CONSENSYS
Diligence

Dandelion Organizations Audit

Date	December 2019
Auditors	Alexander Wade

1 Summary

ConsenSys Diligence conducted a security audit of 1Hive's Dandelion org template and supporting apps. Dandelion orgs are a DAO template that function similarly to MolochDAO,

and are comprised of a suite of modular Aragon apps that can be used in any DAO.

2 Audit Scope

This audit covered the following files:

File Name	SHA-1 Hash
token-request-app/contracts/lib/AddressArrayLib.sol	56997c6cfa74369087731aedcc4e032b29d400c6
redemptions-app/contracts/lib/ArrayUtils.sol	6b184d37d5cca932da3d962a2fdc938389778a76
dandelion-org/contracts/DandelionOrg.sol	bdfeab2a56304ce4bbbbd1a43cff49a2103732e0
dandelion-voting-app/contracts/DandelionVoting.sol	3d131eb7a72d4a44e3ae387b7f12a255f38f5d6a
delay-app/contracts/Delay.sol	cbc8a9cdca4c7cb8a3afecb33d36d2ef2324c903
redemptions-app/contracts/Redemptions.sol	a26ba97e9c71367955a2b34c735ba3b1559cda26
time-lock-app/contracts/TimeLock.sol	b30260b66f876ea0462fc72ee108e085594b1123
token-oracle/contracts/TokenBalanceOracle.sol	0e16f0bbe4870234b264f824d943e08d50c9bc59
token-request-app/contracts/TokenRequest.sol	2fa54ed301858564806416823d5539624a8418c8
token-request-app/contracts/lib/UintArrayLib.sol	1226ee39e2eaca8320647bbaa0fcf2d577665ef6

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

3 Key Observations/Recommendations

- A Dandelion organization functions similarly to MolochDAO:
  - The org has a native asset which represents “shares”. Holders of the asset are conferred voting power within the org.
  - Shares are implemented via `MiniMeToken`, which tracks historical balances.
  - Shares are non-transferrable
  - Shares are minted when new members are accepted to the org
  - Shares are burned when redeemed for a proportion of the org’s assets
  - Members can vote to accept new members into the organization
  - Votes are executed in order of their creation, and between a vote’s passing and execution is a grace period during which members of the org can exit their shares (according to a few requirements)
- Noteable differences:
  - Dandelion orgs are much more configurable
  - Dandelion proposals execute aragonOS EVMScripts, which are batches of arbitrary calls to any destination. As such, Dandelion orgs can execute a much wider range of actions through votes:
    - Modifying voting-specific parameters, like support required, minimum quorum, number of blocks between votes, and vote execution delay period
    - Notably, vote duration (`durationBlocks`) cannot be changed
    - Managing permissions within the organization’s ACL
    - Managing registered applications within the organization
    - Modifying execution methods for executed proposals (via the `EVMScriptRegistry`)
    - Modifying which assets are redeemable when burning shares
    - Modifying which assets are accepted by `TokenRequest` requests for entry into the org
    - … and more.

- MolochDAO is very minimalist. Dandelion's complexity is much higher due to its wider range of actions and configurations, as well as its highly modular design.
  - The wider range of actions allowed in Dandelion orgs allow the organization to pass votes that radically change the structure and function of the organization. It is possible to execute votes that do almost anything, including dissolving the organization. Members of Dandelion orgs will need to fully understand proposals, and carefully consider the entire range of actions executed by a successful vote.
- Dandelion orgs do not require an existing member to submit a proposal for a new member to join. Instead, a new member can create a request to join (via `TokenRequest.createTokenRequest`).
  - Requests offer a deposit of tokens in exchange for the minting of new org tokens
  - The requesting user can cancel their request at any point before its finalization. Cancelled requests involve a refund of the deposited tokens.
  - Requests can be finalized through the passing of a vote in `DandelionVoting`. On finalization, the requestor's deposited tokens are transferred to the org's Vault, and the `TokenManager` mints shares for the requestor.
- Discouraging spam in MolochDAO is implemented by requiring a 10 ETH deposit from the member sponsoring a request to join. In Dandelion, spam is discouraged via the `TimeLock` app, which requires anyone creating a vote to lock up a specified token for a configurable amount of time. The more votes created by a single address before their timelocked tokens are released, the higher the spam penalty is for each vote. The increase in spam penalty is calculated using the configurable `spamPenaltyFactor`.
- `DandelionVoting` introduces a configurable voting quorum and support threshhold, while MolochDAO requires only a simple majority to pass a vote.

4 Security Specification

This section describes, **from a security perspective**, some of 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 investigated by the audit team.

4.1 Actors

The relevant actors are as follows:

- **Deployer**: Responsible for instantiating the organization with its base settings via `DandelionOrg`.
- **Organization Member**: Holds non-transferrable org tokens, which represent a redeemable portion of the org's assets. Responsible for voting on new proposals to carry out a wide range of actions within the org.
- **Requesting Member**: Creates a request to be minted org tokens via `TokenRequest`. Can cancel request before finalization. On finalization, deposit tokens are moved to the org's vault and the requesting member is minted an amount of org tokens.

4.2 Trust Model

In any smart contract system, it's important to identify what trust is expected/required between various actors. For this audit, we established the following trust model:

- The parameters set up during instantiation (`newTokenAndBaseInstance` and `installDandelionApps`) are important to the behavior of many critical systems in the org:
  - `_redemptionsRedeemableTokens` describes the redeemable assets held by the organization. The Deployer should ensure that this is a list of valid ERC20 tokens (optionally including "ETH" by using address(0x00))
  - `_tokenRequestAcceptedDepositTokens` describes the tokens accepted for deposit by requesting members. The Deployer should ensure that this is a list of valid ERC20 tokens (optionally including "ETH" by using address(0x00))
    - The Deployer should be careful when including tokens in this list that are not included in `_redemptionsRedeemableTokens`. If tokens are deposited that are not directly redeemable, they must first be added to `Redemptions` by way of a vote.
    - Additionally, `TokenRequest.acceptedDepositTokens` and `Redemptions.redeemableTokens` have different maximum allowed sizes. The Deployer should note that a token must be redeemable in order for org members to withdraw it.
  - `_timelockToken` is the address of the token users will lock when creating votes via `TimeLock.forward`. It is configured with an array of settings (`uint[3] _timelockSettings`), which corresponds to the duration of the lock, the base amount of tokens to lock, and the factor by which the successive lock penalty

increases. It is crucial that each org choose reasonable settings for `TimeLock`, as successful spamming of `DandelionVoting.newVote` has the potential to halt progress in an org due to longer and longer delays between relevant votes.
        - Additionally, the token chosen for `TimeLock` should be non-transferrable for maximum spam penalty efficacy.
    - `uint64[5] _votingSettings` corresponds to settings in `DandelionVoting`. It is crucial that the Deployer choose reasonable values for their org's purpose:
        - `supportRequiredPct` is the percent of yeas in all casted votes required to pass a vote (expressed as a proportion of 10**18)
        - `minAcceptQuorumPct` is the percent of yeas in the total possible votes required to pass a vote (expressed as a proportion of 10**18)
        - `durationBlocks` is the number of blocks a vote will be open for voters. Deployers should note that of all org settings, this value is not changeable. Therefore, it is very important to choose a reasonable value.
        - `bufferBlocks` is the number of blocks between the start of each subsequent vote.
        - `executionDelayBlocks` is the number of blocks that consist of the delay period between a vote's passing and its actual execution.
- The relative security of an organization's assets depends heavily on the careful consideration of the ramifications of each vote by the org's members. With great power comes great responsibility; batched administrative actions within an org are able to fundamentally change the behavior of many aspects of the org and demand careful attention before being ratified.
- Orgs can be extended through votes by the addition of new apps. These new apps will undoutedly have their own set of permissions and security properties and should be carefully reviewed by voters to ensure maximum compatibility with the goals of the existing org.
- Dandelion orgs and their supporting apps make heavy use of aragonOS. While aragonOS has been audited, a full review of the system is not in scope for this audit.

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 `TimeLock` spam prevention can be bypassed Critical ✓ Addressed

Resolution
This was addressed in commit aa6fc49fbf3230d7f02956b33a3150c6885ee93f by parsing the input evm script and ensuring only a single external call is made. Additionally, commit 453179e98159413d38196b6a5373cdd729483567 added `TimeLock` and `token` to the script runner blacklist.

Description

The `TimeLock` app is a forwarder that requires users to lock some token before forwarding an EVM callscript. Its purpose is to introduce a "spam penalty" to hamper repeat actions within an Aragon org. In the context of a Dandelion org, this spam penalty is meant to stop users from repeatedly creating votes in `DandelionVoting`, as subsequent votes are buffered by a configurable number of blocks (`DandelionVoting.bufferBlocks`). Spam prevention is important, as the more votes are buffered, the longer it takes before "non-spam" votes are able to be executed.

By allowing arbitrary calls to be executed, the `TimeLock` app opens several potential vectors for bypassing spam prevention.

Examples

- Using a callscript to transfer locked tokens to the sender

By constructing a callscript that executes a call to the lock token address, the sender execute calls to the lock token on behalf of `TimeLock`. Any function can be executed, making it possible to not only `transfer` "locked" tokens back to the sender, but also steal other users' locked tokens by way of `transfer`.

- Using a batched callscript to call `DandelionVoting.newVote` repeatedly

Callscripts can be batched, meaning they can execute multiple calls before finishing. Within a Dandelion org, the spam prevention mechanism is used for the `DandelionVoting.newVote` function. A callscript that batches multiple calls to this function can execute `newVote` several times per call to `TimeLock.forward`. Although multiple new votes are created, only one spam penalty is incurred, making it trivial to extend the buffer imposed on "non-spam" votes.

- Using a callscript to re-enter `TimeLock` and `forward` or `withdrawAllTokens` to itself

A callscript can be used to re-enter `TimeLock.forward`, as well as any other `TimeLock` functions. Although this may not be directly exploitable, it does seem unintentional that many of the `TimeLock` contract functions are accessible to itself in this manner.

Recommendation

1. Add the `TimeLock` contract's own address to the evmscript blacklist
2. Add the `TimeLock` lock token address to the evmscript blacklist
3. To fix spamming through batched callscripts, one option is to have users pass in a destination and calldata, and manually perform a call. Alternatively, `CallsScript` can be forked and altered to only execute a single external call to a single destination.

5.2 Passing duplicate tokens to `Redemptions` and `TokenRequest` may have unintended consequences Medium ✓ Addressed

Resolution

This was addressed in Redemptions commit 2b0034206a5b9cdf239da7a51900e89d9931554f by checking `redeemableTokenAdded[token] == false` for each subsequent token added during initialization. Note that ordering is not enforced.

Additionally, the issue in `TokenRequest` was addressed in commit eb4181961093439f142f2e74eb706b7f501eb5c0 by requiring that each subsequent

token added during initialization has a value strictly greater than the previous token added.

Description

Both `Redemptions` and `TokenRequest` are initialized with a list of acceptable tokens to use with each app. For `Redemptions`, the list of tokens corresponds to an organization's treasury assets. For `TokenRequest`, the list of tokens corresponds to tokens accepted for payment to join an organization. Neither contract makes a uniqueness check on input tokens during initialization, which can lead to unintended behavior.

Examples

- In `Redemptions`, each of an organization's assets are redeemed according to the sender's proportional ownership in the org. The redemption process iterates over the `redeemableTokens` list, paying out the sender their proportion of each token listed:

code/redemptions-app/contracts/Redemptions.sol:L112-L121

```
for (uint256 i = 0; i < redeemableTokens.length; i++) {
    vaultTokenBalance = vault.balance(redeemableTokens[i]);

    redemptionAmount = _burnableAmount.mul(vaultTokenBalance).div(burnableToke
    totalRedemptionAmount = totalRedemptionAmount.add(redemptionAmount);

    if (redemptionAmount > 0) {
        vault.transfer(redeemableTokens[i], msg.sender, redemptionAmount);
    }
}
```

If a token address is included more than once, the sender will be paid out more than once, potentially earning many times more than their proportional share of the token.

- In `TokenRequest`, this behavior does not allow for any significant deviation from expected behavior. It was included because the initialization process is similar to that of `Redemptions`.

Recommendation

During initialization in both apps, check that input token addresses are unique. One simple method is to require that token addresses are submitted in ascending order, and that each subsequent address added is greater than the one before.

5.3 The `Delay` app allows scripts to be paused even after execution time has elapsed Medium ✓ Addressed

Resolution
This was addressed in commit 46d8fa414cc3e68c68a5d9bc1174be5f32970611 by requiring that the current timestamp is before the delayed script's execution time.

Description

The `Delay` app is used to configure a delay between when an evm script is created and when it is executed. The entry point for this process is `Delay.delayExecution`, which stores the input script with a future execution date:

code/delay-app/contracts/Delay.sol:L153-L162

```
function _delayExecution(bytes _evmCallScript) internal returns (uint256) {
    uint256 delayedScriptIndex = delayedScriptsNewIndex;
    delayedScriptsNewIndex++;

    delayedScripts[delayedScriptIndex] = DelayedScript(getTimestamp64().add(ex

    emit DelayedScriptStored(delayedScriptIndex);

    return delayedScriptIndex;
}
```

An auxiliary capability of the `Delay` app is the ability to "pause" the delayed script, which sets the script's `pausedAt` value to the current block timestamp:

code/delay-app/contracts/Delay.sol:L80-L85

```
function pauseExecution(uint256 _delayedScriptId) external auth(PAUSE_EXECUTI
    require(!_isExecutionPaused(_delayedScriptId), ERROR_CAN_NOT_PAUSE);
    delayedScripts[_delayedScriptId].pausedAt = getTimestamp64();

    emit ExecutionPaused(_delayedScriptId);
}
```

A paused script cannot be executed until `resumeExecution` is called, which extends the script's `executionTime` by the amount of time paused. Essentially, the delay itself is paused:

code/delay-app/contracts/Delay.sol:L91-L100

```
function resumeExecution(uint256 _delayedScriptId) external auth(RESUME_EXECUT
    require(_isExecutionPaused(_delayedScriptId), ERROR_CAN_NOT_RESUME);
    DelayedScript storage delayedScript = delayedScripts[_delayedScriptId];

    uint64 timePaused = getTimestamp64().sub(delayedScript.pausedAt);
    delayedScript.executionTime = delayedScript.executionTime.add(timePaused);
    delayedScript.pausedAt = 0;

    emit ExecutionResumed(_delayedScriptId);
}
```

A delayed script whose execution time has passed and is not currently paused should be able to be executed via the `execute` function. However, the `pauseExecution` function still allows the aforementioned script to be paused, halting execution.

Recommendation

Add a check to `pauseExecution` to ensure that execution is not paused if the script's execution delay has already transpired.

5.4 Misleading intentional misconfiguration possible through misuse of `newToken` and `newBaseInstance` Medium ✓ Addressed

Resolution

This was addressed in commit b68d89ab0deb22161987e19d1ff0bb9d7303f0a9 by making `newToken` and `newBaseInstance` internal. A later commit addressed an invalid `DandelionVoting` import statement.

Description

The instantiation process for a Dandelion organization requires two separate external calls to `DandelionOrg`. There are two primary functions: `installDandelionApps`, and `newTokenAndBaseInstance`.

`installDandelionApps` relies on cached results from prior calls to `newTokenAndBaseInstance` and completes the initialization step for a Dandelion org.

`newTokenAndBaseInstance` is a wrapper around two publicly accessible functions: `newToken` and `newBaseInstance`. Called together, the functions: * Deploy a new `MiniMeToken` used to represent shares in an organization, and cache the address of the created token:

code/dandelion-org/contracts/DandelionOrg.sol:L128-L137

```
/**
* @dev Create a new MiniMe token and save it for the user
* @param _name String with the name for the token used by share holders in the
* @param _symbol String with the symbol for the token used by share holders i
*/
function newToken(string memory _name, string memory _symbol) public returns (
    MiniMeToken token = _createToken(_name, _symbol, TOKEN_DECIMALS);
    _saveToken(token);
    return token;
}
```

- Create a new dao instance using Aragon's `BaseTemplate` contract:

code/dandelion-org/contracts/DandelionOrg.sol:L139-L160

```
/**
* @dev Deploy a Dandelion Org DAO using a previously saved MiniMe token
* @param _id String with the name for org, will assign `[id].aragonid.eth`
```

```
* @param _holders Array of token holder addresses
* @param _stakes Array of token stakes for holders (token has 18 decimals, mu
* @param _useAgentAsVault Boolean to tell whether to use an Agent app as a mo
*/
function newBaseInstance(
    string memory _id,
    address[] memory _holders,
    uint256[] memory _stakes,
    uint64 _financePeriod,
    bool _useAgentAsVault
)
    public
{
    _validateId(_id);
    _ensureBaseSettings(_holders, _stakes);

    (Kernel dao, ACL acl) = _createDAO();
    _setupBaseApps(dao, acl, _holders, _stakes, _financePeriod, _useAgentAsVau
}
```

- Set up prepackaged Aragon apps, like `Vault`, `TokenManager`, and `Finance`:

code/dandelion-org/contracts/DandelionOrg.sol:L162-L182

```
function _setupBaseApps(
    Kernel _dao,
    ACL _acl,
    address[] memory _holders,
    uint256[] memory _stakes,
    uint64 _financePeriod,
    bool _useAgentAsVault
)
    internal
{
    MiniMeToken token = _getToken();
    Vault agentOrVault = _useAgentAsVault ? _installDefaultAgentApp(_dao) : _i
    TokenManager tokenManager = _installTokenManagerApp(_dao, token, TOKEN_TRA
    Finance finance = _installFinanceApp(_dao, agentOrVault, _financePeriod ==
```

```
    _mintTokens(_acl, tokenManager, _holders, _stakes);
    _saveBaseApps(_dao, finance, tokenManager, agentOrVault);
    _saveAgentAsVault(_dao, _useAgentAsVault);

}
```

Note that `newToken` and `newBaseInstance` can be called separately. The token created in `newToken` is cached in `_saveToken`, which overwrites any previously-cached value:

code/dandelion-org/contracts/DandelionOrg.sol:L413-L417

```
function _saveToken(MiniMeToken _token) internal {
    DeployedContracts storage senderDeployedContracts = deployedContracts[msg.

    senderDeployedContracts.token = address(_token);
}
```

Cached tokens are retrieved in `_getToken`:

code/dandelion-org/contracts/DandelionOrg.sol:L441-L447

```
function _getToken() internal returns (MiniMeToken) {
    DeployedContracts storage senderDeployedContracts = deployedContracts[msg.
    require(senderDeployedContracts.token != address(0), ERROR_MISSING_TOKEN_C

    MiniMeToken token = MiniMeToken(senderDeployedContracts.token);
    return token;
}
```

By exploiting the overwriteable caching mechanism, it is possible to intentionally misconfigure Dandelion orgs.

Examples

`installDandelionApps` uses `_getToken` to associate a token with the `DandelionVoting` app. The value returned from `_getToken` depends on the sender's previous call to `newToken`, which overwrites any previously-cached value. The steps for intentional misconfiguration are as follows:

1. Sender calls `newTokenAndBaseInstance`, creating token `m0` and DAO `A`.
    - The `TokenManager` app in `A` is automatically configured to be the controller of `m0`.
    - `m0` is cached using `_saveToken`.
    - DAO `A` apps are cached for future use using `_saveBaseApps` and `_saveAgentAsVault`.
2. Sender calls `newToken`, creating token `m1`, and overwriting the cache of `m0`.
    - Future calls to `_getToken` will retrieve `m1`.
    - The `DandelionOrg` contract is the controller of `m1`.
3. Sender calls `installDandelionApps`, which installs Dandelion apps in DAO `A`
    - The `DandelionVoting` app is configured to use the current cached token, `m1`, rather than the token associated with `A.TokenManager`, `m0`

Further calls to `newBaseInstance` and `installDandelionApps` create DAO `B`, populate it with Dandelion apps, and assign `B.TokenManager` as the controller of the earlier `DandelionVoting` app token, `m0`.

Many different misconfigurations are possible, and some may be underhandedly abusable.

Recommendation

Make `newToken` and `newBaseInstance` internal so they are only callable via `newTokenAndBaseInstance`.

5.5 `Delay.execute` can re-enter and re-execute the same script twice Minor ✓ Addressed

Resolution
This was addressed in commit f049e978f93765e27783a3ecac4830498bb779ba by deleting the delayed script before it is run. 1Hive elected to keep an empty script blacklist in order to allow delayed actions to be taken on the `Delay` app.

Description

`Delay.execute` does not follow the "checks-effects-interactions" pattern, and deletes a delayed script only after the script is run. Because the script being run executes arbitrary external calls, a script can be created that re-enters `Delay` and executes itself multiple times before being deleted:

code/delay-app/contracts/Delay.sol:L112-L123

```
/**
* @notice Execute the script with ID `_delayedScriptId`
* @param _delayedScriptId The ID of the script to execute
*/
function execute(uint256 _delayedScriptId) external {
    require(canExecute(_delayedScriptId), ERROR_CAN_NOT_EXECUTE);
    runScript(delayedScripts[_delayedScriptId].evmCallScript, new bytes(0), ne

    delete delayedScripts[_delayedScriptId];

    emit ExecutedScript(_delayedScriptId);
}
```

Recommendation

Add the `Delay` contract address to the `runScript` blacklist, or delete the delayed script from storage before it is run.

5.6 `Delay.cancelExecution` should revert on a non-existent script id

Minor ✓ Addressed

Resolution
This was addressed in commit d99c94f5138a9af1fd5f0cd6990c140b46a55925 by adding the `scriptExists(_delayedScriptId)` modifier to `cancelExecution`.

Description

`cancelExecution` makes no existence check on the passed-in script ID, clearing its storage slot and emitting an event:

code/delay-app/contracts/Delay.sol:L102-L110

```
/**
* @notice Cancel script execution with ID `_delayedScriptId`
* @param _delayedScriptId The ID of the script execution to cancel
*/
function cancelExecution(uint256 _delayedScriptId) external auth(CANCEL_EXECUT
    delete delayedScripts[_delayedScriptId];

    emit ExecutionCancelled(_delayedScriptId);
}
```

Recommendation

Add a check that the passed-in script exists.

5.7 ID validation check missing for `installDandelionApps` Minor ✓ Addressed

Resolution

This was addressed in commit 8d1ecb1bc892d6ea1d34c7234e35de031db2bebd by removing the `_id` parameter from `newTokenAndBaseInstance` and `newBaseInstance`, and adding a validation check to `installDandelionApps`.

Description

`DandelionOrg` allows users to kickstart an Aragon organization by using a dao template. There are two primary functions to instantiate an org: `newTokenAndBaseInstance`, and `installDandelionApps`. Both functions accept a parameter, `string _id`, meant to represent an ENS subdomain that will be assigned to the new org during the instantiation process. The two functions are called independently, but depend on each other.

In `newTokenAndBaseInstance`, a sanity check is performed on the `_id` parameter, which ensures the `_id` length is nonzero:

code/dandelion-org/contracts/DandelionOrg.sol:L155

```
_validateId(_id);
```

Note that the value of `_id` is otherwise unused in `newTokenAndBaseInstance`.

In `installDandelionApps`, this check is missing. The check is only important in this function, since it is in `installDandelionApps` that the ENS subdomain registration is actually performed.

Recommendation

Use `_validateId` in `installDandelionApps` rather than `newTokenAndBaseInstance`. Since the `_id` parameter is otherwise unused in `newTokenAndBaseInstance`, it can be removed.

Alternatively, the value of the submitted `_id` could be cached between calls and validated in `newTokenAndBaseInstance`, similarly to `newToken`.

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 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:

```
$ solium -V
Solium version 1.2.5
$ solium -d .

dandelion-org/contracts/DandelionOrg.sol
  86:1     warning    Line contains trailing whitespace          no-trailing-
  226:8    warning    Line exceeds the limit of 145 characters   max-len
```

```
dandelion-voting-app/contracts/DandelionVoting.sol
  272:8    warning    Line exceeds the limit of 145 characters    max-len

token-request-app/contracts/TokenRequest.sol
  62:4      warning    Line exceeds the limit of 145 characters
  104:1     warning    Line contains trailing whitespace

token-request-app/contracts/lib/UintArrayLib.sol
  6:3    error    Only use indent of 4 spaces.    indentation

✖ 1 error, 5 warnings found.
```

6.2 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:

Contracts Description Table

Contract	Type	Bases	
└	**Function Name**	**Visibility**	**Mutability**
AddressArrayLib	Library		
└	deleteItem	Internal 🔒	🛑
└	contains	Internal 🔒	🛑
ArrayUtils	Library		
└	deleteItem	Internal 🔒	🛑
DandelionOrg	Implementation	BaseTemplate	
└	<Constructor>	Public ❗	🛑
└	newTokenAndBaseInstance	External ❗	🛑

Contract	Type	Bases	
L	installDandelionApps	External ❗	🛑
L	newToken	Public ❗	🛑
L	newBaseInstance	Public ❗	🛑
L	_setupBaseApps	Internal 🔒	🛑
L	_installDandelionApps	Internal 🔒	🛑
L	_installDandelionVotingApp	Internal 🔒	🛑
L	_installDandelionVotingApp	Internal 🔒	🛑
L	_createDandelionVotingPermissions	Internal 🔒	🛑
L	_installRedemptionsApp	Internal 🔒	🛑
L	_createRedemptionsPermissions	Internal 🔒	🛑
L	_installTokenRequestApp	Internal 🔒	🛑
L	_createTokenRequestPermissions	Internal 🔒	🛑
L	_installTimeLockApp	Internal 🔒	🛑
L	_installTimeLockApp	Internal 🔒	🛑
L	_createTimeLockPermissions	Internal 🔒	🛑
L	_installTokenBalanceOracle	Internal 🔒	🛑
L	_createTokenBalanceOraclePermissions	Internal 🔒	🛑
L	_setupBasePermissions	Internal 🔒	🛑
L	_setupDandelionPermissions	Internal 🔒	🛑
L	_saveToken	Internal 🔒	🛑
L	_saveBaseApps	Internal 🔒	🛑
L	_saveAgentAsVault	Internal 🔒	🛑
L	_getDao	Internal 🔒	🛑
L	_getToken	Internal 🔒	🛑
L	_getBaseApps	Internal 🔒	🛑
L	_getAgentAsVault	Internal 🔒	🛑
L	_clearDeployedContracts	Internal 🔒	🛑

Contract	Type	Bases	
L	_ensureBaseAppsDeployed	Internal 🔒	🛑
L	_ensureBaseSettings	Private 🔐	
L	_ensureDandelionSettings	Private 🔐	🛑
L	_registerApp	Private 🔐	🛑
L	_setOracle	Private 🔐	🛑
L	_paramsTo256	Private 🔐	🛑
DandelionVoting	Implementation	IForwarder, IACLOracle, AragonApp	
L	initialize	External ❗	🛑
L	changeSupportRequiredPct	External ❗	🛑
L	changeMinAcceptQuorumPct	External ❗	🛑
L	changeBufferBlocks	External ❗	🛑
L	changeExecutionDelayBlocks	External ❗	🛑
L	newVote	External ❗	🛑
L	vote	External ❗	🛑
L	executeVote	External ❗	🛑
L	isForwarder	External ❗	
L	forward	Public ❗	🛑
L	canForward	Public ❗	
L	canPerform	External ❗	
L	canExecute	Public ❗	
L	canVote	Public ❗	
L	getVote	Public ❗	
L	getVoterState	Public ❗	
L	_newVote	Internal 🔒	🛑
L	_vote	Internal 🔒	🛑

Contract	Type	Bases	
└	_canExecute	Internal 🔒	
└	_votePassed	Internal 🔒	
└	_canVote	Internal 🔒	
└	_voterStake	Internal 🔒	
└	_isVoteOpen	Internal 🔒	
└	_isValuePct	Internal 🔒	
Delay	Implementation	AragonApp, IForwarder	
└	initialize	External ❗	🛑
└	setExecutionDelay	External ❗	🛑
└	delayExecution	External ❗	🛑
└	isForwarder	External ❗	
└	pauseExecution	External ❗	🛑
└	resumeExecution	External ❗	🛑
└	cancelExecution	External ❗	🛑
└	execute	External ❗	🛑
└	canExecute	Public ❗	
└	canForward	Public ❗	
└	forward	Public ❗	🛑
└	_isExecutionPaused	Internal 🔒	
└	_delayExecution	Internal 🔒	🛑
Redemptions	Implementation	AragonApp	
└	initialize	External ❗	🛑
└	addRedeemableToken	External ❗	🛑
└	removeRedeemableToken	External ❗	🛑
└	redeem	External ❗	🛑

Contract	Type	Bases	
└	getRedeemableTokens	External ❗	
└	getToken	External ❗	
└	getETHAddress	External ❗	
TimeLock	Implementation	AragonApp, IForwarder, IForwarderFee	
└	initialize	External ❗	🛑
└	changeLockDuration	External ❗	🛑
└	changeLockAmount	External ❗	🛑
└	changeSpamPenaltyFactor	External ❗	🛑
└	withdrawAllTokens	External ❗	🛑
└	withdrawTokens	External ❗	🛑
└	forwardFee	External ❗	
└	isForwarder	External ❗	
└	canForward	Public ❗	
└	forward	Public ❗	🛑
└	getWithdrawLocksCount	Public ❗	
└	getSpamPenalty	Public ❗	
└	_withdrawTokens	Internal 🔒	🛑
TokenBalanceOracle	Implementation	AragonApp, IACLOracle	
└	initialize	External ❗	🛑
└	setToken	External ❗	🛑
└	setMinBalance	External ❗	🛑
└	canPerform	External ❗	
TokenRequest	Implementation	AragonApp	
└	initialize	External ❗	🛑

Contract	Type	Bases	
└	setTokenManager	External ❗	🛑
└	setVault	External ❗	🛑
└	addToken	External ❗	🛑
└	removeToken	External ❗	🛑
└	createTokenRequest	External ❗	💵
└	refundTokenRequest	External ❗	🛑
└	finaliseTokenRequest	External ❗	🛑
└	getAcceptedDepositTokens	Public ❗	
└	getTokenRequest	Public ❗	
└	getToken	Public ❗	🛑
UintArrayLib	Library		
└	deleteItem	Internal 🔒	🛑

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 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.