DaMeta1

Security Assessment

CertiK Assessed on Jan 7th, 2026

CertiK Assessed on Jan 7th, 2026

DaMeta1

The security assessment was prepared by CertiK.

Executive Summary

TYPES	ECOSYSTEM	METHODS
Vesting	Ethereum (ETH)	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE
Solidity	Preliminary comments published on 12/12/2025 Final report published on 01/07/2026

Vulnerability Summary

7	2	1	0	4	0
Total Findings	Resolved	Multi-Sig	Partially Resolved	Acknowledged	Declined

2	Centralization	1 Multi-Sig, 1 Acknowledged	Centralization findings highlight privileged roles & functions and their capabilities, or instances where the project takes custody of users' assets.
0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
0	Major		Major risks may include logical errors that, under specific circumstances, could result in fund losses or loss of project control.
0	Medium		Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
1	Minor	1 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
4	Informational	2 Resolved, 2 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | DAMETA1

CODEBASE | DAMETA1

Repository

DaMeta1

Commit

- 10283008229e8669350cdf177f7f29a55846e504
- 78217a2f239aa42df2a8bbf19433b9cdd1800ac6

AUDIT SCOPE | DAMETA1

jawadijaz-commits/DaMeta1-DMU
masterVestingContract.sol
DMUTokenContract.sol
DMUTokenContract.sol
masterVestingContract.sol

APPROACH & METHODS | DAMETA1

This audit was conducted for DaMeta1 to evaluate the security and correctness of the smart contracts associated with the DaMeta1 project. The assessment included a comprehensive review of the in-scope smart contracts. The audit was performed using a combination of Static Analysis and Manual Review.

The review process emphasized the following areas:

- Architecture review and threat modeling to understand systemic risks and identify design-level flaws.
- Identification of vulnerabilities through both common and edge-case attack vectors.
- Manual verification of contract logic to ensure alignment with intended design and business requirements.
- Dynamic testing to validate runtime behavior and assess execution risks.
- Assessment of code quality and maintainability, including adherence to current best practices and industry standards.

The audit resulted in findings categorized across multiple severity levels, from informational to critical. To enhance the project's security and long-term robustness, we recommend addressing the identified issues and considering the following general improvements:

- Improve code readability and maintainability by adopting a clean architectural pattern and modular design.
- Strengthen testing coverage, including unit and integration tests for key functionalities and edge cases.
- Maintain meaningful inline comments and documentations.
- Implement clear and transparent documentation for privileged roles and sensitive protocol operations.
- Regularly review and simulate contract behavior against newly emerging attack vectors.

REVIEW NOTES | DAMETA1

Overview

DaMeta1 is a Solidity-based vesting contract deployed on the Ethereum mainnet that manages the time-locked distribution of `dmuToken`. It implements a vesting schedule that begins with an initial cliff period, followed by a linear release of tokens to designated beneficiaries.

External Dependencies

In **DaMeta1**, the module inherits or uses a few of the depending injection contracts or addresses to fulfill the need of its business logic. The scope of the audit treats third party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets.

Addresses

The following addresses interact at some point with specified contracts, making them an external dependency. All of following values are initialized either at deploy time or by specific functions in smart contracts.

MasterVesting:

- `dmuToken` .

We assume these contracts or addresses are valid and non-vulnerable actors and implementing proper logic to collaborate with the current project.

Also, the following library/contract are considered as the third-party dependencies:

- `@openzeppelin/contracts/`
- `./libraries/BokkyPooBahsDateTimeLibrary.sol`
- `./libraries/Errors.sol`

Privileged Functions

In the **DaMeta1** project, the privileged roles are adopted to ensure the dynamic runtime updates of the project, which are specified in the **Centralization** findings.

The advantage of those privileged roles in the codebase is that the client reserves the ability to adjust the protocol according to the runtime required to best serve the community. It is also worth noting the potential drawbacks of these functions, which should be clearly stated through the client's action/plan. Additionally, if the private keys of the privileged accounts are compromised, it could lead to devastating consequences for the project.

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of the `Timelock` contract.

FINDINGS | DAMETA1

7	0	2	0	0	1	4
Total Findings	Critical	Centralization	Major	Medium	Minor	Informational

This report has been prepared for DaMeta1 to identify potential vulnerabilities and security issues within the reviewed codebase. During the course of the audit, a total of 7 issues were identified. Leveraging a combination of Static Analysis & Manual Review the following findings were uncovered:

ID	Title	Category	Severity	Status
DAM-02	**Initial Token Distribution**	**Centralization**	**Centralization**	**2/3 Multi-Sig**
DAM-03	**Centralization Risks In MasterVestingContract.Sol**	**Centralization**	**Centralization**	**Acknowledged**
DAM-04	Potential Insufficient Funds	Logical Issue	Minor	Acknowledged
DAM-01	Discussion On The Vesting Loigc	Logical Issue	Informational	Acknowledged
DAM-05	Array Missing `pop` Function	Volatile Code	Informational	Acknowledged
DAM-06	Remove Redundant Condition `tgeTimestamp == 0`	Volatile Code	Informational	Resolved
DAM-07	Inconsistency Between Code And Comment	Volatile Code	Informational	Resolved

DAM-02 | Initial Token Distribution

Category	Severity	Location	Status
Centralization	**Centralization**	**DMUTokenContract.sol (DaMeta1): 65~66**	**2/3 Multi-Sig**

Description

All **DMU** tokens are transferred to a single target address, `recipient`. This introduces a centralization risk, as the `recipient` address can distribute tokens without community consensus. If this address is ever compromised, an attacker could steal and liquidate the tokens on the market, causing significant harm to the project.

Recommendation

It is recommended that the team be transparent regarding the initial token distribution process. The token distribution plan should be published in a public location that the community can access. The team should make efforts to restrict access to the private keys of the deployer account or EOAs. A multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. Additionally, the team can lock up a portion of tokens, release them with a vesting schedule for long-term success, and deanonymize the project team with a third-party KYC provider to create greater accountability.

Alleviation

[DaMeta1, 12/19/2025]: The team have published a dedicated public transparency and governance page outlining the DMU deployment and distribution flow, including:

- Main token contract deployment and verification approach
- Ownership and custody under a 2-of-3 multisignature wallet
- Public disclosure of multisig wallet and signer addresses
- Pre-sale, vesting, and claim flows (with TBA placeholders where applicable)
- Reference to the whitepaper and a forthcoming transparency dashboard

The page also outlines key governance controls, including:

- Execution of privileged actions via multisignature custody with a minimum 24-hour timelock, and
- The renouncement of vesting contract ownership once vesting categories and recipient allocations are finalized and verified DMU Token Transparency & On-Chain Governance: https://dmu.dameta1.com/transparency

[CertiK, 01/07/2026]:

The following deployment information was provided by the project team:

- The `DaMeta1UtilityToken` contract is deployed at 0x2827d26a9eddc0a4b9bde8d152da59a730a402b4.

- The token distribution plan is publicly available at: https://www.dmu.dameta1.com/#tokenomics

Based on these deployment information, the audit team has verified the following:

- The deployed code of the `DaMeta1UtilityToken` contract matches the source code in `DMUTokenContract.sol` from GitHub commit ba2f24223aab18b188f2ceb57258d02d97cf97c3.
- The project utilizes a multi-signature wallet at the following address: 0x9e3a4b50926e62b8a61ba0ac84ffcdb9c38061cf. The multi-signature wallet is configured with three signers:
  - Signer 1: `0xEb0E6dC42Ba54154bBe399C3306818f34c94CADb`
  - Signer 2: `0xb73ce1e5591b5d461F315e62f1247BD889619E7b`
  - Signer 3: `0x7f809Fd074e0d903a39d8902e6bB25faF08F8F5b`

  The signature threshold is set to **2 out of 3** signers.

Upon contract deployment, the `DaMeta1UtilityToken` contract mints a total supply of **5,000,000,000 DMU tokens** (18 decimals). The initial supply is minted directly to the project's multi-signature wallet at `0x9e3a4b50926e62b8a61ba0ac84ffcdb9c38061cf` as part of the contract creation transaction 0x05a542b2c7019e7464f6838e7a0460a5a6633ab85d1eeba6eb42bd486ff16da1.

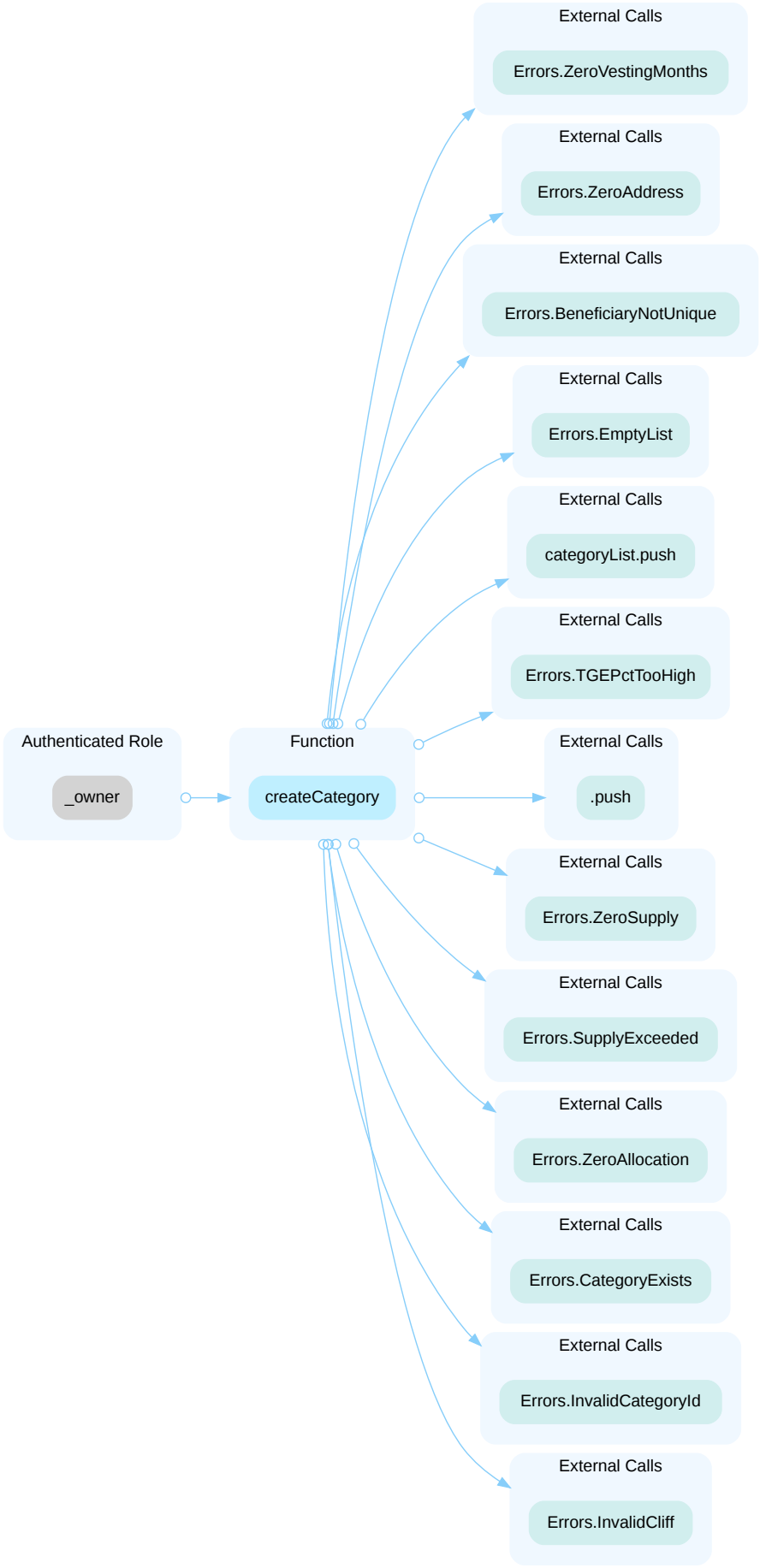
As of **Jan-07-2026 06:10:47 PM +UTC**, the full token supply remains held by the multi-signature wallet, and no distributions or transfers to external addresses have been observed.

DAM-03 | Centralization Risks In MasterVestingContract.Sol

Category	Severity	Location	Status
Centralization	**Centralization**	**masterVestingContract.sol: 166**	**Acknowledged**

Description

In the `MasterVesting` contract, the `_owner` role has full authority over the functions shown in the diagram below. If the `_owner` account is compromised, an attacker could abuse these privileges to create malicious vesting plans for attacker-controlled addresses, allowing them to illegitimately claim the vesting tokens.


External Calls
Errors.ZeroVestingMonths
External Calls
Errors.ZeroAddress
External Calls
Errors.BeneficiaryNotUnique
External Calls
Errors.EmptyList
External Calls
categoryList.push
External Calls
Errors.TGEPctTooHigh
Authenticated Role
_owner
Function
createCategory
External Calls
.push
External Calls
Errors.ZeroSupply
External Calls
Errors.SupplyExceeded
External Calls
Errors.ZeroAllocation
External Calls
Errors.CategoryExists
External Calls
Errors.InvalidCategoryId
External Calls
Errors.InvalidCliff

The `MasterVesting` contract inherits from `Ownable2Step` and `Ownable` from the OpenZeppelin library. As a result, the contract owner (`_owner`) also has authority over the following functions:

- `transferOwnership()`
- `renounceOwnership()`

In the latest version (78217a2f239aa42df2a8bbf19433b9cdd1800ac6), a new function has been added to the `MasterVesting` contract that is restricted to the `_owner` :

- `addBeneficiariesToCategory()`

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[DaMeta1, 12/19/2025]:

The team is executing the short-term mitigation strategy outlined above to reduce the identified risk exposure:

Batch Processing (Implemented Code Update)

To support onboarding of a large number of seed investors across multiple launchpads and to prevent out-of-gas risks, the vesting logic has been updated to support batch-based addition of beneficiaries:

- A vesting category may receive multiple add-beneficiary calls
- The contract enforces:
  - No duplicate beneficiaries per category
  - No modification of previously added beneficiaries
  - No exceeding of the category's total allocation
  - Append-only behavior (only new beneficiaries may be added)

Governance & Ownership Controls

- Vesting categories and their respective allocation percentages are finalized as defined in the whitepaper. Recipient addresses will be finalized prior to TGE, and a newly implemented contract-level check prevents adding any new beneficiaries after TGE, ensuring immutability of vesting allocations.
- All privileged operations prior to finalization are governed by a 2-of-3 multisignature wallet, backed by independent hardware wallets
- A timelock (minimum 24 hours) is applied to multisig executions to enhance transparency and reduce operational risk
- Once vesting categories and recipient allocations are finalized and verified prior to TGE, ownership of the vesting contract will be renounced, ensuring immutability and eliminating future privileged control.
- Multisig transactions are monitored with alerts for additional transparency and risk reduction

[CertiK, 12/26/2025]: The team has proposed a mitigation plan that satisfies the recommended short-term approach. However, the finding's status will only be updated following on-chain deployment and full verification of all deployment details.

DAM-04 | Potential Insufficient Funds

Category	Severity	Location	Status
Logical Issue	Minor	masterVestingContract.sol (DaMeta1): 214~215, 274~275	Acknowledged

Description

The `createCategory()` function allows the `_owner` to create a new vesting category and define its immutable allocations. The `categorySupplyWei` parameter specifies the total number of vesting tokens within that category that can be distributed to beneficiaries.

However, the contract lacks any `dmuToken` transfer logic during category creation. If the contract is not sufficiently pre-funded with vesting tokens, the `claimAll()` function will revert with `InsufficientContractBalance` whenever a beneficiary's claimable amount exceeds the contract's current token balance. This would block all beneficiaries from claiming, requiring the operations team to ensure that the contract remains adequately funded at all times.

Recommendation

It is recommended to fund the contract with sufficient vesting tokens whenever a new vesting category is created.

Alleviation

[DaMeta1, 12/19/2025]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement.

- Vesting contract funding is executed via multisig wallets secured by hardware devices
- The team ensures sufficient tokens are deposited into the vesting contract before TGE and before enabling claims for any category, preventing under-funding scenarios

DAM-01 | Discussion On The Vesting Loigc

Category	Severity	Location	Status
Logical Issue	Informational	masterVestingContract.sol (DaMeta1): 328~329	Acknowledged

Description

The `tokensToClaim()` function returns the claimable amount for a given category and beneficiary.

```
    uint256 vestedMonths = getVestedMonths(tgeTimestamp, block.timestamp);
    ...
    if (cliff != 0) {

        if (vestedMonths <= cliff) {
            return tgeAmount - tokensClaimed;
        }
```

The `vestedMonths` value represents the number of months between the current timestamp and the constant `tgeTimestamp`.

Under the current logic, linear vesting begins only when `vestedMonths` > `cliff`. Consider the following example:

- `tgeTimestamp` = 2025-01-01 00:00
- `cliff` = 1 month

However, according to the current vested token calculation logic, the linear vesting always begins when `vestedMonths > cliff`.

`block.timestamp`	vestedMonths	Condition	Claimable
2025-02-01 00:00	1	1 <= cliff	Only `tgeAmount`
2025-02-28 23:59	1	1 <= cliff	Only `tgeAmount`
2025-03-01 00:00	2	2 > cliff	First month of linear vesting begins

This shows that linear vesting does not start precisely at the 1-month cliff boundary (> 2025-02-01 00:00). Instead, it effectively begins one full month later at the moment when `vestedMonths` increments from 1 to 2. Consequently, beneficiaries receive only the `tgeAmount` throughout the entire first month and up until the exact moment the second month begins.

If the intended behavior is for linear vesting to start immediately at the 1-month mark, this condition introduces an unintended one-month delay.

Recommendation

We recommend discussing this behavior with the team to confirm whether this design is intentional and aligned with the vesting specification.

Alleviation

[DaMeta1, 12/19/2025]: After further internal review, we confirm that the existing vesting logic is intentional and reflects our business design. Linear vesting is designed to begin only after the cliff period has fully elapsed, rather than exactly at the cliff boundary. This ensures that categories with a zero-month cliff do not begin linear vesting at TGE, preventing unintended immediate vesting at launch and maintaining predictable initial token circulation. This behavior is consistent with our token distribution strategy and will be clearly communicated to the community. Accordingly, no code changes were required for DAM-01.

DAM-05 | Array Missing `pop` Function

Category	Severity	Location	Status
Volatile Code	Informational	masterVestingContract.sol (DaMeta1): 78, 84, 87	Acknowledged

Description

Arrays without the pop operation in Solidity can lead to inefficient memory management and increase the likelihood of out-of-gas errors.

Recommendation

Consider adding functionality to remove elements from the array to prevent it from becoming too large over the lifetime of the contract.

Alleviation

[DaMeta1, 12/19/2025]: We acknowledge the informational finding noted in DAM-05. While no code changes are required at this stage, the recommendation has been documented and incorporated into our internal guidelines and future reviews.

DAM-06 | Remove Redundant Condition `tgeTimestamp == 0`

Category	Severity	Location	Status
Volatile Code	Informational	masterVestingContract.sol (DaMeta1): 301	Resolved

Description

The constructor ensures that `tgeTimestamp` is always set to a non-zero value and is not in the past. Therefore, the runtime check `tgeTimestamp == 0` in `tokensToClaim()` cannot be triggered and serves no practical purpose.

Recommendation

It is recommended to remove this condition to improve code efficiency.

Alleviation

[DaMeta1, 12/19/2025]: The team heeded the advice and resolved the issue by removing the redundant condition `tgeTimestamp == 0` in commit 78217a2f239aa42df2a8bbf19433b9cdd1800ac6.

DAM-07 | Inconsistency Between Code And Comment

Category	Severity	Location	Status
Volatile Code	Informational	masterVestingContract.sol (DaMeta1): 142	Resolved

Description

The constructor comment states that the TGE timestamp “MUST be in the future”, but the implementation only enforces `_tgeTimestamp < block.timestamp` as invalid. As a result, a TGE timestamp equal to the deployment block timestamp is allowed.

Recommendation

It is recommended to update either the comment or the implementation to ensure they are consistent.

Alleviation

[DaMeta1, 12/19/2025]: The team heeded the advice and resolved the issue by by modifying the inconsistent code in commit 78217a2f239aa42df2a8bbf19433b9cdd1800ac6.

The condition `_tgeTimestamp < block.timestamp` has been updated to `_tgeTimestamp <= block.timestamp` in the modified version (line 146), aligning the implemented logic with the associated comment and the intended behavior.

FORMAL VERIFICATION | DAMETA1

Formal guarantees about the behavior of smart contracts can be obtained by reasoning about properties relating to the entire contract (e.g. contract invariants) or to specific functions of the contract. Once such properties are proven to be valid, they guarantee that the contract behaves as specified by the property. As part of this audit, we applied formal verification to prove that important functions in the smart contracts adhere to their expected behaviors.

Considered Functions And Scope

In the following, we provide a description of the properties that have been used in this audit. They are grouped according to the type of contract they apply to.

Verification of ERC-20 Compliance

We verified properties of the public interface of those token contracts that implement the ERC-20 interface. This covers

- Functions `transfer` and `transferFrom` that are widely used for token transfers,
- functions `approve` and `allowance` that enable the owner of an account to delegate a certain subset of her tokens to another account (i.e. to grant an allowance), and
- the functions `balanceOf` and `totalSupply`, which are verified to correctly reflect the internal state of the contract.

The properties that were considered within the scope of this audit are as follows (note that overflow properties were excluded from the verification):

Property Name	Title
erc20-transfer-exceed-balance	`transfer` Fails if Requested Amount Exceeds Available Balance
erc20-transfer-correct-amount	`transfer` Transfers the Correct Amount in Transfers
erc20-transferfrom-correct-allowance	`transferFrom` Updated the Allowance Correctly
erc20-transfer-false	If `transfer` Returns `false`, the Contract State Is Not Changed
erc20-approve-never-return-false	`approve` Never Returns `false`
erc20-balanceof-succeed-always	`balanceOf` Always Succeeds
erc20-approve-false	If `approve` Returns `false`, the Contract's State Is Unchanged
erc20-balanceof-correct-value	`balanceOf` Returns the Correct Value
erc20-approve-correct-amount	`approve` Updates the Approval Mapping Correctly
erc20-transfer-never-return-false	`transfer` Never Returns `false`
erc20-allowance-succeed-always	`allowance` Always Succeeds

Property Name	Title
erc20-allowance-correct-value	`allowance` Returns Correct Value
erc20-transferfrom-false	If `transferFrom` Returns `false`, the Contract's State Is Unchanged
erc20-totalsupply-succeed-always	`totalSupply` Always Succeeds
erc20-totalsupply-correct-value	`totalSupply` Returns the Value of the Corresponding State Variable
erc20-approve-succeed-normal	`approve` Succeeds for Valid Inputs
erc20-approve-revert-zero	`approve` Prevents Approvals For the Zero Address
erc20-transferfrom-fail-exceed-balance	`transferFrom` Fails if the Requested Amount Exceeds the Available Balance
erc20-transfer-revert-zero	`transfer` Prevents Transfers to the Zero Address
erc20-transferfrom-revert-zero-argument	`transferFrom` Fails for Transfers with Zero Address Arguments
erc20-balanceof-change-state	`balanceOf` Does Not Change the Contract's State
erc20-allowance-change-state	`allowance` Does Not Change the Contract's State
erc20-totalsupply-change-state	`totalSupply` Does Not Change the Contract's State
erc20-transferfrom-correct-amount	`transferFrom` Transfers the Correct Amount in Transfers
erc20-transferfrom-fail-exceed-allowance	`transferFrom` Fails if the Requested Amount Exceeds the Available Allowance
erc20-transferfrom-never-return-false	`transferFrom` Never Returns `false`

Verification Results

For the following contracts, formal verification established that each of the properties that were in scope of this audit (see scope) are valid:

Detailed Results For Contract DaMeta1UtilityToken (DMUTokenContract.sol) In Commit 78217a2f239aa42df2a8bbf19433b9cdd1800ac6

Verification of ERC-20 Compliance

Detailed Results for Function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-exceed-balance	True	
erc20-transfer-correct-amount	True	
erc20-transfer-false	True	
erc20-transfer-never-return-false	True	
erc20-transfer-revert-zero	True	

Detailed Results for Function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-correct-allowance	True	
erc20-transferfrom-false	True	
erc20-transferfrom-fail-exceed-balance	True	
erc20-transferfrom-revert-zero-argument	True	
erc20-transferfrom-correct-amount	True	
erc20-transferfrom-fail-exceed-allowance	True	
erc20-transferfrom-never-return-false	True	

Detailed Results for Function `approve`

Property Name	Final Result	Remarks
erc20-approve-never-return-false	True	
erc20-approve-false	True	
erc20-approve-correct-amount	True	
erc20-approve-succeed-normal	True	
erc20-approve-revert-zero	True	

Detailed Results for Function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	True	
erc20-balanceof-correct-value	True	
erc20-balanceof-change-state	True	

Detailed Results for Function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	True	
erc20-allowance-correct-value	True	
erc20-allowance-change-state	True	

Detailed Results for Function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	True	
erc20-totalsupply-correct-value	True	
erc20-totalsupply-change-state	True	

Detailed Results For Contract DaMeta1UtilityToken (DMUTokenContract.sol) In Commit 10283008229e8669350cdf177f7f29a55846e504

Verification of ERC-20 Compliance

Detailed Results for Function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-exceed-balance	True	
erc20-transfer-correct-amount	True	
erc20-transfer-false	True	
erc20-transfer-never-return-false	True	
erc20-transfer-revert-zero	True	

Detailed Results for Function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-correct-allowance	True	
erc20-transferfrom-fail-exceed-balance	True	
erc20-transferfrom-fail-exceed-allowance	True	
erc20-transferfrom-correct-amount	True	
erc20-transferfrom-revert-zero-argument	True	
erc20-transferfrom-false	True	
erc20-transferfrom-never-return-false	True	

Detailed Results for Function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-correct-value	True	
erc20-totalsupply-succeed-always	True	
erc20-totalsupply-change-state	True	

Detailed Results for Function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	True	
erc20-allowance-change-state	True	
erc20-allowance-correct-value	True	

Detailed Results for Function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-correct-value	True	
erc20-balanceof-succeed-always	True	
erc20-balanceof-change-state	True	

Detailed Results for Function `approve`

Property Name	Final Result	Remarks
erc20-approve-correct-amount	True	
erc20-approve-never-return-false	True	
erc20-approve-revert-zero	True	
erc20-approve-succeed-normal	True	
erc20-approve-false	True	

APPENDIX | DAMETA1

Finding Categories

Categories	Description
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.

Details on Formal Verification

Some Solidity smart contracts from this project have been formally verified. Each such contract was compiled into a mathematical model that reflects all its possible behaviors with respect to the property. The model takes into account the semantics of the Solidity instructions found in the contract. All verification results that we report are based on that model.

The following assumptions and simplifications apply to our model:

- Certain low-level calls and inline assembly are not supported and may lead to a contract not being formally verified.
- We model the semantics of the Solidity source code and not the semantics of the EVM bytecode in a compiled contract.

Formalism for property specifications

All properties are expressed in a behavioral interface specification language that CertiK has developed for Solidity, which allows us to specify the behavior of each function in terms of the contract state and its parameters and return values, as well as contract properties that are maintained by every observable state transition. Observable state transitions occur when the contract's external interface is invoked and the invocation does not revert, and when the contract's Ether balance is changed by the EVM due to another contract's "self-destruct" invocation. The specification language has the usual Boolean connectives, as well as the operator `\old` (used to denote the state of a variable before a state transition), and several types of specification clause:

Apart from the Boolean connectives and the modal operators "always" (written `[]`) and "eventually" (written `<>`), we use the following predicates to reason about the validity of atomic propositions. They are evaluated on the contract's state whenever a discrete time step occurs:

- `requires [cond]` - the condition `cond`, which refers to a function's parameters, return values, and contract state variables, must hold when a function is invoked in order for it to exhibit a specified behavior.
- `ensures [cond]` - the condition `cond`, which refers to a function's parameters, return values, and both `\old` and current contract state variables, is guaranteed to hold when a function returns if the corresponding requires condition held

when it was invoked.

- `invariant [cond]` - the condition `cond`, which refers only to contract state variables, is guaranteed to hold at every observable contract state.
- `constraint [cond]` - the condition `cond`, which refers to both `\old` and current contract state variables, is guaranteed to hold at every observable contract state except for the initial state after construction (because there is no previous state); constraints are used to restrict how contract state can change over time.

Description of the Analyzed ERC-20 Properties

Properties related to function `transfer`

erc20-transfer-correct-amount

All non-reverting invocations of `transfer(recipient, amount)` that return `true` must subtract the value in `amount` from the balance of `msg.sender` and add the same value to the balance of the `recipient` address.

Specification:

```
requires recipient != msg.sender;
requires balanceOf(recipient) + amount <= type(uint256).max;
ensures \result ==> balanceOf(recipient) == \old(balanceOf(recipient) + amount)
&& balanceOf(msg.sender) == \old(balanceOf(msg.sender) - amount);
  also
requires recipient == msg.sender;
ensures \result ==> balanceOf(msg.sender) == \old(balanceOf(msg.sender));
```

erc20-transfer-exceed-balance

Any transfer of an amount of tokens that exceeds the balance of `msg.sender` must fail.

Specification:

```
requires amount > balanceOf(msg.sender);
ensures !\result;
```

erc20-transfer-false

If the `transfer` function in contract `DaMeta1UtilityToken` fails by returning `false`, it must undo all state changes it incurred before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

erc20-transfer-never-return-false

The transfer function must never return `false` to signal a failure.

Specification:

```
ensures \result;
```

erc20-transfer-revert-zero

Any call of the form `transfer(recipient, amount)` must fail if the recipient address is the zero address.

Specification:

```
ensures \old(recipient) == address(0) ==> !\result;
```

Properties related to function `transferFrom`

erc20-transferfrom-correct-allowance

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` must decrease the allowance for address `msg.sender` over address `from` by the value in `amount`.

Specification:

```
ensures \result ==> allowance(\old(sender), msg.sender) == \old(allowance(sender,
msg.sender)) - \old(amount)
                 || (allowance(\old(sender), msg.sender) == \old(allowance(sender,
msg.sender)) && \old(allowance(sender, msg.sender)) == type(uint256).max);
```

erc20-transferfrom-correct-amount

All invocations of `transferFrom(from, dest, amount)` that succeed and that return `true` subtract the value in `amount` from the balance of address `from` and add the same value to the balance of address `dest`.

Specification:

```
requires recipient != sender;
requires balanceOf(recipient) + amount <= type(uint256).max;
ensures \result ==> balanceOf(\old(recipient)) == \old(balanceOf(recipient) +
amount)
                 && balanceOf(\old(sender)) == \old(balanceOf(sender) - amount);
  also
requires recipient == sender;
ensures \result ==> balanceOf(\old(recipient)) == \old(balanceOf(recipient));
```

erc20-transferfrom-fail-exceed-allowance

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the allowance of address `msg.sender` must fail.

Specification:

```
requires msg.sender != sender;
requires amount > allowance(sender, msg.sender);
ensures !\result;
```

erc20-transferfrom-fail-exceed-balance

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the balance of address `from` must fail.

Specification:

```
requires amount > balanceOf(sender);
ensures !\result;
```

erc20-transferfrom-false

If `transferFrom` returns `false` to signal a failure, it must undo all incurred state changes before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

erc20-transferfrom-never-return-false

The `transferFrom` function must never return `false` .

Specification:

```
ensures \result;
```

erc20-transferfrom-revert-zero-argument

All calls of the form `transferFrom(from, dest, amount)` must fail for transfers from or to the zero address.

Specification:

```
ensures \old(sender) == address(0) ==> !\result;
also
ensures \old(recipient) == address(0) ==> !\result;
```

Properties related to function `approve`

erc20-approve-correct-amount

All non-reverting calls of the form `approve(spender, amount)` that return `true` must correctly update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount`.

Specification:

```
requires spender != address(0);
ensures \result ==> allowance(msg.sender, \old(spender)) == \old(amount);
```

erc20-approve-false

If function `approve` returns `false` to signal a failure, it must undo all state changes that it incurred before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

erc20-approve-never-return-false

The function `approve` must never returns `false`.

Specification:

```
ensures \result;
```

erc20-approve-revert-zero

All calls of the form `approve(spender, amount)` must fail if the address in `spender` is the zero address.

Specification:

```
ensures \old(spender) == address(0) ==> !\result;
```

erc20-approve-succeed-normal

All calls of the form `approve(spender, amount)` must succeed, if

- the address in `spender` is not the zero address and
- the execution does not run out of gas.

Specification:

```
requires spender != address(0);
ensures \result;
reverts_only_when false;
```

Properties related to function `balanceOf`

erc20-balanceof-change-state

Function `balanceOf` must not change any of the contract's state variables.

Specification:

```
assignable \nothing;
```

erc20-balanceof-correct-value

Invocations of `balanceOf(owner)` must return the value that is held in the contract's balance mapping for address `owner`.

Specification:

```
ensures \result == balanceOf(\old(account));
```

erc20-balanceof-succeed-always

Function `balanceOf` must always succeed if it does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `allowance`

erc20-allowance-change-state

Function `allowance` must not change any of the contract's state variables.

Specification:

```
assignable \nothing;
```

erc20-allowance-correct-value

Invocations of `allowance(owner, spender)` must return the allowance that address `spender` has over tokens held by address `owner`.

Specification:

```
ensures \result == allowance(\old(owner), \old(spender));
```

erc20-allowance-succeed-always

Function `allowance` must always succeed, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `totalSupply`

erc20-totalsupply-change-state

The `totalSupply` function in contract DaMeta1UtilityToken must not change any state variables.

Specification:

```
assignable \nothing;
```

erc20-totalsupply-correct-value

The `totalSupply` function must return the value that is held in the corresponding state variable of contract DaMeta1UtilityToken.

Specification:

```
ensures \result == totalSupply();
```

erc20-totalsupply-succeed-always

The function `totalSupply` must always succeeds, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Elevating Your Web3 Journey

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is the largest blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.