DIMO

Dimo Smart Contracts

Security Assessment Report

Version: 2.0

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Dimo Network smart contracts. This included the Token, Vesting and Identity smart contracts. The review focused solely on the security aspects of the Solidity implementation of the contract, though general recommendations and informational comments are also provided.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the smart contract. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the set of Dimo Network smart contracts contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

Outputs of automated testing that were developed during this assessment are also included for reference (in the Appendix: Test Suite).

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the collection of Dimo Network smart contracts.

Overview

Dimo Network contracts in the current audit scope included the following projects:

- **Dimo Token:** an `ERC20Token` which will be used in the extended Dimo Network for accounting purposes. The token is intended for use within Dimo Vesting contracts and as use within Governance protocols.
- **Dimo Vesting:** a contract which handles the issuance of tokens held by Dimo beneficiaries. The delivery is managed through a standard vesting lockout period and redeemable iteratively after a set cliff period has expired.
- **Dimo Web3 Identity:** a modular NFT capable of delegating calls to permissioned modules with access to specific storage slots. This modular contract aims at storing, updating and deleting defined attributes based on the requirements of the Dimo NFT. More specifically, this contract can store and update user permissions, vehicle attributes and approved modules. The ability for future modules to be added allows the contract to adapt to changing requirements of the Dimo Network.

Security Assessment Summary

This review was conducted on the files hosted on the following repositories:

1. Dimo Token repository, assessed at commit ea6729ec14.
2. Dimo Vesting repository, assessed at commit 3510f41521.
3. Dimo Identity repository, assessed at commit 3b5dbb199f.

Initial retesting activities have been targetting the following commits:

1. Dimo Token repository, commit 9c8a350e8a.
2. Dimo Vesting repository, commit d13151c226.
3. Dimo Identity repository, commit 4ddf1402e8.

Note: the OpenZeppelin libraries and dependencies were excluded from the scope of this assessment.

The manual code review section of the report is focused on identifying any and all issues/vulnerabilities associated with the business logic implementation of the contracts. This includes their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Ethereum Virtual Machine (for example, verifying correct storage/memory layout). Additionally, the manual review process focused on all known Solidity anti-patterns and attack vectors. These include, but are not limited to, the following vectors: re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers. For a more thorough, but non-exhaustive list of examined vectors, see [1, 2].

To support this review, the testing team used the following automated testing tools:

- Mythril: `https://github.com/ConsenSys/mythril`
- Slither: `https://github.com/trailofbits/slither`

Output for these automated tools is available upon request.

Findings Summary

The testing team identified a total of 17 issues during this assessment. Categorised by their severity:

- High: 1 issue.
- Medium: 2 issues.
- Low: 1 issue.
- Informational: 13 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Dimo smart contracts. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contracts, including gas optimisations, are also described in this section and are labelled as “informational”.

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
DMO-01	Removal of `ERC20VotesUpgradeable` Might Allow Double Voting	High	Closed
DMO-02	Token `mint()` Produces Conflicts Across Bridges	Informational	Resolved
DMO-03	Voting Process Reverts	Medium	Closed
DMO-04	Function Signature Collision Creates Inaccessible Codeblocks	Medium	Resolved
DMO-05	Ownership Functionality Can Be Lost Through One Step Transfers	Low	Resolved
DMO-06	Ineffective Vesting Transfer Mechanism	Informational	Closed
DMO-07	Token Upgradeability May Lead to Denial-of-Service	Informational	Closed
DMO-08	No `initialize()` Function on Proxy Implementation	Informational	Resolved
DMO-09	Potentially Unsafe Use of `delegatecall()` Opcode	Informational	Closed
DMO-10	Ineffective Function Signature Registry Change Process	Informational	Resolved
DMO-11	`fallback` Function Could Potentially Cause Unintended Side-Effects	Informational	Resolved
DMO-12	Bypassable Limitations on NFT Minting	Informational	Resolved
DMO-13	Releasable Funds Can Be Revoked	Informational	Closed
DMO-14	Possible Reentrancy by Permissioned `owner` Account	Informational	Resolved
DMO-15	Lack of Validation on Vesting Start Time	Informational	Closed
DMO-16	Token Deployment Script Conflicts with `UUPS` Recommendations	Informational	Closed
DMO-17	Miscellaneous General Comments	Informational	Closed

DMO-01	Removal of `ERC20VotesUpgradeable` Might Allow Double Voting		
Asset	`DimoV2.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

Because it was not possible to run a governance propose/vote test, this vulnerability could not be further investigated, however it is a concern that OpenZeppelin's `ERC20VotesUpgradeable` contract was removed between this version of the token and its predecessor.

To quote OpenZeppelin's Documentation:

> *"This extension will keep track of historical balances so that voting power is retrieved from past snapshots rather than current balance, which is an important protection that prevents double voting."*

Recommendations

Be aware of this issue and ensure that double vote counting is not possible with the protocol's new governance implementation.

Resolution

After communication with the development team, it has been determined that the governance functionality of the DIMO token is out of scope of this audit. This feature is in active development and therefore related issues have been closed. As the governance functionality is out of scope, the testing team cannot attest to the security of this system. The testing team acknowledges that the development team is still moving forward based on recommendations provided in this issue.

DMO-02	Token `mint()` Produces Conflicts Across Bridges
Asset	`Dimo.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The Polygon child token contract (also called `Dimo.sol`) has both a `deposit()` function for the Polygon bridge, but also a `mint()` function. As the main token is on the Ethereum blockchain, the purpose of this child token is to serve as a bridged representation of the Ethereum Mainnet token on Polygon.

However, this function is incompatible due to the capacity to mint tokens on Polygon that do not correspond to tokens on the Ethereum Mainnet.

Consider this simplified scenario involving a situation where no tokens have been bridged to Polygon:

1. Alice uses the Polygon bridge to bridge 1,000 `DIMO` tokens to Polygon from the Ethereum Mainnet. She receives 1,000 child tokens and her original tokens are locked in the bridge.
2. `MINTER_ROLE` is able to call `mint()` on the child token contract. He receives 1,000 child tokens on Polygon.
3. `MINTER_ROLE` bridges his tokens back to Ethereum Mainnet, and receives the 1,000 DIMO tokens locked by Alice.
4. Alice attempts to bridge back to Mainnet. There are no tokens available in the Polygon bridge resulting in a net deficit.

Recommendations

Remove the function `mint()` from the child token.

Resolution

After discussion with the development team, this issue has been deemed inapplicable/irrelevant as far as security risks are concerned. Polygon documentation references the requirement for mintable assets in their documentation here. We have lowered the severity to informational as total token supply is split across ethereum and polygon chains in a manner that might be confusing. With a solely ETH minted asset, its total supply would always be canonical.

DMO-03	Voting Process Reverts		
Asset	`DimoV2.sol`, `DimoGovernor.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

Because of the removal of OpenZeppelin's `ERC20VotesUpgradeable` contract from the Dimo token in `DimoV2.sol`, it is not possible to propose a governance vote.

Calls to `DimoGovernance.propose()` revert because `propose()` calls `token.getPastVotes(account, blockNumber)`.

Recommendations

Modify `DimoV2` to reimplement OpenZeppelin's `ERC20VotesUpgradeable`.

Resolution

After communication with the development team, it has been determined that the governance functionality of the DIMO token is out of scope of this audit. This feature is in active development and therefore related issues have been closed. As the governance functionality is out of scope, the testing team cannot attest to the security of this system. The testing team acknowledges that the development team is still moving forward based on recommendations provided in this issue.

DMO-04	Function Signature Collision Creates Inaccessible Codeblocks		
Asset	`DIMORegistry.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

When new modules are added in `_addModule()`, their function signatures are checked against `s.implementations[selectors[i]]` on line [**129**] to ensure that the same function signature is not added twice. However, there is no check against the existing function signatures of `DIMORegistry`.

With the selection of modules as per the installation script, this situation does occur and several functions in the modules are installed but inaccessible because they are blocked by function signatures already present in `DIMORegistry`:

- `Getter.ownerOf()` is shadowed and blocked by `ERC721Base.ownerOf()`.
- `Getter.name()` is shadowed and blocked by `ERC721Metadata.name()`.
- `Getter.symbol()` is shadowed and blocked by `ERC721Metadata.symbol()`.
- `Getter.tokenURI()` is shadowed and blocked by `ERC721Metadata.tokenURI()`.

These shadowed functions, and any future shadowed functions, are inaccessible. This may have security implications if the team develop functionality on the assumption that one version of a function is being called, when in fact it is another.

The risk increases as more modules are added and the complexity of the system grows.

Recommendations

The development team might want to consider adding the `DIMORegistry` contract itself as a module, with all of its function signatures, simply to block potential collisions. If a tool such as `hardhat` is used to list all of a contract's selectors, then nothing will be missed.

Alternatively, a `require` check could be added within `_addModule` with `DIMORegistry`'s function signatures. This would be more reliable as it could not be removed.

Resolution

The testing team acknowledges that original issues regarding shadowing with internal modules has been resolved. The team have implemented the recommendation by adding `DIMORegistry` contract itself as a module. The fix can be found in the following commit: [4ddf1402](https://github.com/DIMO-Network/dimo-identity/commit/4ddf1402e81bb5d0f6ddaabfbd2f296721010c23).

DMO-05	Ownership Functionality Can Be Lost Through One Step Transfers		
Asset	`DIMOVesting.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

`DIMOVesting.sol` inherits from OpenZeppelin's `Ownable`. There are two potential issues in this contract which can allow ownership to be lost entirely:

1. As ownership transfer in `transferOwnership()` is unilateral, ownership can be accidentally transferred to an uncontrolled address.
2. An accidental call to `renounceOwnership()` transfers ownership to the zero address, effectively destroying all ownership functionality permanently.

Recommendations

Change ownership transfer to a propose/accept model and block or remove the `renounceOwnership()` function.

One convenient way to do this is to replace OpenZeppelin's `Ownable` with Chainlink's `ConfirmedOwnerWithProposal`

Resolution

The issue has been fixed in commit d13151c2. Chainlink's `ConfirmedOwnerWithProposal` was used to allow for a propose-accept pattern for owner transfers.

DMO-06	Ineffective Vesting Transfer Mechanism
Asset	`DIMOVesting.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

The `DIMOVesting` contract expects transfers to be made before creating a vesting schedule using `createVestingSchedule()`. As a result, there are checks to ensure that funds are sufficient.

However, if a user accidentally transfers funds to the vesting contract and the owner creates the vesting schedule without transferring, this will lock funds that did not belong to the owner.

Reversing this process is unnecessarily challenging. The owner will need to revoke the schedule, transfer the funds back to the user and then transfer new funds to the vesting schedule, whilst accounting for any potentially released funds, then creating the vesting schedule again.

This lengthy correction process is unnecessary and introduces potential for human error.

Recommendations

Ensure this behaviour is understood, if this behaviour isn't intentional consider forcing the user to transfer funds when they interact with `createVestingSchedule`

Resolution

The development team indicated that "the behavior is understood and intended" as shown in PR#1.

DMO-07	Token Upgradeability May Lead to Denial-of-Service
Asset	`DimoV2.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

The Dimo token uses the `EIP-1967` transparent proxy pattern to allow upgradeability, as has been used to upgrade from version 1. Whilst upgradeability adds flexibility, its downside is potential uncertainty: any upgradeable contract can be modified to any functionality.

For many contracts, the flexibility advantages are regarded as balancing the risks of upgradeability. In the case of tokens, however, as the functionality is simple, well established, and successfully implemented in many other instances, upgradeability is often viewed as an unacceptable risk by users.

Recommendations

Be aware of the potential and perceived potential for the token contract's proxy admin to do anything it wants to the token supply.

Unfortunately, the only way to convert an upgradeable proxy token to a standard token is to create a new contract and allow the tokens to be swapped for each other.

Nevertheless, transitioning to a static contract for a token is recommended for the benefits in stability and reliability it would provide.

Resolution

The development team indicated that "the behavior is understood" and that "token upgradability is likely to be temporary". These comments can be found in PR#2.

DMO-08	No `initialize()` Function on Proxy Implementation
Asset	`DimoV2.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

An `initialize()` function in a proxy implementation is used to set the initial values within the storage space of the proxy contract. `DimoV2` lacks such a function, and the reason for that seems to be that these values were already set within the first version of the Dimo token contract.

So long as the contract `DimoV2` is only used to upgrade from a successfully initialised first version Dimo token contract, this does not cause any problems. However, if any token were ever deployed with `DimoV2` as its first implementation contact, it would lack the values set in its `initialize()` function.

Recommendations

The team should acknowledge the issue and remain aware of it should they ever wish to deploy `DimoV2`. Alternatively, `DimoV2` could have an `initialize()` function added.

Resolution

The development team indicated that "the behavior is understood" as seen in PR#2. The reason to not have an `initialize()` function is that all variables were already initialized in the first version.

DMO-09	Potentially Unsafe Use of `delegatecall()` Opcode
Asset	`DIMORegistry.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

The functionality of `DIMORegistry` relies on `delegatecall`. However, this design decision may present significant risks in the future.

Indeed, any module that is added to `DIMORegistry` can potentially access and modify any part of the storage of `DIMORegistry`. That means it can grant any role, change any token allocation, or add other modules. The greatest danger is that a module is added which in some way ends up making a call to `selfdestruct`. This would cause `DIMORegistry` itself to delete its own code and result in a denial of service for any dependency contracts.

Recommendations

After discussion with the development team, and careful evaluation of the design, the testing team acknowledges this issue as an informational. Notwithstanding the informational status of this finding, the development team should place a high priority on ensuring that all modules added to `DIMORegistry` are reviewed for security. These reviews should not be performed in isolation, but rather ensure that the interactions with other existing modules are sound. Specific weight should be placed on ensuring storage collisions do not occur.

Resolution

The development team indicated that "the behavior is understood" as shown in PR#25. They intend to conduct a security review on all modules added to the system.

DMO-10	Ineffective Function Signature Registry Change Process
Asset	`DIMORegistry.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

Because of the check on line [**129**] of `DIMORegistry.sol`, it is not possible to add additional selectors from an implementation contract once any selectors at all have been added. This is because `selectorsHash[implementation]` will be set on module addition at line [**141**] and this value is never modified.

This may be present a risk if an implementation contract is removed, and then the protocol wishes to add it back to the registry again, or if the protocol wishes to add more function signatures from an already registered implementation contract.

In either case, the implementation contract in question would need to be redeployed.

Recommendations

Make sure this behaviour is understood and expected.

The development team may want to consider adding `s.selectorsHash[implementation] = 0x0` at line [**165**], at the end of `_removeModule()`, so that once a module has been removed, it could be added back if desired.

Resolution

The development team indicated that "the behavior is understood and expected" as shown in PR#25. However, they also implemented requested changes here. Modules can now be added back to the system after removal.

DMO-11	`fallback` Function Could Potentially Cause Unintended Side-Effects
Asset	`DIMORegistry.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The `_addModule()` function will add any function signature, even if it is not present on a module contract. If a module contract is added which contains a `fallback` function, that is what will be called when the nonexistant function signature is called on the registry contract.

There is no particular exploit scenario present in the current contracts, but this does suggest itself as either a way to hide attack code, or a possible unexpected scenario that might result from a deployment error (if a function signature is entered incorrectly, for example).

Recommendations

Be aware of this issue and avoid module contracts that implement a `fallback`.

Resolution

The development team indicated that "the behavior is understood" as shown in PR#25. They intend to conduct a security review on all modules added to the system. In doing so they will ensure no fallback functions are present in added modules.

DMO-12	Bypassable Limitations on NFT Minting
Asset	`Root.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

`Root.mintRoot()` contains a system based on the value `s.controllers[_owner].rootMinted` which only allows on root NFT to be minted to any given address. As clarified by the development team, the purpose of this check is to avoid one address holding multiple root NFTs.

However, as the NFTs are transferable, it would be easy for their owners to move them around and place many on a single address. In effect, therefore, the restriction imposed by `s.controllers[_owner].rootMinted` only really limits the minting powers of `DIMO`.

Recommendations

The team may wish to make the NFTs non-transferable or transferable only by the `DIMO` team. It would be possible to modify the NFT contracts to cause a transfer to revert if the new owner already owns a root NFT.

Alternatively, `s.controllers[_owner].rootMinted` and its associated checks and restrictions could simply be removed.

Resolution

This issue was fixed in the following commit #bc5efdf3. Token transfers are no longer possible.

DMO-13	Releasable Funds Can Be Revoked
Asset	`DIMOVesting.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

The `DIMOVesting` contract handles vesting of tokens in order to guarantee release after a certain period of time has expired. Tokens can be released after the cliff period, with all tokens becoming redeemable after the vesting duration has expired.

Vesting contracts are generally transparent contracts that handle how funds are released and when. However, Dimo reserves the ability to `revoke()` vesting positions at any time. In some circumstances revoking is necessary, however, if users already have a redeemable amount of tokens, the owner of the vesting contract can revoke the redeemable amount.

The contract will only validate the total previously released amount, and any unreleased amount will also be sent back to the vesting `owner`. This logic is described in the code snippet below:

```
uint256 unreleased = vestingSchedule.amountTotal -
vestingSchedule.released;
    if (unreleased > 0) {
        _token.safeTransfer(owner(), unreleased);
    }
```

Recommendations

Make sure this behaviour is understood and intended, and consider mentioning this explicitly in any relevant documentation.

If this behaviour is not intended, modification of `DIMOVesting` might include calculations of releasable token amounts and excluding these from the `unreleased` variable on line [**100**].

Resolution

The development team indicated that "the behavior is understood and intended" as shown in PR#1.

DMO-14	Possible Reentrancy by Permissioned `owner` Account
Asset	`DIMOVesting.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The token transfer on line [**103**] of `DIMOVesting.sol` is an external call. If `owner` can regain execution control during that call, they could potentially call `revoke()` for the same vest and keep draining out tokens.

The impact of this issue is mitigated heavily by DMO-13, which already gives `owner` the ability to withdraw all the tokens in the vesting contract by calling `revoke` on all open vests. This issue would become a lot more significant if that issue were resolved.

Recommendations

Consider moving the token transfer to the end of the `revoke()` function, after all the state variable updates.

Resolution

The issue has been fixed in commit 6a04cbf8. Check-effects pattern was implemented, preventing re-entrancy from impacting accountancy variables.

DMO-15	Lack of Validation on Vesting Start Time
Asset	`DIMOVesting.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

The `DIMOVesting` contract handles vesting of tokens in order to guarantee release after a certain period of time has elapsed. Vesting relies on a function `_computeReleasableAmount()` to calculate the amount to release. This function will only compute after a set start time has elapsed.

If `owner()` sets the start time to 0 (or any equivalent early epoch), vesting beneficiaries can instantly redeem the full amount.

Recommendations

Ensure this behaviour is understood and consider adding checks for the `_start` variable to be within valid bounds (for example restricting start time to at least the current epoch).

Resolution

The development team indicated that "the behavior is understood and intended" as shown in PR#1.

DMO-16	Token Deployment Script Conflicts with `UUPS` Recommendations
Asset	`scripts/deploy.js`
Status	**Closed:** See Resolution
Rating	Informational

Description

`DimoToken` inherits the `UUPS` upgradeable logic, implying the use of this type of proxy is desired. The development team has confirmed this intention. However, deployment scripts and mainnet deployed V2 token appear to ignore `UUPS` design decisions and leverage the use of `ERC1967` proxy.

The error can be seen in the the code block below:

```
15 const Dimo = await ethers.getContractFactory("Dimo");
   console.log("Deploying proxy and implementation.");
17 const dimo = await upgrades.deployProxy(Dimo);
   await dimo.deployed();
```

line [**17**] shows the use of OpenZeppelin Hardhat upgrades plugin. This plugin requires the explicit declaration of the proxy type as follows: `upgrades.deployProxy(Dimo, {kind: 'UUPS'})`. In the implementation, the kind of proxy is missing. OZ documentation states that use of function `deployProxy()` that is missing a value for proxy `type` will deploy as `Transparent` proxy.

In practice, the default behaviour of the proxy types were being altered at the time of deployment, which explains the behaviour of the contract deployments on mainnet. On mainnet, `TokenV1` was deployed as an `ERC1967` proxy. During the upgrade to V2, logic was included to ensure `UUPS` implementations were inherited and valid `_authorizeUpgrade` overrides prevented arbitrary users from upgrading the token. The mainnet `TokenV2` contains logic to ensure only `UPGRADER_ROLE` is able to upgrade, which effectively mitigates the issue in the deployment script logic.

It is worth mentioning that the current `UPGRADER_ROLE` is held by the GNOSIS SAFE Proxy contract with address `84ae2025b9620fd926d4e60673fcea2385c79d8a`.

Recommendations

Make sure this behaviour is understood and acknowledged.

Resolution

The development team indicated that "the behavior is understood" as seen in PR#2.

DMO-17	Miscellaneous General Comments
Asset	*.sol
Status	**Closed:** See Resolution
Rating	Informational

Description

This section details miscellaneous findings in the `Dimo` contracts.

1. **Minimise Function Access:**
   It is best practice to use the minimum visibility for a function, for gas saving purposes:
   - `DimoV2.pause()` should be declared external.
   - `DimoV2.unpause()` should be declared external.
   - `DimoV2.mint()` should be declared external.
2. **Checks Effects Interactions:**
   It is best practice to structure code in the order of making checks then updating variables and lastly interacting externally. This is to prevent reentrancy. Although there was no specific exploitable reentrancy occurrence detected (apart from DMO-14), it is nonetheless recommended to reorder code to follow the *"Checks-Effects-Interactions"* pattern.
   - In `Root,sol`, `_safeMint()` should be moved from line [**112**] to line [**116**], below the state variable updates.
   - In `Root,sol`, `_safeMint()` should be moved from line [**84**] to line [**88**], below the state variable updates.
3. **Zero Value Checks:**
   The zero value checks should be considered for the following parameters:
   - `DimoChildToken.deposit.depositData`
   - `DimoChildToken.withdraw.amount`
   - `DimoChildToken.mint.amount`
   - `Dimo.mint.amount`
   - `DimoV2.mint.amount`
   - `Root.setController._controller`
   - `Root.mintRootBatch._owner`
   - `Root.mintRoot._owner`
   - `Vehicle.mintVehicle._owner` (`Vehicle.mintVehicleSign._owner` is not necessary because of the signature)
   - `Root.mintRootBatch._owner`
4. **Comment Issues:**
   - `DIMORegistry` line [**108**] and line [**144**] comments describe the add functionality, not remove.

5. **Incorrect Error Message/Unreachable Code:**

   `DIMORegistry` line [**161**] states that a selector is unregistered. By the logic of this contract, this check should always pass, as execution can only reach this point if the check on line [**153**] established that `s.selectorsHash[implementation]` is equal to a hash that contains the selector in question. Nevertheless, it could be modified by a module. In that case, the comment could be incorrect: a selector that is registered to a different implementation would also fail this check.

6. **Registry Function Signature Collisions:**

   DIMORegistry does not allow modules to be added if `s.implementations[selectors[i]] != address(0)`, therefore registered function selectors cannot be duplicated across modules. This provides some security benefits, however, function signatures can collide. For example `withdraw(uint256)` and `OwnerTransferV7b711143(uint256)` unexpectedly produce a single function signature of `2e1a7d4d`. All modules should therefore be checked for any potential function signature collisions with other modules.

7. **Possible Specification Problem:**

   `Root.mintRootBatch()` can be successfully called whether 'name' attribute is whitelisted in advance or not. Note this in relation to the comment on line [**66**] which says *It is assumed the 'name' attribute is whitelisted in advance*.

8. **Vehicle without Root:**

   `Vehicle.addRootAttribute()` and `Vehicle.mintRoot()` are both callable if no vehicle `nodeType` is set in the `VehicleStorage`.

9. **Error Message Wording:**

   The error string on line [**128**] of `DIMOVesting.sol` refers to "vested" tokens, when it means "releasable" tokens. It might be clearer to rephrase the entire message as, "amount is too high".

10. **Uninitialised Roles:**

    Dimo Token V1 initially grants all roles to the admin address in its initializer. Dimo Token V2 does not have an initializer and so the role `BURNER_ROLE` is not initially granted to any address.

    This role can however be granted directly through the function `grantRole()`.

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Resolution

he issues were addressed in three separate pull requests across three repositories. Those include DIMO token PR#2, .

1. **Minimise Function Access:**

   1a) Fixed in commit #9b97f8ed

2. **Checks Effects Interactions:**

   2a) Fixed in commit #https://github.com/DIMO-Network/dimo-identity/commit/ade7190314582a87df0083dfb139bef2

3. **Zero Value Checks:**

   The zero value checks should be considered for the following parameters:

   - `DimoChildToken.deposit.depositData`

- `DimoChildToken.withdraw.amount`
- `DimoChildToken.mint.amount`
- `Dimo.mint.amount`
- `DimoV2.mint.amount`
- `Root.setController._controller` : Fixed in commit #bd4b17ab
- `Root.mintRootBatch._owner` : Not fixed, accepted as unnecessary due to checks elswhere
- `Root.mintRoot._owner` : Not fixed, accepted as unnecessary due to checks elswhere
- `Vehicle.mintVehicle._owner` : Not fixed, accepted as unnecessary due to checks elswhere
- `Root.mintRootBatch._owner` : Not fixed, accepted as unnecessary due to checks elswhere

4. **Comment Issues:**
   (a) Fixed in commit #46e2dc51
5. **Incorrect Error Message/Unreachable Code:**
   (a) Fixed in commit #7456c0c3
6. **Registry Function Signature Collisions:**
   (a) Behaviour is understood and accepted
7. **Possible Specification Problem:**
   (a) Behaviour is understood and accepted
8. **Vehicle without Root:**
   (a) Behaviour is understood and accepted. They will set the nodeType right after the module is added.
9. **Error Message Wording:**
   (a) Fixed in commit #b835a7e3
10. **Uninitialised Roles:**
    10a) Behaviour is accepted and intended

Appendix A Test Suite

A non-exhaustive list of tests were constructed to aid this security review and are given along with this document. The `brownie` framework was used to perform these tests and the output is given below.

```
dimo-token
dimo_governance.py::test_values PASSED                                                                  [ 16%]
dimo_governance.py::test_propose XFAIL (GovernorVotes tries to call token.getPastVotes(account, blockNumber)) [ 33%]
dimo_v2.py::test_mint PASSED                                                                            [ 50%]
dimo_v2.py::test_burn PASSED                                                                            [ 66%]
dimo_v2.py::test_pause PASSED                                                                           [ 83%]
dimo_v2.py::test_unpause PASSED                                                                         [100%]

dimo-token-local
child_token.py::test_mint PASSED                                                                        [  9%]
child_token.py::test_deposit PASSED                                                                     [ 18%]
child_token.py::test_withdraw PASSED                                                                    [ 27%]
child_token.py::test_pause PASSED                                                                       [ 36%]
child_token.py::test_unpause PASSED                                                                     [ 45%]
storage_v1.py::test_storage PASSED                                                                      [ 54%]
v1.py::test_snapshot PASSED                                                                             [ 63%]
v1.py::test_mint PASSED                                                                                 [ 72%]
v1.py::test_burn PASSED                                                                                 [ 81%]
v1.py::test_pause PASSED                                                                                [ 90%]
v1.py::test_unpause PASSED                                                                              [100%]

dimo-web3-identity
AccessControl.py::test_grant_role PASSED                                                                [  2%]
AccessControl.py::test_revoke_role PASSED                                                               [  5%]
AccessControl.py::test_renounce_role PASSED                                                             [  8%]
AccessControl.py::test_get_role_admin PASSED                                                            [ 10%]
AttributeSet.py::test_nothing PASSED                                                                    [ 13%]
DIMORegistry.py::test_deployment PASSED                                                                 [ 16%]
DIMORegistry.py::test_add_module PASSED                                                                 [ 18%]
DIMORegistry.py::test_add_module_again PASSED                                                           [ 21%]
DIMORegistry.py::test_remove_module PASSED                                                              [ 24%]
DIMORegistry.py::test_remove_module_failures PASSED                                                     [ 27%]
DIMORegistry.py::test_update_module PASSED                                                              [ 29%]
DIMORegistry.py::test_add_module_overlapped PASSED                                                      [ 32%]
DIMORegistry.py::test_remove_module_then_add_again PASSED                                               [ 35%]
FallbackModulePOC.py::test_fallback_module PASSED                                                       [ 37%]
Getter.py::test_name XFAIL (module function signature is blocked by contract function)                  [ 40%]
Getter.py::test_symbol XFAIL (module function signature is blocked by contract function)                [ 43%]
Getter.py::test_base_uri PASSED                                                                         [ 45%]
Getter.py::test_token_uri XFAIL (module function signature is blocked by contract function)             [ 48%]
Getter.py::test_owner_of XFAIL (module function signature is blocked by contract function)              [ 51%]
Getter.py::test_get_node_type PASSED                                                                    [ 54%]
Getter.py::test_get_info PASSED                                                                         [ 56%]
Metadata.py::test_set_base_uri PASSED                                                                   [ 59%]
Metadata.py::test_set_token_uri PASSED                                                                  [ 62%]
Root.py::test_deployment PASSED                                                                         [ 64%]
Root.py::test_set_root_node_type PASSED                                                                 [ 67%]
Root.py::test_add_root_attribute PASSED                                                                 [ 70%]
Root.py::test_set_controller PASSED                                                                     [ 72%]
Root.py::test_mint_root PASSED                                                                          [ 75%]
Root.py::test_set_root_info PASSED                                                                      [ 78%]
Root.py::test_is_controller PASSED                                                                      [ 81%]
Root.py::test_is_root_minted PASSED                                                                     [ 83%]
Root.py::test_mint_root_batch PASSED                                                                    [ 86%]
Vehicle.py::test_set_vehicle_node_type PASSED                                                           [ 89%]
Vehicle.py::test_add_vehicle_attribute PASSED                                                           [ 91%]
Vehicle.py::test_mint_vehicle PASSED                                                                    [ 94%]
Vehicle.py::test_set_vehicle_info PASSED                                                                [ 97%]
Vehicle.py::test_mint_vehicle_sign PASSED                                                               [100%]

dimo_vesting
```

```
dimo_vesting.py::test_initial_conditions PASSED                                    [ 25%]
dimo_vesting.py::test_vesting_release PASSED                                       [ 50%]
dimo_vesting.py::test_revoke_releasable XFAIL (Shouldn't revoke releasable amounts) [ 75%]
dimo_vesting.py::test_vesting_durations PASSED                                     [100%]
```

Appendix B Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

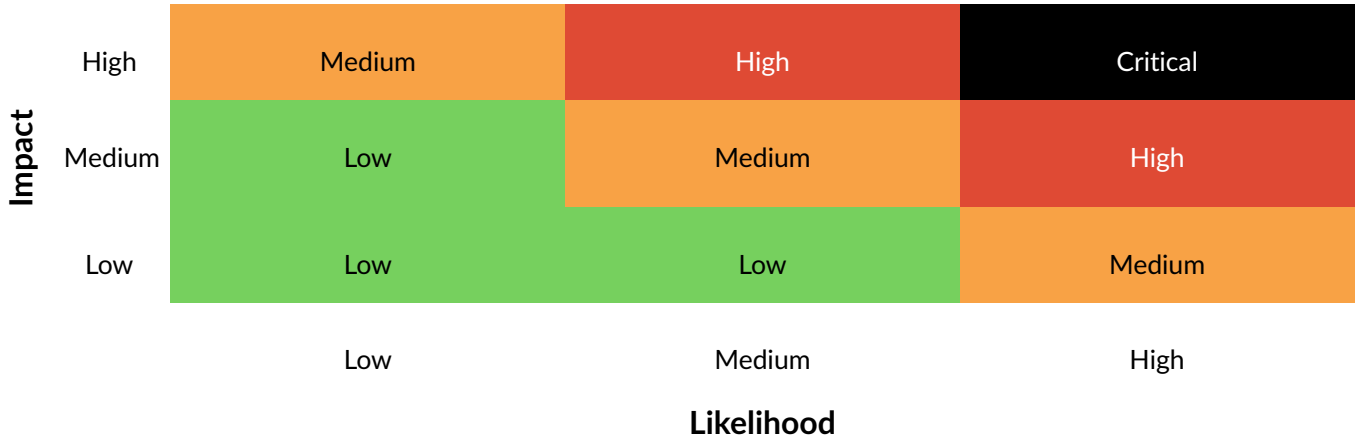

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References

[1] Sigma Prime. Solidity Security. Blog, 2018, Available: https://blog.sigmaprime.io/solidity-security.html. [Accessed 2018].

[2] NCC Group. DASP - Top 10. Website, 2018, Available: http://www.dasp.co/. [Accessed 2018].