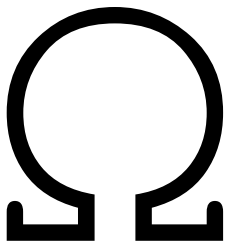
DXdao Governance Contracts

Final Audit Report

March 6, 2023



Team Omega

teamomega.eth.limo

Summary **4**
Scope of the Audit **4**
Resolution **5**
Methods Used **5**
Disclaimer **6**
Severity definitions **6**
Findings **7**
General 7
G1. Use Solidity Custom Errors instead of strings [low] [resolved] 7
G2. Remove the use of SafeMath [low] [resolved] 7
G3. Some tests are failing [low] [resolved] 7
DAOAvatar.sol 7
DAOController.sol 7
C1. Scheme registered on initialize is not usable [medium] [resolved] 7
C2. Counter of schemes that can manage schemes is not updated correctly [low] [resolved] 8
C3. Permissioning of scheme is not effective [info] [resolved] 8
C4. Duplicated check that a scheme is registered when unregistering [info] [resolved] 9
C5. Use delete instead of reassignment to clean value of a struct [info] [resolved] 9
C6. Proposals lists queries may run out of gas [info] [resolved] 9
C7. Remove unused variable [info] [resolved] 10
C8. endProposal allows schemes to remove proposals of other schemes [info] [resolved] 10
C9. Start proposal can assign existing proposal to another scheme [info] [resolved] 11
C10. onlyRegisteredScheme is unnecessary and used inconsistently [info] [resolved] 11
C11. getSchemesWithManageSchemesPermissionsCount can be removed [info] [not resolved] 11
DAOReputation.sol 12
R1. Snapshot is called unnecessarily too often [low] [partially resolved] 12
Scheme.sol 12
S1. Incomplete check that certain functions don’t transfer value [low] [resolved] 12
S2. Use constructor instead of initializer [low] [not resolved] 13
S3. Wrong contract name used in errors [info] [resolved] 13
S4. Call the voting machine propose directly [info] [resolved] 13
S5. Avoid saving description hash and title on chain [info] [not resolved] 14
S6. Replace proposals list with events [info] [not resolved] 14
S7. Remove requirement that numberOfOptions is equal to 2 [info] [resolved] 14
S8. Proposal expects multiple choice support in the voting machine [low] [resolved] 15
AvatarScheme.sol and WalletScheme.sol 15
A1. Calls to setETHPermissionUsed can be frontrun to block execution of proposals [medium] [resolved] 15
A2. Follow Checks/Effects/Interactions pattern in executeProposal [low] [resolved] 16

A3. Check that scheme can make avatar calls will block proposal execution [info] [resolved] 16
A4. Proposal state set to ExecutionSucceeded multiple times [info] [resolved] 17
DXDVotingMachine.sol 17
VM1. Anyone can steal all staking tokens [high] [resolved] 17
VM2. Proposals with multiple choices break the contract logic [high] [resolved] 18
VM3. Owner can steal all staking tokens of users [high] [resolved] 18
VM4. Proposal that reverts on execution will be stuck [medium] [resolved] 18
VM5. Staking and voting with signature doesn't verify the chain ID [medium] [resolved] 19
VM6. Inconsistent usage of precision in percentages [low] [resolved] 19
VM7. Boosted execution bar is ignored if its value is bigger than the execution bar [info] [resolved] 20
VM8. Remove zero check on address recovered from signature [info] [resolved] 20
VM9. Remove unused lines [info] [resolved] 20
VM10. Remove unnecessary requirement for withdrawal [info] [resolved] 20
VM11. Mark score function external [info] [resolved] 21
VM12. Mark avatarOwner immutable [info] [resolved] 21
VM13. Stakers who lost can redeem their stake [high] [resolved] 21
VM14. Stake reward belonging to the DAO will be stuck in the contract [medium] [resolved] 21
VM15. Wrong counting of the preBoostedProposalsCounter [medium] [resolved] 22
VM16. Inconsistent behavior when claiming DAO bounty [info] [resolved] 22
VM17. Missing check of success for transferFrom call [low] [resolved] 23
VM18. Redeem fails when DAO bounty can't be paid [low] [not resolved] 23
DXDVotingMachineCallbacks.sol 23

Summary

DXdao has asked Team Omega to audit the contracts that define the behavior of the DXdao DXgovernance contracts.

We found **4 high severity issues** - these are issues that can lead to a loss of funds, and are essential to fix. We classified **6** issues as “medium” - this is an issue we believe you should definitely address. In addition, **10** issues were classified as “low”, and **24** issues were classified as “info” - we believe the code would improve if these issues were addressed as well.

All high and medium severity issues were addressed, and almost all other issues as well.

Severity	**Number of issues**	**Number of resolved issues**
High	4	4
Medium	6	6
Low	12	9
Info	23	20

Scope of the Audit

We audited the code from the following repository and commit:

The scope of the audit concerns the changes made to the governance mechanism. This code is currently developed in the following repository and branch:

`https://github.com/DXgovernance/dxdao-contracts`

We audited the following commit hash:

```
099caf1da40a965b34a13278776fad523635cf74
```

In particular, the scope of the audit regards the files:

```
contracts/dao/DAOAvatar.sol
contracts/dao/DAOController.sol
```

```
contracts/dao/DAOReputation.sol
contracts/dao/schemes/Scheme.sol
contracts/dao/schemes/AvatarScheme.sol
contracts/dao/schemes/WalletScheme.sol
contracts/dao/votingMachine/DXDVotingMachine.sol
contracts/dao/votingMachine/DXDVotingMachineCallbacks.sol
```

Resolution

The current report is the result of a number of iterations over the code, where we audited the following commits:

```
83642286610ff6d0adcc92e91585ea2e8d69fd09
44f7247d7d837cd5a552e0b8f5b2da5568f73336
03a138c4d0c161028ef541e31a454cc53e2cdf55
5b5c3e36ff14c59ba9f1c5db33f17ad6b6e87548
```

And finally:

```
dc3e2ec6a995b2d37a4f4a1203b60bce4242f2c2
```

We have reviewed the fixes and reported our findings at each issue

Methods Used

Code Review

We manually inspected the source code to identify potential security flaws.

The contracts were compiled, deployed, and tested in a test environment.

Automatic analysis

We have used static analysis tools to detect common potential vulnerabilities. The tools have detected a number of low severity issues, concerning mostly the variables naming and external calls, were found. We have included any relevant issues below in the appropriate parts of the report.

Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

Severity definitions

High	Vulnerabilities that can lead to loss of assets or data manipulations.
Medium	Vulnerabilities that are essential to fix, but that do not lead to assets loss or data manipulations
Low	Issues that do not represent direct exploit, such as poor implementations, deviations from best practice, high gas costs, etc
Info	Matters of opinion

Findings

General

G1. Use Solidity Custom Errors instead of strings [low] [resolved]

The use of custom errors, that are available in Solidity since version 0.8.4, instead of error strings, will reduce gas costs. Cf. https://blog.soliditylang.org/2021/04/21/custom-errors/
Recommendation: Use Custom Errors instead of error strings.
Severity: Low
Resolution: This issue was resolved as recommended.

G2. Remove the use of SafeMath [low] [resolved]

Many contracts across the project still import and use the `SafeMath` library of OpenZeppelin, even though math operations are already safe by default in Solidy 0.8, which makes this entirely unnecessary.
Recommendation: Remove `SafeMath` from the project.
Severity: Low
Resolution: This issue was resolved as recommended.

G3. Some tests are failing [low] [resolved]

In the version provided, 37 out of 163 tests are failing.
Recommendation: Fix the tests.
Severity: Low
Resolution: This issue was resolved.

DAOAvatar.sol

We did not find any issues with this contract

DAOController.sol

C1. Scheme registered on initialize is not usable [medium] [resolved]

In the `initialize` function, a given scheme address is added to the schemes mapping with the value of `paramsHash` set to `bytes32(0)`.

If the scheme uses the `DXDVotingMachine` to decide on proposals, this value for the `paramsHash` will make the scheme unusable. This is because calling `scheme.proposeCalls(..)` will call `votingMachine.propose(..)` with `bytes32(0)` as the value for `paramsHash`. This call will then revert on line 1233 where it reads:

```
//Check parameters existence.
require(parameters[_paramsHash].queuedVoteRequiredPercentage >= 50);
```

Recommendation: Allow specifying the `paramsHash` that is passed to the `initialize` function.
Severity: Medium
Resolution: This issue was resolved as recommended.

C2. Counter of schemes that can manage schemes is not updated correctly [low] [resolved]

The contract defines a `schemesWithManageSchemesPermission` counter which is intended to track the number of schemes that can manage the schemes register. This counter is used to ensure that there is always at least one scheme with permission to register new schemes to the DAO.
However, this counter is not always updated correctly. The `registerScheme` function can be used to remove the permission to manage schemes. In this scenario, the counter is not subtracted from to reflect that change, which means its value might be wrong, and the controller might remain with no registering schemes.
Recommendation: On `registerScheme`, subtract from the `schemesWithManageSchemesPermission` counter in case the permission to register schemes is removed from a scheme that already has it, and ensure that after the change, the counter will not reach 0.
Severity: Low
Resolution: This issue was resolved as recommended. However, in line 139, the condition checks that either the scheme is not registered, or that it cannot manage other schemes, but this is unnecessary since if a scheme can manage other scheme it is also necessarily registered, so the condition can be simplified as follows:

```
if (!scheme.canManageSchemes && _canManageSchemes) {
```

C3. Permissioning of scheme is not effective [info] [resolved]

Schemes that are registered in the DAOControllor have the following boolean flags that control if the schemes are allowed, respectively, to manage the list of schemes, and to call the Avatar directly:

```
canManageSchemes;
```

```
canMakeAvatarCalls;
```

All registered schemes can mint or burn reputation, independently of whether these flags are set or not. Controlling the reputation distribution effectively means controlling all schemes that use the DXDVotingMachine with that reputation distribution - a malicious scheme could assign all reputation to a compromised account and pass any proposals in all schemes that use that reputation by an absolute majority vote.
This means that in practice, the scheme permissions do not provide any real security.
Recommendation: Consider removing these permissions, as they can give a false sense of security to implementers, and do not serve any real security purpose.
Severity: Info
Resolution: This issue was resolved. There's now a new permission defined that is needed to mint and burn reputation, and can be set separately.

C4. Duplicated check that a scheme is registered when unregistering [info] [resolved]

On line 118 in `unregisterScheme`, the condition checks that the scheme to be unregistered is currently registered. This is also checked earlier, on line 114, which will `return` if the scheme is not registered. So this second check is not necessary.
Recommendation: Remove the first condition of the `if` statement on line 118 that checks if the scheme to remove is registered.
Severity: Info
Resolution: This issue was resolved as recommended.

C5. Use delete instead of reassignment to clean value of a struct [info] [resolved]

On line 128, when unregistering a scheme, its data is removed by assigning all its values to 0. Yet instead of assigning the values to 0, it will be more efficient and save some gas to just use the `delete` keyword.
Recommendation: Replace lines 128 to 133 with: `delete schemes[_scheme];`
Severity: Info
Resolution: This issue was resolved as recommended.

C6. Proposals lists queries may run out of gas [info] [resolved]

Both the `getActiveProposals` and `getInactiveProposals` functions run a loop on an unbounded array, which may run out of gas if the arrays get very large. In the case of `getInactiveProposals`,

once it reaches that gas limit, it will never be usable again, as there is no option to remove proposals from the array.
The fact that these functions use an unbounded amount of gas and might eventually be unusable means that no on-chain logic should depend on them.
Recommendation: Remove the `getActiveProposals` and `getInactiveProposals` functions, as well as the related `startProposal` and `endProposal` functions which help manage these and the related variables.
Instead, for keeping track of active and inactive proposals off-chain (for example for the user interface), define appropriate indexed events, which will be cheaper and safer to use.
Severity: Info
Resolution: This issue was resolved as recommended

C7. Remove unused variable [info] [resolved]

The contract defines a `schemesAddresses` array but never uses or even populates it, so it should be removed.
Recommendation: Remove the `schemesAddresses` variable.
Severity: Info
Resolution: This issue was resolved as recommended.

C8. endProposal allows schemes to remove proposals of other schemes [info] [resolved]

The function `endProposal` requires that `msg.sender` is a registered scheme (or, if it is not, that the proposal to be ended belongs to a non-registered scheme):

```
schemes[msg.sender].isRegistered ||
(!schemes[schemeOfProposal[_proposalId]].isRegistered &&
      activeProposals.contains(_proposalId)),
```

The requirement does not check if the caller is actually the scheme that originally started the proposal.
Recommendation: Replace the condition with:

```
schemeOfProposal[_proposalId]] == msg.sender ||
(!schemes[schemeOfProposal[_proposalId]].isRegistered &&
      activeProposals.contains(_proposalId)),
```

Compare also issue C6, where we propose to remove this function altogether.
Severity: Info
Resolution: This issue was resolved. Management of the proposals list was moved to the voting machine.

C9. Start proposal can assign existing proposal to another scheme [info] [resolved]

The function `startProposal` lets any scheme assign itself as the proposer of a certain proposal ID in the `schemeOfProposal` mapping. This means any scheme can also assign itself as the proposer of the proposals of any other scheme. This could then lead to the original scheme failing to execute, because only the scheme mapped as the owner is able to call `endProposal`.
Recommendation: Check that a proposal ID is not already used by another scheme when calling the `startProposal` function is not a good solution, because it will allow a scheme to block proposals from another scheme. We believe you can remove the logic entirely, as suggested in C6. If however you decide to keep track of proposals in the Controller, make sure that different schemes do not get in each other's way, for example by storing `keccak256(msg.sender, proposalId)` in the array of `activeProposals`.
Severity: Info
Resolution: This issue was resolved. Management of the proposals list was moved to the voting machine.

C10. onlyRegisteredScheme is unnecessary and used inconsistently [info] [resolved]

The `onlyRegisteredScheme` modifier is no longer necessary in the current version of the code, as there is no action that a registered scheme with no further permissions is allowed to execute in the current system. It can be safely removed from the code.
Recommendation: Remove the `onlyRegisteredScheme` modifier from the code.
Severity: Info
Resolution: This issue was resolved as recommended.

C11. getSchemesWithManageSchemesPermissionsCount can be removed [info] [not resolved]

`schemesWithManageSchemesPermission` is public, and the compiler will create a getter. So the function `getSchemesWithManageSchemesPermissionsCount` can be removed.
Recommendation: Remove the function `getSchemesWithManageSchemesPermissionsCount` from the code.
Severity: Info
Resolution: This issue was not resolved.

DAOReputation.sol

R1. Snapshot is called unnecessarily too often [low] [partially resolved]

The documentation states that the reputation "uses a snapshot mechanism to keep track of the reputation at the moment of each proposal creation." Yet this is not what actually happens.
In the `DAOReputation.sol` code, a snapshot is taken every single time there is a change in the reputation, and with batched changes, the snapshot is taken after every individual change in the batch. For example, in `mintMultiple`, on line 43ff:

```
for (uint256 i = 0; i < _accounts.length; i++) {
      _mint(_accounts[i], _amount[i]);
      _snapshot();
}
```

This is unnecessarily wasteful, and does not correspond to the documentation as quoted above.
Recommendation: Follow the behavior specified in the documentation and only take a snapshot of the reputation when a new proposal is created. I.e., let schemes call the reputation (through the controller), to take a snapshot whenever a new proposal is created.
Severity: Low
Resolution: This issue was partially resolved. The contract's logic was moved to the `utils/ERC20/ERC20SnapshotRep.sol` file. Now, in the `mintMultiple` and `burnMultiple` functions, the call to take the snapshot was moved to the end after the loop so the snapshot is only taken at the end of the operation. However, some more gas could be saved by following our recommendation to take the snapshot only on proposal creation, and not after every reputation.

Scheme.sol

S1. Incomplete check that certain functions don't transfer value [low] [resolved]

On line 148, there is a check that proposed calls which target ERC20-style `transfer` and `approve` functions cannot send ETH value with the transaction.
This check is unnecessary, because these functions are not `payable` in ERC20 contracts and so will revert. For contracts that do have `payable` functions, the check is also easy to bypass by calling `transferFrom` instead of `transfer`, or (if implemented) by calling `increaseAllowance` and `decreaseAllowance` instead of `approve`.
Recommendation: This check (and the entire loop, lines 144 to 153) is both unnecessary and incomplete, and so we suggest removing it.

Severity: Low
Resolution: This issue was resolved as recommended.

S2. Use constructor instead of initializer [low] [not resolved]

Configuration of a newly deployed scheme is done by calling the `initialize` function, i.e. the configuration of the contract is done in a separate step, instead of in the constructor.
This pattern makes the call to `initialize` vulnerable to front-running attacks (even if it is not clear to what gain). It also costs some extra gas on deployment, as it does not allow to declare certain state variables `immutable`.
This pattern is common in upgradeable contracts, but as the Scheme contracts are not upgradeable, there seems to be no reason to not do the configuration in a constructor function.
Recommendation: Move the `initialize` logic to a `constructor` function. Or, if that is not possible, create a `SchemeFactory` contract that deploys and initializes a new scheme in a single step.
Severity: Low
Resolution: This issue was not resolved.

S3. Wrong contract name used in errors [info] [resolved]

All error messages in the contract mention `WalletScheme`. This contract is not `WalletScheme`, it is just called `Scheme`.
Recommendation: Update error messages in the contract to say `Scheme` instead of `WalletScheme`, or replace them with custom errors as suggested in G1.
Severity: Info
Resolution: This issue was resolved as recommended.

S4. Call the voting machine propose directly [info] [resolved]

The call on line 164 to the `propose` function of the voting machine is done using `functionCall` instead of calling the `propose` function directly. The code will be more readable and gas efficient by just calling the `propose` function:

```
proposalId = IVotingMachine(votingMachine).propose(_totalOptions, ...)
```

Recommendation: Replace the use of `functionCall` by calling the `propose` function of the voting machine directly.
Severity: Info

Resolution: This issue was resolved as recommended.

S5. Avoid saving description hash and title on chain [info] [not resolved]

The `Proposal` struct currently saves two `string` fields, `title` and `descriptionHash`, for each proposal. As no other contracts need to read these data, there is no need to save these data on-chain. Moreover, as these fields can be of arbitrary length, gas costs for storing these data can be arbitrarily high.
Recommendation: Do not save the `title` and `descriptionHash` on chain; instead emit an event with these data so they can be indexed and read off-chain.
Severity: Info
Resolution: This issue was not resolved.

S6. Replace proposals list with events [info] [not resolved]

The `proposalsList` is not used by the contract logic, nor by any other contract in the system. As we assume it is only meant to be utilized by the UI, it will be cheaper to remove it and use events to track the proposals instead.
Recommendation: Use events instead of keeping the `proposalsList` variable.
Severity: Info
Resolution: This issue was not resolved.

S7. Remove requirement that numberOfOptions is equal to 2 [info] [resolved]

On line 157 in the `proposeCalls` function, it is checked that the value of `_totalOptions` is equal to 2.

```
require(_totalOptions == 2, "WalletScheme: The total amount of options
should be 2");
```

This check can be omitted if the votingMachine is an instance of DXDVotingMachine - as the implementation of the `propose` function there just ignores the argument. However, the check makes it impossible to use this Scheme implementation with (future) voting machines that have more than 2 options, which seems an unnecessary limitation.
Recommendation: Remove the check on line 157
Severity: Info
Resolution: This issue was resolved

S8. Proposal expects multiple choice support in the voting machine [low] [resolved]

The function `proposeCalls` takes a `totalOptions` argument that can be any integer value. A version of the proposal will then be created in the associated voting machine with the following call:

```
proposalId = votingMachine.propose(totalOptions, voteParams, msg.sender,
address(avatar));
```

In the current implementation of the `propose` function in `VotingMachine.sol` and `DXDVotingMachine.sol`, the `totalOptions` argument is ignored - instead, a default value of 2 is taken. This means that users can only vote for the first two options, and therefore none of the other options can be executed.
This is clearly not as intended, and may lead to confusion,and, depending on the UI implementation, may potentially mislead proposers or voters.
Recommendation: Check explicitly that the `votingMachine` is compatible with the Scheme. In this case, we recommend to change the `propose` function in `VotingMachine.sol` and `DXDVotingMachine.sol` and instead of ignoring the `totalOptions` argument, check that `totalOptions == NUM_OF_OPTIONS` and revert if it is not.
Severity: Low
Resolution: This issue was resolved. The voting machine's `propose` function will now revert if the `totalOptions` passed are not equal to 2.

AvatarScheme.sol and WalletScheme.sol

A1. Calls to setETHPermissionUsed can be frontrun to block execution of proposals [medium] [resolved]

The function `setETHPermissionUsed` on the `PermissionRegistry` can be called by anyone. If this permission is set, a malicious attacker could frontrun a call to `executeProposal` from the `votingMachine` and call `setETHPermission` with the `valueTransfered` equal to the `permission.valueAllowed`.
Any further calls in the same block to `setETHPermissionUsed` with a non-zero value for `valueTransfered` will now revert, and it will not be possible to execute the proposal in the current block.
Recommendation: Currently, `setETHPermissionUsed` in the `PermissionRegistry` keeps track of how much ETH was sent in the whole block in a global way. Consider instead keeping a counter that is limited to `msg.sender`.
Severity: Medium

Resolution: This issue was resolved. Now only the owner of the `PermissionRegistry` can call `setETHPermissionUsed` with a `from` address of another user, while all others can only call it for themselves.

A2. Follow Checks/Effects/Interactions pattern in executeProposal [low] [resolved]

As a general best practice, it is recommended to do state changes before doing external calls. This works as a safeguard against re-entrancy attacks, but also guarantees that if the external contract does any callbacks to the original contract, it can expect that the original contract is in a consistent state. The implementation of the `executeProposal` function does not follow this pattern.
The `executeProposal` function does implement a basic reentrancy guard in the form of the `executingProposal` flag. This however only avoids reentrancy into the `executeProposal` function itself. Other contracts may expect the Scheme to be in a consistent state (for example, they may expect a proposal to be executed exactly when its `state` is equal to `ExecutionSucceeded`), and are not protected by the `executingProposal` flag.
Recommendation: Make sure that the external calls happen as late as possible in the function execution. Specifically, move the lines where the `proposal.state` is set, and where `controller.endProposal` is called, to a place before the external calls.
Severity: Low
Resolution: This issue was resolved. The `executeProposal` function now changes the state of the proposal before making external calls. Also since the VotingMachine now no longer reverts if the `executeProposal` function reverts, a new `finishProposal` function was added to ensure the proposal state can be changed even if the `executeProposal` function was reverted.

A3. Check that scheme can make avatar calls will block proposal execution [info] [resolved]

In `AvatarScheme.sol`, the `require` call on line 60 checks that the scheme is approved to make avatar calls before proceeding with the execution. At best, this is unnecessary, since if a proposal has passed but the scheme is not approved to make avatar calls, the execution will fail anyway.
However, whether the proposal did not pass, it will be impossible to execute it, and it will be stuck in the state machine until the problem is resolved (see also issue VM4).
A similar observation applies to the `WalletScheme`.
Recommendation: Remove the `require` check on line 60.
Severity: Info
Resolution: This issue was resolved as recommended.

A4. Proposal state set to ExecutionSucceeded multiple times [info] [resolved]

On line 116 in AvatarScheme.sol, and line 104 in WalletScheme.sol, the `state` of the proposal is set to `ProposalState.ExecutionSucceeded`, but since this is within the loop that makes each call, this state change will unnecessarily happen multiple times if a proposal has more than a single call.
Recommendation: Move updating the state of the proposal from line 116 to outside the loop, preferably to an earlier place in the execution process (cf. A2)
Severity: Info
Resolution: This issue was resolved. The change of the proposal state was moved to a new `finishProposal` function so it only happens once.

DXDVotingMachine.sol

This file was renamed to `VotingMachine.sol`.

VM1. Anyone can steal all staking tokens [high] [resolved]

The `redeemDaoBounty` function has been changed to pay the dao bounty from the staking funds that are deposited in the contract. In the previous version this bounty was paid from the Avatar of the organization the proposal belongs to. In the current version, there is no place where the organization transfers the dao bounty into the voting machine contract, so the bounty is paid from funds that belong to the stakes of other users. If that happens, these users can lose funds (they might lose their stake even on winning proposals).
An attacker can steal the entire token balance of the contract by passing a proposal with a minimum DAO bounty that is equal to the token balance. This works as follows: the attacker calls `setParameters` and registers a “paramsHash” with `minimumDaoBounty` set to the contract’s balance. She then creates a new proposal with an arbitrary organization, stakes 1 token, passes the proposal (the proposal is decided on the basis of a reputation distribution she controls), and then calls `redeemDaoBounty` which will send the attacker the amount `minimumDaoBounty`, stealing all stakes of all other users of the voting machine.
Recommendation: Return the previous logic of the contract which transferred the bounty from the Avatar.
Severity: High
Resolution: This issue was resolved. However, there are issues with the new `redeem` function which we added at the end of the report, like VM13 and VM16.

VM2. Proposals with multiple choices break the contract logic [high] [resolved]

The contract's `proposeMultipleChoice` function allows to create proposals that offer multiple choices (beyond just the regular binary YES and NO).
However, such proposals are not properly supported in the contract. In the whole contract, there are issues with multiple choice logic, like in voting, staking, redeeming, score calculation, and getting proposal status.
To give one specific example, consider the staking logic. The contract allows users to call `stake` with any of the valid options. However, in the logic of the Genesis Protocol, proposals are boosted/unboosted on the basis of the"score" of the proposal, which only counts the fraction of stakes for choice 1 relative to choice 2, and ignores all stakes of other users.
Recommendation: Remove the option to create proposals with more than 2 options until you implement support for such options.
Severity: High
Resolution: This issue was resolved as recommended. It is no longer possible to create proposals with multiple options.

VM3. Owner can steal all staking tokens of users [high] [resolved]

The contract defines a `claimStakingTokens` function which lets the `avatarOwner` take all staking tokens from the contract. These tokens belong to the users who staked them, which puts the users funds at risk. In addition, removing the entire token balance from the contract makes functions that depend on sending staking tokens, such as the `executeBoosted` and `redeem` functions always revert.
Recommendation: Remove the `claimStakingTokens` in its current form.
Severity: High
Resolution: This issue was resolved as recommended.

VM4. Proposal that reverts on execution will be stuck [medium] [resolved]

In line 1129 in the `_execute()` function, there is a call to `executeProposal` on the scheme implementation, which is called when an outcome is decided:

```
ProposalExecuteInterface(proposal.callbacks).executeProposal(_proposalId,
proposal.winningVote);
```

This call can revert for many different reasons. If this call does revert, then so will the call to `_execute`, and the proposal will effectively be stuck in the DXDVotingMachine state machine in its present state.

Specifically, what could happen (either by accident or by intent) that a proposal is `Boosted`, has collected enough votes to be accepted (i.e. the number of "YES" votes is above the `boostedExecutionBar`), but any attempt to actually execute the proposal fails. In this case, the proposal will remain in the `Boosted` state, and there is no way to remove the proposal from that state. This is a problem for several reasons, such as that it will make it more expensive to boost subsequent proposals, and funds staked on the proposal will be stuck as well.
Recommendation: Separate the DXD State Machine logic from the actual execution of the proposal: do not call `executeProposal` as part of the `_execute` function, but set a flag instead, and implement a separate function that executes the proposal only if this flag is set.
Severity: Medium
Resolution: This issue was resolved.
The call to `executeProposal` now is wrapped in a `try … catch` statement so it can fail without making the transaction revert.
In the current implementation, if the call to `executeProposal` fails, it will not be possible to try to execute the proposal a second time - instead, the proposal will remain in a failed state. This is not a problem for the protocol per se, but may not be the desired behavior.

VM5. Staking and voting with signature doesn't verify the chain ID [medium] [resolved]

The contract allows both staking and voting to be done by signature, yet nowhere in these signatures does the signer commit to a specific chain ID. This means signatures are possible to replay on different chains where the contract exists.
Recommendation: Add chain ID to the signed data both for staking and voting with signature.
Severity: Medium
Resolution: This issue was resolved.

VM6. Inconsistent usage of precision in percentages [low] [resolved]

In the `_execute` function, the value of `queuedVoteRequiredPercentage` is expressed with a precision of `100` (i.e. 13% corresponds to `13`), while `_boostedVoteRequiredPercentage` is expressed with a precision of `10000` (i.e. 13% corresponds with `1300`).
This is confusing, and it will be easy for users to make mistakes.
Recommendation: Decide on a precision for representing percentages and use that consistently.
Severity: Low
Resolution: This issue was resolved as recommended.

VM7. Boosted execution bar is ignored if its value is bigger than the execution bar [info] [resolved]

In case a proposal crosses the execution bar while boosted, the boosted execution bar will not be checked. There is, however, no guarantee that the boosted execution bar is lower than the regular execution bar.
Recommendation: Require that the execution bar will always be higher than or equal to the boosted execution bar.
Severity: Info
Resolution: This issue was resolved as recommended.

VM8. Remove zero check on address recovered from signature [info] [resolved]

On line 626 there is a check that the address recovered from a signature is not the 0 address. It is considered impossible that `recover` will return the 0 address, and there is no reason to check for this case.
Recommendation: Remove the `require` check on line 626.
Severity: Info
Resolution: This issue was resolved as recommended.

VM9. Remove unused lines [info] [resolved]

The lines 1012 and 1013 just mention 2 variables but don't do anything and could just be removed.
Recommendation: Remove lines 1012 and 1013.
Severity: Info
Resolution: This issue was resolved as recommended.

VM10. Remove unnecessary requirement for withdrawal [info] [resolved]

The `require` on line 647 limits withdrawal of deposited funds to only callers that have the `voteGas` configured to more than 0. Yet this is unnecessary, and may make it harder to withdraw for an organization which has first deleted their `voteGas` before withdrawing their balance.
Recommendation: Remove the `require` on line 647.
Severity: Info
Resolution: This issue was resolved.

VM11. Mark score function external [info] [resolved]

The `score` function is not used in the contract, and can be declared `external`.
Recommendation: Declare the `score` function as `external`.
Severity: Info
Resolution: This issue was resolved. The `score` function is now used also internally and so can remain `public`.

VM12. Mark avatarOwner immutable [info] [resolved]

The `avatarOwner` variable is only ever set in the constructor and could be marked `immutable` to save gas.
Recommendation: Mark the `avatarOwner` variable as `immutable`.
Severity: Info
Resolution: This issue was resolved. The `avatarOwner` variable was removed.

VM13. Stakers who lost can redeem their stake [high] [resolved]

The `redeem` function was refactored and will now allow stakers who staked for the losing outcome to redeem their entire staked position. This breaks the staking logic, and will cause a double spending of the funds as both winners and losers now have a claim on the stake of the losers.
Recommendation: Delete line 372 and the comment above it.

```
// Default reward is the stakes amount
reward = staker.amount;
```

Severity: High
Resolution: This issue was resolved. The code was changed so that stakers who lost cannot redeem their stake, which resolves the issue.

VM14. Stake reward belonging to the DAO will be stuck in the contract [medium] [resolved]

The DAO no longer has a way to claim its earnings from its initial stake against each proposal, as these lines were removed from the `redeem` function. This means that (if VM13 is fixed) that for each failed

proposal, a part of the staking reward is allocated to the DAO, but this reward cannot be claimed and will be stuck in the contract.
Recommendation: Add a way for the DAO to claim its earnings, or if it is intentional that the DAO cannot claim earnings, reduce its stake from the calculation in line 384 to avoid tokens from being stuck, for example like this:

```
      reward =
          (staker.amount * totalStakesWithoutDaoBounty) /
          (proposalStakes[proposalId][proposal.winningVote] -
proposal.daoBounty);
```

Severity: Medium
Resolution: This issue was resolved. The DAO will now receive its own share of the rewards as part of the `redeem()` logic.

VM15. Wrong counting of the preBoostedProposalsCounter [medium] [resolved]

In line 839 the `preBoostedProposalsCounter` is reduced by one, although in case `MAX_BOOSTED_PROPOSALS` was reached, the proposal state remains as pre-boosted, which means the counter will be wrong, and might cause underflow error which will not be fixable.
Recommendation: Only reduce from the counter if the proposal state was changed from `preBoosted` to another value.
Severity: Medium
Resolution: This issue was resolved as recommended.

VM16. Inconsistent behavior when claiming DAO bounty [info] [resolved]

Currently, when claiming the extra DAO bounty for a successful proposal, if the DAO didn't give enough allowance for the voting machine, the staker will just lose the DAO bounty. But if there is enough allowance but not enough balance, the `redeem` transaction will revert. This seems inconsistent and may not be as intended.
Recommendation: We recommend implementing a consistent behavior for when the transfer of the `daoBounty` fails - for example with a `try .. catch` statement. Consider also adding a way for users to claim their DAO bounty at a later time, if the claiming failed when calling `redeem`.
Severity: Info
Resolution: This issue was resolved. The call to `redeem` will now revert in case any transfer of tokens fails. Note that this also means that the redeem might fail in case the dao does not have the enough funds (or did not approve enough) to pay the DAO bounty, in which case no redeeming will be possible.

VM17. Missing check of success for transferFrom call [low] [resolved]

In the `redeem` function, there is a call to the `stakingToken.transferFrom` function, but its return value is not checked for success. This could lead to a failed transfer going unhandled.
Recommendation: Use OpenZeppelin's `safeTransferFrom`. Or verify the success of the transfer by checking if the call returns true.
Severity: Low
Resolution: The issue was resolved as recommended.

VM18. Redeem fails when DAO bounty can't be paid [low] [not resolved]

The call to `redeem` will now revert in case any transfer of tokens fails. This means that the `redeem` might fail in case the DAO does not have the enough funds (or did not approve enough) to pay the DAO bounty, in which case no redeeming will be possible, and even the original amount staked will be stuck in the contract until the DAO can make its DAO bounty payments.
Recommendation: Implement a way for stakers to get their own deposit back and the amount of the winning, but without paying out the DAO bounty. A quick fix would be to check not revert when there are not enough funds to pay out the DAO bounty, while a more complete solution would allow for users to claim their DAO bounty at a later stage as well
Severity: Low
Resolution: This issue was not resolved.

DXDVotingMachineCallbacks.sol

This file was renamed to `VotingMachineCallbacks.sol`

We did not find any issues with this contract.