A CONSENSYS DILIGENCE AUDIT REPORT

Aave Governance Dao

Date	August 2020
Lead Auditor	John Mardlin
Co-auditors	Daniel Luca

1 Executive Summary

This report presents the results of our engagement with Aave to review their implementation of a Governance DAO which will enable token holders to vote on changes and upgrades to the Aave Protocol.

The review was conducted over the course of two weeks, from January 27th to February 7th by Daniel Luca and John Mardlin. A total of 15 person-days were spent.

During the first week, we focused our efforts on understanding the intention of the design (which is primarily documented by thorough natspec comments within the code), and defining the key risk factors and potential vulnerabilities requiring further investigation.

During the second week we focused more on detailed review of the code, and investigated potential vulnerabilities in edge cases of the voting mechanism. All our key findings have been addressed, and are reported below.

1.1 Scope and Objectives

Our review initially focused on the commit hash `c0f5ec54bf4d263f3421adbdec484bbc9f78b304`. During the course of our review, small changes were made to address our findings and comments, resulting in the most recent hash of `d6170403ed61f2f8cc4702604fd8aac1c773b6c0`. At a later date another small change was added in how the IPFS hash is stored which is identified by the commit hash `1ccda649ad2908223e0d962f609e462c725602ad`. The list of files in scope can be found in the Appendix.

Our primary objectives were to: 1. Ensure that the system is implemented consistently with the intended functionality, and without unintended edge cases. 2. Identify known vulnerabilities particular to smart contract systems, as outlined in our Smart Contract Best Practices, and the Smart Contract Weakness Classification Registry.

We also sought opportunities to improve the quality of the code either by reducing the complexity, or improving clarity and readability.

2 Recommendations and Advice

During the course of our review we identified a few possible improvements that are not security issues, but can bring value to the developers and the people who want to interact with the system.

2.1 Increase the number of tests

A good rule of thumb is to have 100% test coverage. This does not guarantee that security problems don’t exist, but it means that the desired functionality behaves as intended. Also the negative tests bring a lot of value because not allowing some actions to happen is also part of the desired behavior.

For example a specific functionality that was not previously tested was to move a proposal from the voting stage to the validating stage while having multiple voting options passing the minimum threshold.

Fixed

After the report was delivered the code coverage was increased to 100%.

```
$ npm run dev:coverage
[...]

-----------------------------------|----------|----------|----------|------
File                               |  % Stmts | % Branch |  % Funcs |  % Li
-----------------------------------|----------|----------|----------|------
 governance/                       |      100 |      100 |      100 |
  AavePropositionPower.sol         |      100 |      100 |      100 |
  AaveProtoGovernance.sol          |      100 |      100 |      100 |
  AssetVotingWeightProvider.sol    |      100 |      100 |      100 |
  GovernanceParamsProvider.sol     |      100 |      100 |      100 |
 interfaces/                       |      100 |      100 |      100 |
  IAaveProtoGovernance.sol         |      100 |      100 |      100 |
  IAssetVotingWeightProvider.sol   |      100 |      100 |      100 |
  IGovernanceParamsProvider.sol    |      100 |      100 |      100 |
  ILendingPoolAddressesProvider.sol |     100 |      100 |      100 |
  IProposalExecutor.sol            |      100 |      100 |      100 |
-----------------------------------|----------|----------|----------|------
All files                          |      100 |      100 |      100 |
-----------------------------------|----------|----------|----------|------
```

2.2 Do not change asset weights while a proposal is running

The asset weight feature is added to accommodate users voting with different assets on the same proposal. The asset weight normalizes the asset availability making different assets compatible with each other. Even though this functionality will be used in its minimal form in the beginning (for only one asset) it is important to state that changing the asset weight during a vote has some impact on the system.

The current way of implementing asset weight in voting allows for correct vote cancelling or replacing without creating any overflows or underflows. The only problem that can arise if the asset weight was changed, is to force the users that already voted to recast their vote to reflect the new weight. The users will want to do this if the newly set asset weight is higher than the previous one.

Because of the low number of projected proposals, this issue can be easily avoided.

2.3 Only whitelist validated assets

Because some of the functionality relies on correct token behavior, any whitelisted token should be audited in the context of this system. Problems can arise if a malicious token is whitelisted because it can block people from voting with that specific token or gain unfair advantage if the balance can be manipulated.

Make sure to audit any new whitelisted asset.

2.4 Review all comments

Review all comments and make sure they reflect what the code currently does.

As developers we often forget to update the comments when updating the code. Because the inaccurate comments do not affect us immediately sometimes we forget to update the comments. Make sure to review all of the comments after the code was frozen.

2.5 New proposals should be tested before deployed on the mainnet

Make sure you understand the risks of using `delegatecall` as well as contract storage layout when creating the `execute()` method on new proposals. Also the contract that has the `execute()` method should have the source code available and should be easy to read; all of the variables should be clearly available in the method itself not in the contract storage.

2.6 Execute proposals in the correct order

Because the proposal has a lot of power over the contracts it is very important to execute the proposals in the desired order. This can be avoided if there is only one proposal running at a time.

2.7 Enforce the cap to match the council member length

Add a `require` statement along the lines `require(cap == council.length)` in the `AavePropositionPower` token constructor. This will prevent unexpected consequences when creating a new proposal because not all of the tokens were minted.

2.8 Review the Code Quality recommendations in Appendix 1

Other comments related to readibility and best practices are listed in Appendix 1

3 System Overview

3.1 `AaveProtoGovernance`

`AaveProtoGovernance` is the core contract in the system. It implements a state machine for voting logic, which includes the following noteworthy functionality:

Proposal execution via DELEGATECALL: Voting is used to decide Yes or No on whether or not the `AaveProtoGovernance` contract should be allowed to DELEGATECALL the `execute` function on a particular contract address. This would typically result in a call to contract method in the AAVE protocol which is only accessible to the `AaveProtoGovernance` contract. Risks and trust implications of this design are discussed in the Security Specifications section.

Support for vote relaying: This enables token holders to sign their vote off-line, and submit it to the contract from another EOA acting as a relayer.

Token voting without lockups: Typical token voting schemes require depositing tokens to a contract during the voting period to prevent sybil voter fraud. To improve the UX `AaveProtoGovernance` uses an “optimistic” model: any votes submitted during the `Voting` period are counted proportional to the voters token balance. This is followed by a `Validation` period during which anyone may challenge a list of voters. If any voter’s balance is less than it was at the time of voting, all of their votes will be invalidated.

Minimum voting threshold: Each proposal defines a threshold of votes which must be met in order to pass. The voting period does not end until this threshold has been met, and the defined duration of the voting period has passed.

Multiple voting periods: If enough votes are challenged and invalidated during the `Validation` period to go below the threshold. The voting period begins again. This process can continue up to a maximum number of voting periods. Following the final voting and validation periods, whether or not the threshold is met, the result of the voting will be respected.

3.2 `AavePropositionPower`

In order to submit a proposal to the proposer must hold a sufficent quantity of the `AavePropositionPower` token. Upon creation, a list of addresses is provided which will each receive one token. If an address is listed multiple times, it can receive multiple tokens.

3.3 `AssetVotingWeightProvider`

The `AssetVotingWeightProvider` which holds a list of other ERC20 compliant tokens that may also be used to vote, and their relative voting weights. Tokens may only be added at the time of initialization, but the contract has an `owner` which may update the weight of each token at anytime.

3.4 `GovernanceParamsProvider`

The `GovernanceParamsProvider` holds three important parameters:

1. The address of the `AssetVotingWeightProvider`
2. The address of the `ExecutiveGovernanceAsset`
3. The `govAssetThreshold` which defines the amount of the `ExecutiveGovernanceAsset` required to submit a new proposal.

This contract has an `owner` who may update these properties at any time.

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 listed below with their respective power.

- Proposers can
  - Create new proposals at any time
- Voters can
  - Vote on active proposals
  - Cancel their vote
  - Replace their vote
  - Send their vote to a relayer plus the associated signature
- Voters as a result of passed proposals can
  - Set or change the token asset that is needed to have proposition power
  - Set the minimum power a proposer needs to have in order to create a new proposal
  - Set the voting weight for the tokens that are used to vote with
- Relayers can
  - Submit votes for the voters as long as they have the correct signature and vote data
  - Cancel votes for the voters as long as they have the correct signature and vote data
  - Replace votes for the voters as long as they have the correct signature and vote data
- Any other user can
  - Read any contract parameters, including
    - voting weights for any whitelisted asset
    - proposal data
    - proposal votes for each option
    - proposal vote of a lend owner
    - proposal leading vote option
  - Verify
    - The nonce of a voter for any proposal
    - A relayer’s action based on the signature receiver from the voter
  - Challenge any votes trying to reveal double voting behavior
  - Try to move a proposal from the voting state to the validating state
  - Resolve a proposal, effectively executing the code attached to the proposal, if the “yes” option wins.

4.2 Trust Model

In any smart contract system, it's important to identify what trust is expected/required between various actors. This system is fairly decentralized with the token owners having a lot of decision power, as well as being able to stop malicious proposals.

Considering all this, we identified the following trusted points for users to be aware of before they interact with the system:

1. ***Initial* Ownership of `AssetVotingWeightProvider` and `GovernanceParamsProvider`:** The variables defined in the `AssetVotingWeightProvider` and `GovernanceParamsProvider` contracts are critical to the outcome of voting. These variables can be set by the `owner` address, which is initialized to the deployer's address. The Aave team clarified to us that their plan is to transfer ownership of these contracts to the `AaveProtoGovernance`, meaning that any important change to the system must first pass a full voting cycle. Once completed, users can easily verify that this point of centralization has been removed.
2. **Proposal creation:** Proposals may only be submitted by holder of the `AavePropositionPower`. This is kept in balance by the token holders ability to vote to reject proposals.
3. **Voting:** Users of the Aave protocol place some trust in votes to reject malicous proposals. However, if the majority of the token owners want to attack the system, the other legitimate actors have time to exit the system based on the minimum time set in the contract. This minimum time is determined based on the voting blocks duration and the validating blocks duration. Both of the time periods need to be at least the minimum set in the contract `MIN_STATUS_DURATION`. We believe this gives sufficient time for the users to exit the system in any way they find necessary.

Our general perspective is that the system is very decentralized while remaining powerfully flexible.

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 VotingMachine - tryToMoveToValidating can lock up proposals Major ✓ Fixed

Resolution

Fixed per our recommendation.

Description

After a vote was received, the proposal can move to a validating state if any of the votes pass the proposal's `precReq` value, referred to as the minimum threshold.

code/contracts/governance/VotingMachine.sol:L391

```
tryToMoveToValidating(_proposalId);
```

Inside the method `tryToMoveToValidating` each of the vote options are checked to see if they pass `precReq`. In case that happens, the proposal goes into the next stage, specifically `Validating`.

code/contracts/governance/VotingMachine.sol:L394-L407

```
/// @notice Function to move to Validating the proposal in the case the last v
///  was done before the required votingBlocksDuration passed
/// @param _proposalId The id of the proposal
function tryToMoveToValidating(uint256 _proposalId) public {
    Proposal storage _proposal = proposals[_proposalId];
    require(_proposal.proposalStatus == ProposalStatus.Voting, "VOTING_STATU
    if (_proposal.currentStatusInitBlock.add(_proposal.votingBlocksDuration)
        for (uint256 i = 0; i <= COUNT_CHOICES; i++) {
            if (_proposal.votes[i] > _proposal.precReq) {
                internalMoveToValidating(_proposalId);
            }
        }
    }
}
```

The method `internalMoveToValidating` checks the proposal's status to be `Voting` and proceeds to moving the proposal into `Validating` state.

code/contracts/governance/VotingMachine.sol:L270-L278

```
/// @notice Internal function to change proposalStatus from Voting to Validati
/// @param _proposalId The id of the proposal
function internalMoveToValidating(uint256 _proposalId) internal {
    Proposal storage _proposal = proposals[_proposalId];
    require(_proposal.proposalStatus == ProposalStatus.Voting, "ONLY_ON_VOTI
    _proposal.proposalStatus = ProposalStatus.Validating;
    _proposal.currentStatusInitBlock = block.number;
    emit StatusChangeToValidating(_proposalId);
}
```

The problem appears if multiple vote options go past the minimum threshold. This is because the loop does not stop after the first found option and the loop will fail when the method `internalMoveToValidating` is called a second time.

code/contracts/governance/VotingMachine.sol:L401-L405

```
for (uint256 i = 0; i <= COUNT_CHOICES; i++) {
    if (_proposal.votes[i] > _proposal.precReq) {
        internalMoveToValidating(_proposalId);
    }
}
```

The method `internalMoveToValidating` fails the second time because the first time it is called, the proposal goes into the `Validating` state and the second time it is called, the require check fails.

code/contracts/governance/VotingMachine.sol:L274-L275

```
require(_proposal.proposalStatus == ProposalStatus.Voting, "ONLY_ON_VOTING_S
_proposal.proposalStatus = ProposalStatus.Validating;
```

This can lead to proposal lock-ups if there are enough votes to at least one option that pass the minimum threshold.

Recommendation

After moving to the `Validating` state return successfully.

```
function tryToMoveToValidating(uint256 _proposalId) public {
    Proposal storage _proposal = proposals[_proposalId];
    require(_proposal.proposalStatus == ProposalStatus.Voting, "VOTING_STATU
    if (_proposal.currentStatusInitBlock.add(_proposal.votingBlocksDuration)
        for (uint256 i = 0; i <= COUNT_CHOICES; i++) {
            if (_proposal.votes[i] > _proposal.precReq) {
                internalMoveToValidating(_proposalId);
                return; // <- this was added
            }
        }
    }
}
```

An additional change can be done to `internalMoveToValidating` because it is called only in `tryToMoveToValidating` and the parent method already does the check.

```
/// @notice Internal function to change proposalStatus from Voting to Validat
/// @param _proposalId The id of the proposal
function internalMoveToValidating(uint256 _proposalId) internal {
    Proposal storage _proposal = proposals[_proposalId];
    // The line below can be removed
    // require(_proposal.proposalStatus == ProposalStatus.Voting, "ONLY_ON_VO
    _proposal.proposalStatus = ProposalStatus.Validating;
    _proposal.currentStatusInitBlock = block.number;
    emit StatusChangeToValidating(_proposalId);
}
```

5.2 VotingMachine - verifyNonce should only allow the next nonce Major ✓ Fixed

Resolution

Fixed per our recommendation.

Description

When a relayer calls `submitVoteByRelayer` they also need to provide a nonce. This nonce is cryptographicly checked against the provided signature. It is also checked again to be higher than the previous nonce saved for that voter.

code/contracts/governance/VotingMachine.sol:L232-L239

```
/// @notice Verifies the nonce of a voter on a proposal
/// @param _proposalId The id of the proposal
/// @param _voter The address of the voter
/// @param _relayerNonce The nonce submitted by the relayer
function verifyNonce(uint256 _proposalId, address _voter, uint256 _relayerNc
    Proposal storage _proposal = proposals[_proposalId];
    require(_proposal.voters[_voter].nonce < _relayerNonce, "INVALID_NONCE")
}
```

When the vote is saved, the previous nonce is incremented.

code/contracts/governance/VotingMachine.sol:L387

```
voter.nonce = voter.nonce.add(1);
```

This leaves the opportunity to use the same signature to vote multiple times, as long as the provided nonce is higher than the incremented nonce.

Recommendation

The check should be more restrictive and make sure the consecutive nonce was provided.

```
require(_proposal.voters[_voter].nonce + 1 == _relayerNonce, "INVALID_NONCE"
```

5.3 VoteMachine - Cancelling vote does not increase the nonce Minor ✓ Fixed

Resolution

Fixed per our recommendation.

Description

A vote can be cancelled by calling `cancelVoteByRelayer` with the proposal ID, nonce, voter's address, signature and a hash of the sent params.

The parameters are hashed and checked against the signature correctly.

The nonce is part of these parameters and it is checked to be valid.

code/contracts/governance/VotingMachine.sol:L238

```
require(_proposal.voters[_voter].nonce < _relayerNonce, "INVALID_NONCE");
```

Once the vote is cancelled, the data is cleared but the nonce is not increased.

code/contracts/governance/VotingMachine.sol:L418-L434

```
if (_cachedVoter.balance > 0) {
    _proposal.votes[_cachedVoter.vote] = _proposal.votes[_cachedVoter.vote].
    _proposal.totalVotes = _proposal.totalVotes.sub(1);
    voter.weight = 0;
    voter.balance = 0;
    voter.vote = 0;
    voter.asset = address(0);
    emit VoteCancelled(
        _proposalId,
        _voter,
        _cachedVoter.vote,
        _cachedVoter.asset,
        _cachedVoter.weight,
        _cachedVoter.balance,
        uint256(_proposal.proposalStatus)
    );
}
```

This means that in the future, the same signature can be used as long as the nonce is still higher than the current one.

Recommendation

Considering the recommendation from issue https://github.com/ConsenSys/aave-governance-dao-audit-2020-01/issues/4 is implemented, the nonce should also increase when the vote is cancelled. Otherwise the same signature can be replayed again.

5.4 Possible lock ups with SafeMath multiplication Minor

Acknowledged

Resolution

The situation described is unlikely to occur, and does not justify mitigations which might introduce other risks.

Description

In some cases using SafeMath can lead to a situation where a contract is locked up due to an unavoidable overflow.

It is theoretically possible that both the `internalSubmitVote()` and `internalCancelVote()` functions could become unusable by voters with a high enough balance, if the asset weighting is set extremely high.

Examples

This line in `internalSubmitVote()` could overflow if the voter's balance and the asset weight were sufficiently high:

code/contracts/governance/VotingMachine.sol:L379

```
uint256 _votingPower = _voterAssetBalance.mul(_assetWeight);
```

A similar situation occurs in `internalCancelVote()`:

code/contracts/governance/VotingMachine.sol:L419-L420

```
_proposal.votes[_cachedVoter.vote] = _proposal.votes[_cachedVoter.vote].sub(
_proposal.totalVotes = _proposal.totalVotes.sub(1);
```

Recommendation

This could be protected against by setting a maximum value for asset weights. In practice it is very unlikely to occur in this situation, but it could be introduced at some point in the future.

Appendix 1 - Code Quality Recommendations

A.1.1 Naming of `proposalId` variable [Done]

The name of the `proposalID` variable defined in `newProposal()` is slightly misleading. It actually represents the length of the proposals array, and is one greater than the true proposal ID.

A.1.2 Incomplete comment [Done]

The natspec `@notice` comment on `internalCancelVote()` says:

```
Internal function to cancel a vote. This function is called from the external cancel vote
functions, by relayers and directly by voters
```

.

For completeness, this comment should also mention `challengeVoters()` and `internalSubmitVote()` as calling functions.

A.1.3 Pin Solidity Version [Done]

Most of the files use a floating pragma statement `pragma solidity ^0.5.0;`. We recommend settling on the most recent version of Solidity `0.5.x` or at least the latest version `^0.5.x`.

A.1.4 Use consistent ordering when passing variables [Done]

The `submitVoteByRelayer` and `cancelVoteByRelayer` receive their arguments in one order, but pass them to `abi.encodePacked` in a different order. Maintaining their order would improve readability.

```
function submitVoteByRelayer(
        uint256 _proposalId,
        uint256 _nonce,
        uint256 _vote,
        address _voter,
        IERC20 _asset,
        bytes calldata _signature,
        bytes32 _paramsHashByVoter)
    external {
        validateRelayAction(
            keccak256(abi.encodePacked(_proposalId, _vote, _voter, _asset, _
```

```
function cancelVoteByRelayer(
        uint256 _proposalId,
        uint256 _nonce,
        address _voter,
        bytes calldata _signature,
        bytes32 _paramsHashByVoter)
    external {
        Proposal storage _proposal = proposals[_proposalId];
        require(_proposal.proposalStatus == ProposalStatus.Voting, "VOTING_S
        validateRelayAction(
            keccak256(abi.encodePacked(_proposalId, _voter, _nonce)),
```

A.1.5 Be consistent about skipping SafeMath where possible. [Acknowledged]

The OpenZeppelin SafeMath library is used in most, but not all arithmetic operations, in particular for reducing the length of the `proposals` array by 1 to get `_proposalId`. This is safe, but there are other cases where `.add(1)` or `.mul(2)` are used unnecessarily.

We suggest either always using SafeMath, or always not using when it is obviously unnecessary.

A.1.6 Consider breaking up long SafeMath chains [Done]

Several statements combine the use of SafeMath and nested struct member accesses. Breaking these expressions up over several lines would improve their readability.

For example, before:

```
    _proposal.votes[_cachedVoter.vote] = _proposal.votes[_cachedVoter.vote].
```

and after:

```
_proposal.votes[_cachedVoter.vote] = _proposal.votes[_cachedVoter.vote].sub
        _cachedVoter.balance.mul(
            _cachedVoter.weight
        )
    );
```

A.1.7 Consider emitting the newly created proposal ID [Done]

When a new proposal is created an event is emitted with the details of the newly created proposal.

```
event ProposalCreated(
    address indexed creator,
    bytes32 indexed proposalType,
    uint256 executiveReputationOfCreator,
    uint256 precReq,
    uint256 maxMovesToVotingAllowed,
    uint256 votingBlocksDuration,
    uint256 validatingBlocksDuration,
    address proposalExecutor
);
```

The event does not contain the proposal ID.

It will help the web UI and other developers monitoring the contract if the proposal ID is included in the event emitted.

```
event ProposalCreated(
    uint256 proposalId, // <- add something like this
    address indexed creator,
    bytes32 indexed proposalType,
    uint256 executiveReputationOfCreator,
    uint256 precReq,
    uint256 maxMovesToVotingAllowed,
    uint256 votingBlocksDuration,
    uint256 validatingBlocksDuration,
    address proposalExecutor
);
```

Appendix 2 - Files in Scope

Our review covered the following files at the outset:

File	git hash-object
contracts/governance/AssetVotingWeightProvider.sol	2f7f62047d04db1fe7e10edb0f543d000b090c87
contracts/governance/ExecutiveReputation.sol	N/A
contracts/governance/GovernanceParamsProvider.sol	dab9996e4b55f8b440bab1226cdddb3c263c25d4

File	git hash-object
contracts/governance/VotingMachine.sol	N/A
contracts/interfaces/IAssetVotingWeightProvider.sol	5c70d5e2fc68756f2c09666e89435b9e354ec3d6
contracts/interfaces/IFeeProvider.sol	N/A
contracts/interfaces/IGovernanceParamsProvider.sol	acc9dabd181ba6f521a1051c342c918add576490
contracts/interfaces/ILendingPoolAddressesProvider.sol	d6a84a6410577e2d1c717d3ebad25bd370cc06d5
contracts/interfaces/IProposalExecutor.sol	30fe300977f1b2eaa5378f4b47c3ea1162c2c4af

During the course of our review, the files and contracts were renamed and updated to the following:

File	git hash-object
contracts/interfaces/IAaveProtoGovernance.sol	886ac68d5ab1e239037368a6e38fdab252c23dd1
contracts/interfaces/IAssetVotingWeightProvider.sol	5c70d5e2fc68756f2c09666e89435b9e354ec3d6
contracts/interfaces/IGovernanceParamsProvider.sol	acc9dabd181ba6f521a1051c342c918add576490
contracts/interfaces/ILendingPoolAddressesProvider.sol	d6a84a6410577e2d1c717d3ebad25bd370cc06d5
contracts/interfaces/IProposalExecutor.sol	30fe300977f1b2eaa5378f4b47c3ea1162c2c4af
contracts/governance/AavePropositionPower.sol	5fb7632adbff0585687554ba7b1b8c23b3de7170
contracts/governance/AaveProtoGovernance.sol	9def8e674b4435fc8b2101403f958c8a2cfb72ac

File	git hash-object
contracts/governance/AssetVotingWeightProvider.sol	2f7f62047d04db1fe7e10edb0f543d000b090c87
contracts/governance/GovernanceParamsProvider.sol	dab9996e4b55f8b440bab1226cdddb3c263c25d4

Appendix 3 - Artifacts

This section contains some of the artifacts generated during our review by automated tools, the test suite, etc. If any issues or recommendations were identified by the output presented here, they have been addressed in the appropriate section above.

A.3.1 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.

Below is the raw output of the MythX vulnerability scan.

Report for /Users/primary/Projects/Audits/aave/aave-governance-dao-audit-2020-01/code-final/contracts/governance/AaveProtoGovernance.sol
https://dashboard.mythx.io/#/console/analyses/41988310-fcf8-474c-9932-930479138753

```
╒════════╤═══════════════════════════════════════════════╤═════
│   Line │ SWC Title                                     │ Seve
╞════════╪═══════════════════════════════════════════════╪═════
│    203 │ Weak Sources of Randomness from Chain Attributes │ Medi
├────────┼───────────────────────────────────────────────┼─────
│    270 │ Weak Sources of Randomness from Chain Attributes │ Medi
├────────┼───────────────────────────────────────────────┼─────
│    280 │ Weak Sources of Randomness from Chain Attributes │ Medi
├────────┼───────────────────────────────────────────────┼─────
│    404 │ Weak Sources of Randomness from Chain Attributes │ Medi
```

477	Weak Sources of Randomness from Chain Attributes	Medi
478	Weak Sources of Randomness from Chain Attributes	Medi
278	Presence of unused variables	Medi
279	Presence of unused variables	Medi
280	Presence of unused variables	Medi
1	Floating Pragma	Low
96	Assert Violation	Low
345	Assert Violation	Low
66	Assert Violation	Low
242	Assert Violation	Low
48	Assert Violation	Low
474	Assert Violation	Low
62	Assert Violation	Low
402	Assert Violation	Low
88	Assert Violation	Low
500	Assert Violation	Low
79	Assert Violation	Low
514	Assert Violation	Low
159	Assert Violation	Low

368	Assert Violation	Low
109	Assert Violation	Low
327	Assert Violation	Low
82	Assert Violation	Low
449	Assert Violation	Low

A.3.2 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:

A.3.3 Contracts Description Table

Contract	**Type**	**Bases**		
L	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
AavePropositionPower	Implementation	ERC20Capped, ERC20Detailed		
L		Public ❗️	🛑	ERC20Capped ERC20Detailed

Contract	Type	Bases		
AaveProtoGovernance	Implementation	IAaveProtoGovernance		
L		Public ❗️	🛑	NO❗️
L		External ❗️	💵	NO❗️
L	newProposal	External ❗️	🛑	NO❗️
L	verifyParamsConsistencyAndSignature	Public ❗️		NO❗️
L	verifyNonce	Public ❗️		NO❗️
L	validateRelayAction	Public ❗️		NO❗️
L	internalMoveToVoting	Internal 🔒	🛑	
L	internalMoveToValidating	Internal 🔒	🛑	
L	internalMoveToExecuted	Internal 🔒	🛑	
L	submitVoteByVoter	External ❗️	🛑	NO❗️
L	submitVoteByRelayer	External ❗️	🛑	NO❗️
L	cancelVoteByVoter	External ❗️	🛑	NO❗️
L	cancelVoteByRelayer	External ❗️	🛑	NO❗️
L	internalSubmitVote	Internal 🔒	🛑	

Contract	Type	Bases		
L	tryToMoveToValidating	Public ❗️	🛑	NO❗️
L	internalCancelVote	Internal 🔒	🛑	
L	challengeVoters	External ❗️	🛑	NO❗️
L	resolveProposal	External ❗️	🛑	NO❗️
L	getLimitBlockOfProposal	Public ❗️		NO❗️
L	getLeadingChoice	Public ❗️		NO❗️
L	getProposalBasicData	External ❗️		NO❗️
L	getVoterData	External ❗️		NO❗️
L	getVotesData	External ❗️		NO❗️
L	getGovParamsProvider	External ❗️		NO❗️
AssetVotingWeightProvider	Implementation	Ownable, IAssetVotingWeightProvider		
L		Public ❗️	🛑	NO❗️
L	getVotingWeight	Public ❗️		NO❗️
L	setVotingWeight	External ❗️	🛑	onlyOwner
L	internalSetVotingWeight	Internal 🔒	🛑	

Contract	Type	Bases		
GovernanceParamsProvider	Implementation	Ownable, IGovernanceParamsProvider		
└		Public ❗️	🛑	NO❗️
└	setPropositionPowerThreshold	External ❗️	🛑	onlyOwner
└	setPropositionPower	External ❗️	🛑	onlyOwner
└	setAssetVotingWeightProvider	External ❗️	🛑	onlyOwner
└	internalSetPropositionPowerThreshold	Internal 🔒	🛑	
└	internalSetPropositionPower	Internal 🔒	🛑	
└	internalSetAssetVotingWeightProvider	Internal 🔒	🛑	
└	getPropositionPower	External ❗️		NO❗️
└	getPropositionPowerThreshold	External ❗️		NO❗️
└	getAssetVotingWeightProvider	External ❗️		NO❗️

A.3.4 Legend

Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

A.3.5 Tests Suite

Below is the output generated by running the test suite:

```
$ npm run dev:test

> aave-protocol-dao@1.0.0 dev:test /Users/primary/Projects/Audits/aave/aave-
> buidler test

Compiling...
Downloading compiler version 0.5.13
Compiled 37 contracts successfully


  AavePropositionPower
    ✓ Has a non-null address after deployment
    ✓ Has correct metadata
    ✓ It's not possible to mint more tokens because of the cap
    ✓ The cap of the AavePropositionPower is correct
    ✓ The Council members have 1000000000000000000 AavePropositionPower each

  AaveProtoGovernance basic tests
    ✓ Has a non-null address after deployment
    ✓ Creation of a new proposal fails when trying with an address with no 
    ✓ govParamsProvider is registered properly
    ✓ Checks the data of a newly created proposal in the AaveProtoGovernance
    ✓ Checks the data of a secondly created proposal in the AaveProtoGoverna

  AssetVotingWeightProvider
    ✓ Has a non-null address after deployment
    ✓ Has correct voting weights for the test voting assets

  GovernanceParamsProvider
    ✓ Has a non-null address after deployment
    ✓ Has the correct aavePropositionPower registered
    ✓ Has the correct propositionPowerThreshold registered
    ✓ Has the correct assetVotingWeightProvider registered

  LendingPoolAddressesProvider
    ✓ Has a non-null address after deployment
    ✓ The owner is signers[0]

  TestVotingAssetA
    ✓ Has a non-null address after deployment
```

```
    ✓ Has a non-null address after deployment
    ✓ Has correct metadata
    ✓ Mints tokens to 2 voters (91ms)
    ✓ Transfer tokens from voter 1 to voter 3 (88ms)

  AaveProtoGovernance - Scenarios
    Voting and Cancel directly
      ✓ Voter 1 receives 5M tokens from minting (93ms)
      ✓ Voter 2 receives 600K tokens from minting (83ms)
      ✓ Voter 1 votes (193ms)
      ✓ Voter 1 votes (380ms)
      ✓ Voter 1 cancels vote (211ms)
      ✓ (REVERT EXPECTED) Trigger resolveProposal() (89ms)
      ✓ Voter 1 votes (179ms)
      ✓ Fast forward blocks (6288ms)
      ✓ Voter 2 votes (193ms)
      ✓ (REVERT EXPECTED) Voter 2 votes (66ms)
      ✓ (REVERT EXPECTED) Trigger resolveProposal() (89ms)
      ✓ Fast forward blocks (6033ms)
      ✓ Trigger resolveProposal() (155ms)
      ✓ (REVERT EXPECTED) Trigger resolveProposal() (78ms)
    Voting and Cancel through relayers
      ✓ Voter 1 receives 5M tokens from minting (80ms)
      ✓ Voter 2 receives 600K tokens from minting (79ms)
      ✓ Voter 1 votes through relayer (224ms)
      ✓ Voter 1 votes through relayer (450ms)
      ✓ Voter 1 cancels vote through relayer (271ms)
      ✓ Voter 1 votes through relayer (215ms)
      ✓ Fast forward blocks (6093ms)
      ✓ Voter 2 votes through relayer (226ms)
      ✓ Voter 2 votes (reverting) through relayer (97ms)
      ✓ (REVERT EXPECTED) Trigger resolveProposal() (87ms)
      ✓ Fast forward blocks (6067ms)
      ✓ Trigger resolveProposal() (161ms)
    Voting directly and through relayers
      ✓ Voter 1 receives 5M tokens from minting (87ms)
      ✓ Voter 2 receives 600K tokens from minting (84ms)
      ✓ Voter 1 votes through relayer (212ms)
      ✓ Fast forward blocks (6218ms)
      ✓ Voter 2 votes directly (190ms)
      ✓ (REVERT EXPECTED) Challenge voter 1 (65ms)
      ✓ Fast forward blocks (6175ms)
      ✓ Trigger resolveProposal() (81ms)
    Voting with double-voting attempt through relayers
      ✓ Voter 1 receives 10 tokens from minting (86ms)
      ✓ Voter 2 receives 6 tokens from minting (83ms)
      ✓ Voter 1 votes through relayer (218ms)
      ✓ Voter 1 transfer tokens to Voter 3 (103ms)
      ✓ Fast forward blocks (6341ms)
      ✓ Voter 3 votes through relayer (232ms)
      ✓ Challenge voter 1 double voting (59ms)
      ✓ (REVERT EXPECTED) Challenge voter 1 double voting (69ms)
      ✓ (REVERT EXPECTED) Challenge voter 1 double voting (70ms)
```

```
    4 small voters voting No and a whale Yes
      ✓ Voter 1 receives 300K tokens from minting (88ms)
      ✓ Voter 2 receives 100K tokens from minting (92ms)
      ✓ Voter 3 receives 500K tokens from minting (86ms)
      ✓ Voter 4 receives 200K tokens from minting (88ms)
      ✓ Voter 5 receives 6M tokens from minting (87ms)
      ✓ Voter 1 votes through relayer (225ms)
      ✓ Voter 2 votes directly (185ms)
      ✓ Voter 3 votes directly (174ms)
      ✓ Voter 4 votes directly (191ms)
      ✓ (REVERT EXPECTED) Trigger resolveProposal() (81ms)
      ✓ Fast forward blocks (6311ms)
      ✓ Voter 5 votes directly (224ms)
      ✓ Fast forward blocks (6922ms)
      ✓ Trigger resolveProposal() (175ms)


  79 passing (1m)
```

Appendix 4 - 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 may use any content at any other web sites to which you link from the Reports. ConsenSys and CD assumes no responsibility for the use of third 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.