ❗	🛑	nonReentrant
└	updateDelegateKey	Public ❗	🛑	nonReentrant onlyShareholder
└	max	Internal 🔒		
└	getCurrentPeriod	Public ❗		NO ❗
└	getProposalQueueLength	Public ❗		NO ❗
└	getProposalFlags	Public ❗		NO ❗
└	canRagequit	Public ❗		NO ❗
└	canBailout	Public ❗		NO ❗
└	hasVotingPeriodExpired	Public ❗		NO ❗
└	getMemberProposalVote	Public ❗		NO ❗
ERC20	Implementation	IERC20		
└	totalSupply	Public ❗		NO ❗
└	balanceOf	Public ❗		NO ❗
└	allowance	Public ❗		NO ❗
└	transfer	Public ❗	🛑	NO ❗
└	approve	Public ❗	🛑	NO ❗
└	transferFrom	Public ❗	🛑	NO ❗
└	increaseAllowance	Public ❗	🛑	NO ❗
└	decreaseAllowance	Public ❗	🛑	NO ❗
└	_transfer	Internal 🔒	🛑	
└	_mint	Internal 🔒	🛑	
└	_burn	Internal 🔒	🛑	
└	_approve	Internal 🔒	🛑	
└	_burnFrom	Internal 🔒	🛑	

Contract	Type	Bases		
IERC20	Interface			
└	transfer	External ❗	🛑	NO ❗
└	approve	External ❗	🛑	NO ❗
└	transferFrom	External ❗	🛑	NO ❗
└	totalSupply	External ❗		NO ❗
└	balanceOf	External ❗		NO ❗
└	allowance	External ❗		NO ❗
ReentrancyGuard	Implementation			
└	<Constructor>	Internal 🔒	🛑	
SafeMath	Library			
└	mul	Internal 🔒		
└	div	Internal 🔒		
└	sub	Internal 🔒		
└	add	Internal 🔒		
Submitter	Implementation			
└	<Constructor>	Public ❗	🛑	
└	submitProposal	Public ❗	🛑	NO ❗
└	submitWhitelistProposal	Public ❗	🛑	NO ❗
└	submitGuildKickProposal	Public ❗	🛑	NO ❗
ERC20Detailed	Implementation	IERC20		
└	<Constructor>	Public ❗	🛑	
└	name	Public ❗		NO ❗
└	symbol	Public ❗		NO ❗
└	decimals	Public ❗		NO ❗
IClaimsToken	Interface			
└	withdrawFunds	External ❗	💸	NO ❗
└	availableFunds	External ❗		NO ❗
└	totalReceivedFunds	External ❗		NO ❗
ClaimsToken	Implementation	IClaimsToken, ERC20, ERC20Detailed		
└	<Constructor>	Public ❗	🛑	ERC20Detailed
└	transfer	Public ❗	🛑	NO ❗
└	transferFrom	Public ❗	🛑	NO ❗
└	totalReceivedFunds	External ❗		NO ❗
└	availableFunds	Public ❗		NO ❗
└	_registerFunds	Internal 🔒	🛑	
└	_calcUnprocessedFunds	Internal 🔒		
└	_claimFunds	Internal 🔒	🛑	

Contract	Type	Bases		
L	_prepareWithdraw	Internal 🔒	🛑	
ClaimsTokenERC20Extension	Implementation	IClaimsToken, ClaimsToken		
L	<Constructor>	Public ❗	🛑	ClaimsToken
L	withdrawFunds	al ❗	💵	NO ❗
L	tokenFallback	c ❗	🛑	onlyFundsToken
Token	Implementation	ERC20		
L	<Constructor>	Public ❗	🛑	
L	updateTransfersEnabled	External ❗	🛑	NO ❗
L	updateTransfersReturningFalse	External ❗	🛑	NO ❗
L	transfer	Public ❗	🛑	NO ❗

Legend

Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

Appendix 1 - Disclosure

ConsenSys Diligence ("CD") typically receives compensation from one or more clients (the "Clients") for performing the analysis contained in these reports (the "Reports"). The Reports may be distributed through other means, including via ConsenSys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond Solidity that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty.

CD makes the Reports available to parties other than the Clients (i.e., "third parties") – on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners. You agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that ConsenSys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you v use any content at any other web sites to which you link from the Reports. ConsenSys and CD assumes no responsibility for the use .nird party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of

any outcome generated by such software.

TIMELINESS OF CONTENT The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by ConsenSys and CD.