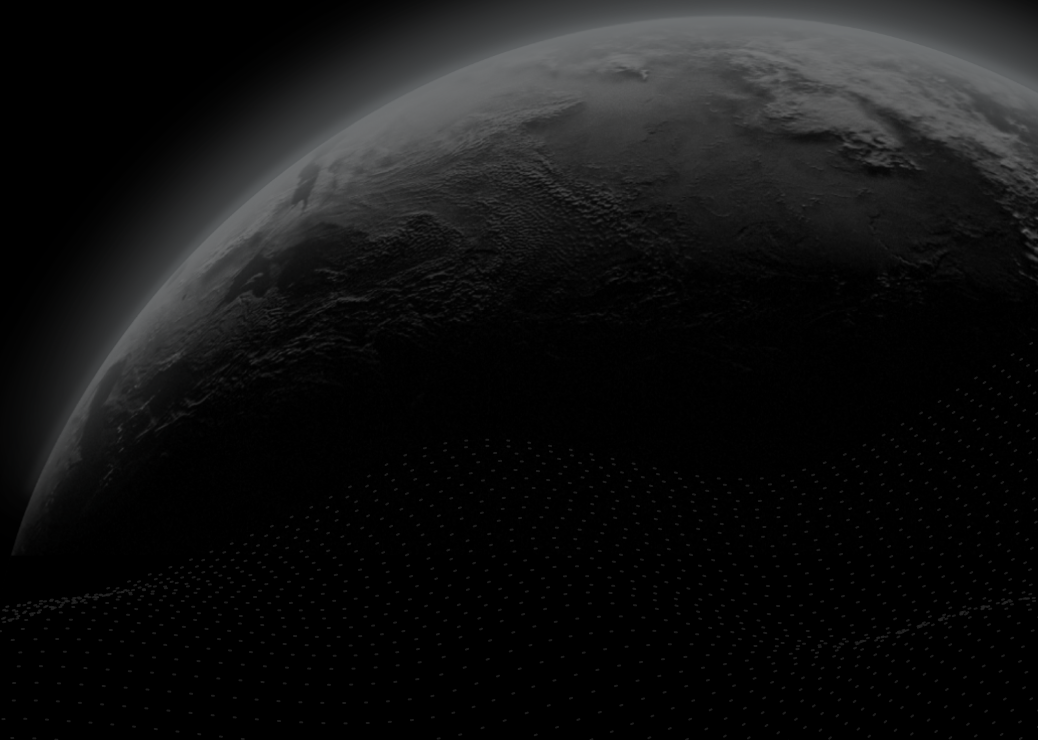
CERTIK

Security Assessment

Momentum - Addendum

CertiK Assessed on Jul 12th, 2023



CertiK Assessed on Jul 12th, 2023

Momentum - Addendum

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
ERC-20, NFT	Binance Smart Chain (BSC) | Ethereum (ETH)	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 07/12/2023	N/A

CODEBASE

https://github.com/momentum-xyz/contracts/

View All in Codebase Page

COMMITS

- e10687248f1af24907587d5ece38abfb0754a94f
- 8b5d331e81e75a28eeb953999c3a0b61182f154f

View All in Codebase Page

Highlighted Centralization Risks

- Contract upgradeability
- Initial owner token share is 100%
- Transfers can be paused
- Privileged role can mint tokens

Vulnerability Summary

16	11	0	0	5	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

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.
4	Major	1 Resolved, 3 Acknowledged	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
4	Medium	3 Resolved, 1 Acknowledged	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
5	Minor	4 Resolved, 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.

3	Informational	3 Resolved	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 | MOMENTUM - ADDENDUM

CODEBASE | MOMENTUM - ADDENDUM

Repository

https://github.com/momentum-xyz/contracts/

Commit

- e10687248f1af24907587d5ece38abfb0754a94f
- 8b5d331e81e75a28eeb953999c3a0b61182f154f

AUDIT SCOPE | MOMENTUM - ADDENDUM

5 files audited ● 5 files with Acknowledged findings

ID	Repo	File	SHA256 Checksum
● OFT	momentum-xyz/contracts	contracts/nft/OdysseyNFT.sol	e426ca8702c229afed5b2a11f899ac06b1f8e35e299d65c30ff3423bfeb72b34
● STN	momentum-xyz/contracts	contracts/staking/Staking.sol	0c49504971055309671e048df0b4f1e9f315152f6fbc2adc09821837b8e9eb37
● DDT	momentum-xyz/contracts	contracts/token/DADToken.sol	de01a9f3090b10c933fc28bae8620acbb41effe80fc2c2c002c23e84315db02c
● MTH	momentum-xyz/contracts	contracts/token/MomToken.sol	5f59db9c590d46ab3cae4211617d13339ea170c8ed653e8f0433bb88ef2609a2
● VET	momentum-xyz/contracts	contracts/vesting/Vesting.sol	de254e2d431576c270547a52380eb3cdc54c85a494088a666f81869891d06873

APPROACH & METHODS | MOMENTUM - ADDENDUM

This report has been prepared for Momentum to discover issues and vulnerabilities in the source code of the Momentum - Addendum project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

REVIEW NOTES | MOMENTUM - ADDENDUM

Overview

Momentum is an NFT ecosystem. The review contains the following modules:

- **OdysseyNFT**: The `OdysseyNFT` contract is a non-fungible token(NFT) based on `ERC721`. Normal users can stake/unstake tokens (`MOM` or `DAD` discussed below) on Odyssey NFT.
- **MOMToken**: The `MOMToken` contract is an `ERC20` token short for `MOM`. It can be used as staked token by users to stake on Odyssey NFT to gain rewards. In addition, `MOM` is also the reward token minted to users or the owner of Odyssey NFT. It is worth noting that the accounts with `MINTER_ROLE` role can mint any number of `MOM` to any account.
- **DADToken**: The `DADToken` contract is an `ERC20` token short for `DAD`. It can be used as staked token by users to stake on Odyssey NFT to gain rewards. In addition, `DAD` is only allowed to be transferred by accounts with `TRANSFER_ROLE` role granted. It is worth noting that the accounts with `MINTER_ROLE` role can mint any number of `DAD` to any account.
- **Staking**: The `Staking` is the core contract of `Momentum` including functions like staking, unstaking, updating rewards, claiming rewards, etc. Users can stake `MOM` or `DAD` tokens on Odyssey NFTs. The address granted with `MANAGER_ROLE` role will update the rewards in this contract. The calculation of rewards is not reflected in this contract. Once the rewards updated, users can claim their rewards. The owner of Odyssey NFT can also gain rewards if the Odyssey NFT has been staked into. All the staked tokens kept in the `Staking` contract can be withdrawn by users while unstaking.
- **Vesting**: The `Vesting` contract provides a vesting mechanism for `DAD` holders to gradually burn `DAD` tokens and get `MOM` tokens gradually over 2 years.

External Dependencies

In **Momentum**, the project relies on a few external contracts or addresses to fulfill the needs of its business logic.

MOMToken:

- the addresses `dad` and `vesting`

Staking:

- rewards calculated by third party: the logic to calculate the rewards is a black-box
- the addresses `mom_token`, `dad_token`, and `odyssey_nfts`
- treasury: when the `MANAGER_ROLE` updates rewards, the `Staking` contract will also mint some `MOM` tokens to this address

It is assumed that the rewards calculator and the address of `treasury` are trusted and implemented properly within the current project.

Privileged Functions

In the **Momentum°** project, multiple privileged roles are adopted to ensure the dynamic runtime updates of the project, which were specified in the following findings `CON-02 | Centralization Related Risks`.

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 | MOMENTUM - ADDENDUM

16	0	4	4	5	3
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Momentum - Addendum. Through this audit, we have uncovered 16 issues ranging from different severity levels. Utilizing the techniques of Static Analysis & Manual Review to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
CON-02	**Centralization Related Risks**	**Centralization**	**Major**	**Acknowledged**
MTH-01	Initial Token Distribution	Centralization / Privilege	Major	Acknowledged
STK-05	**Centralized Control Of Contract Upgrade**	**Centralization**	**Major**	**Acknowledged**
STN-01	Wrong Rewards Claimed From Oddysey	Coding Issue	Major	Resolved
OFT-02	Function `transferFrom` Of Contract `OdysseyNFT` Does NOT Fail When Paused	Logical Issue	Medium	Resolved
STN-02	Possible For Stakes To Be Locked In The Contract	Design Issue	Medium	Acknowledged
STN-03	Inaccurate Calculation On Effective Timestamp	Coding Issue	Medium	Resolved
VET-04	Possible For Last Claim Date To Go Back In Time	Logical Issue	Medium	Resolved
OFT-01	Function `burn()` Of `OdysseyNFT` Contract Cannot Be Paused	Coding Issue	Minor	Resolved
STN-04	Incompatibility With Deflationary Tokens	Logical Issue	Minor	Acknowledged

ID	Title	Category	Severity	Status
STN-06	Usage Of `effective_timestamp` In Function `calculate_effective_timestamp()`	Design Issue	Minor	Resolved
STN-07	Possible Ether Locked In `Staking` Contract	Language-Specific	Minor	Resolved
VET-02	Check-Effects-Interactions Pattern Violation	Logical Issue	Minor	Resolved
OFT-03	Unused Event	Coding Issue	Informational	Resolved
VET-01	Potentially Arithmetic Underflow	Coding Issue	Informational	Resolved
VET-05	Missing Emit Events	Coding Style	Informational	Resolved

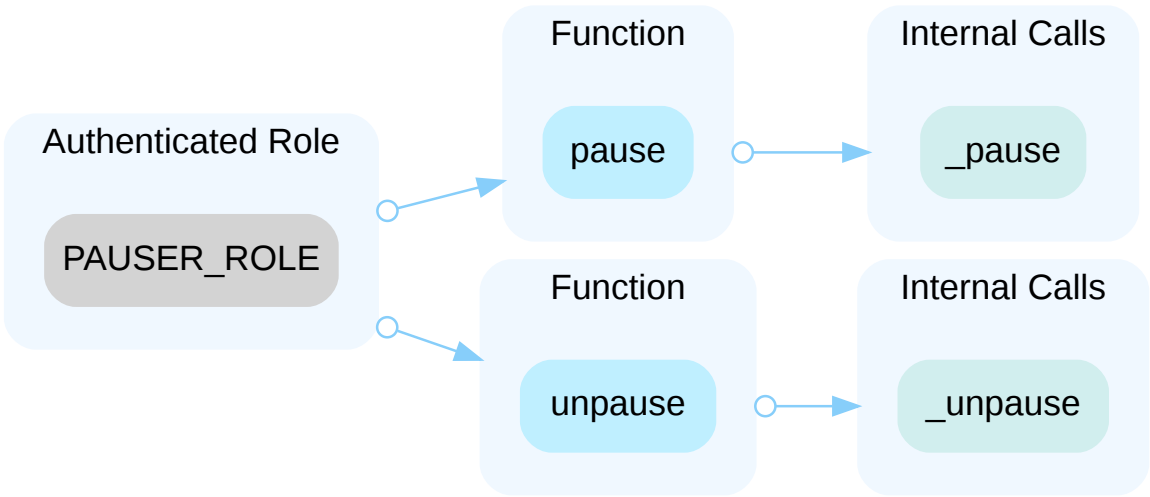
CON-02 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization	Major	contracts/nft/OdysseyNFT.sol: 72, 79, 95, 118; contracts/staking/Staking.sol: 260, 276, 317, 327; contracts/token/DADToken.sol: 42, 47, 57, 74, 83, 89, 98; contracts/token/MomToken.sol: 62, 70, 80, 92, 114, 124; contracts/vesting/Vesting.sol: 93, 104	Acknowledged

Description

In the contract `MOMToken`, the role `PAUSER_ROLE` has authority over the functions shown in the list and diagram below.:

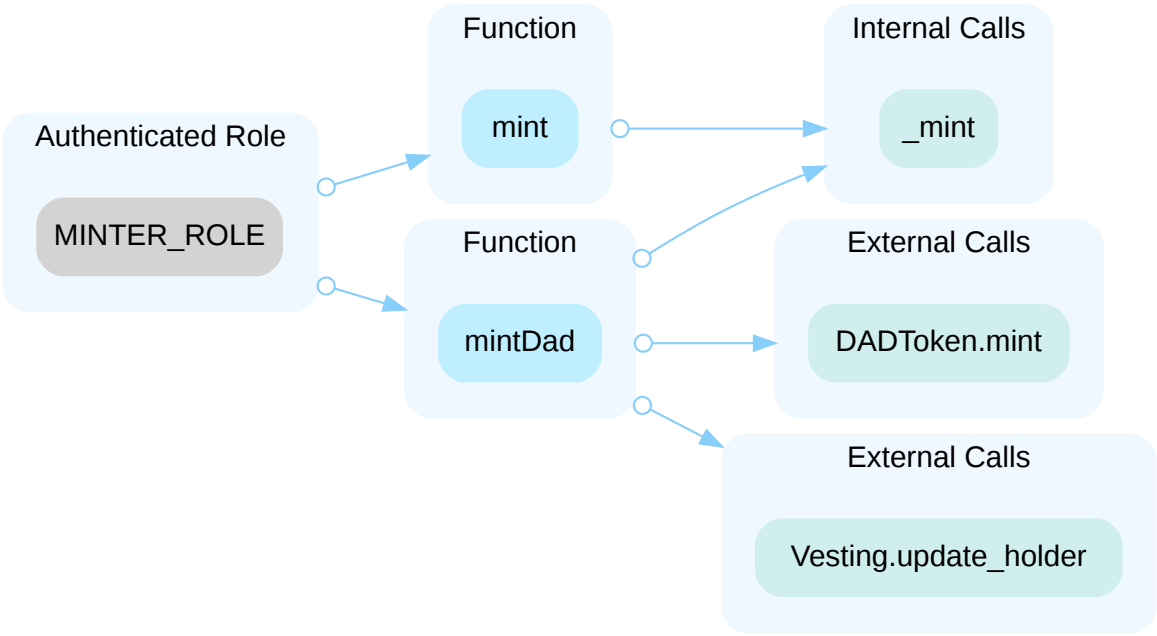
- function pause(): Accounts assigned the `PAUSER_ROLE` can pause the contract, which will prevent the transfers or burning of `MOM` tokens.
- function unpause(): Accounts assigned the `PAUSER_ROLE` can unpause the contract, which will allow the transfers or burning of `MOM` tokens.



Any compromise to the `PAUSER_ROLE` accounts may allow the hacker to take advantage of this authority and pause or unpause `MOM` tokens to be transferred.

In the contract `MOMToken`, the role `MINTER_ROLE` has authority over the functions shown in the list and diagram below:

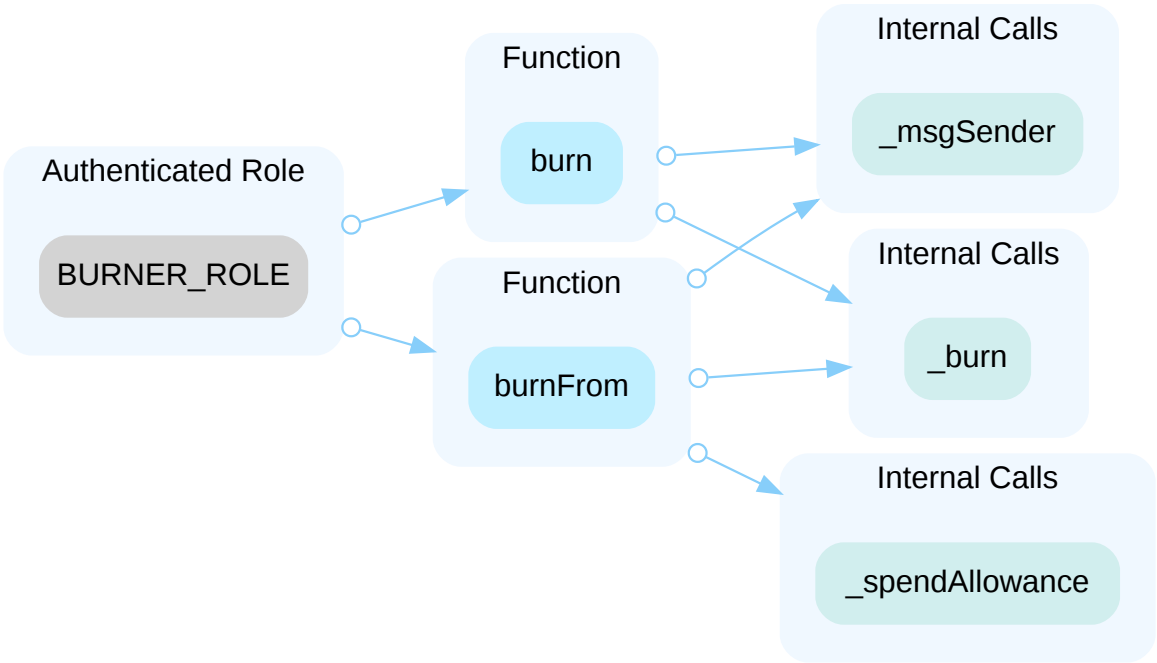
- function mint(): Accounts assigned the `MINTER_ROLE` can mint arbitrary amounts of `MOM` tokens to any account.
- function mintDad(): Accounts assigned the `MINTER_ROLE` can mint arbitrary amounts of `DAD` tokens to any account and mint the same amount to `Vesting` contract.



Any compromise to the `MINTER_ROLE` accounts may allow the hacker to take advantage of this authority and mint `DAD` tokens to any accounts and mint `MOM` to `Vesting` contract.

In the contract `MOMToken`, the role `BURNER_ROLE` has authority over the functions shown in the list and diagram below:

- function burn(): Accounts assigned the `BURNER_ROLE` can burn their own `MOM` tokens.
- function burnFrom(): Accounts that have been assigned the `BURNER_ROLE` can use their allowance to burn `MOM` tokens belonging to other accounts.



Any compromise to the `BURNER_ROLE` accounts may allow the hacker to take advantage of this authority and burn `MOM`

tokens.

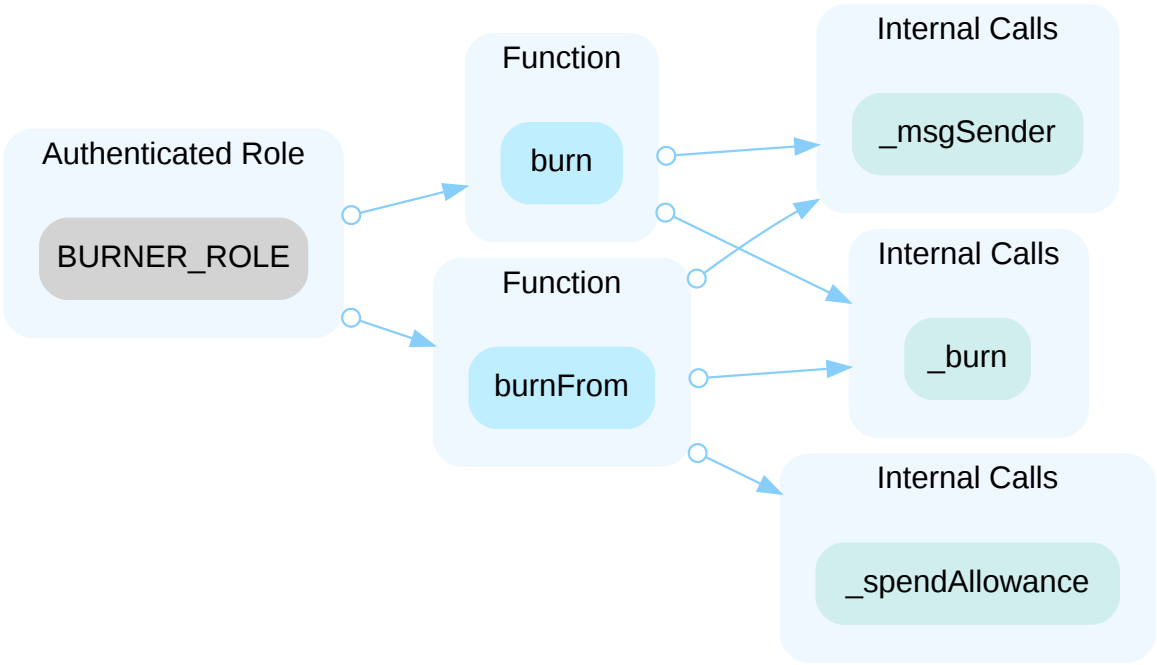
In the contract `MOMToken`, there is also `DEFAULT_ADMIN_ROLE` role inherited from `AccessControl` contract. Accounts granted with `DEFAULT_ADMIN_ROLE` can grant or revoke other roles as mentioned above. By default, the deployer of `MOMToken` contract has been granted all these roles.

- DEFAULT_ADMIN_ROLE
- PAUSER_ROLE
- MINTER_ROLE
- BURNER_ROLE

Any compromise to the `DEFAULT_ADMIN_ROLE` accounts may allow the hacker to take advantage of this authority and operate the same actions as mentioned above.

In the contract `DADToken`, the role `BURNER_ROLE` has authority over the functions shown in the list and diagram below:

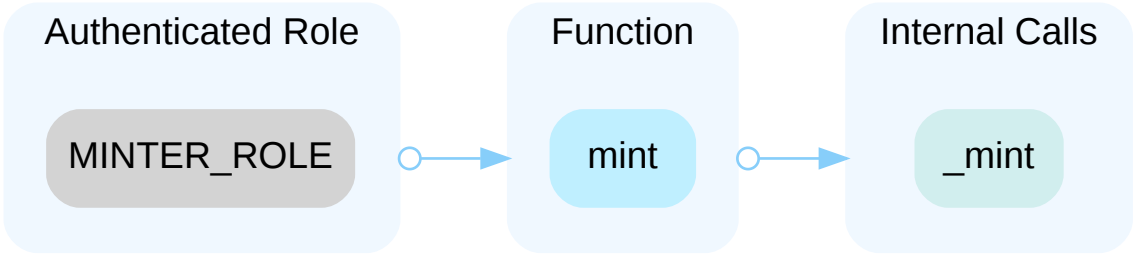
- function burn(): Accounts assigned the `BURNER_ROLE` can burn their own `DAD` tokens.
- function burnFrom(): Accounts that have been assigned the `BURNER_ROLE` can use their allowance to burn `DAD` tokens belonging to other accounts.



Any compromise to the `BURNER_ROLE` accounts may allow the hacker to take advantage of this authority and burn `DAD` tokens.

In the contract `DADToken`, the role `MINTER_ROLE` has authority over the functions shown in the list and diagram below:

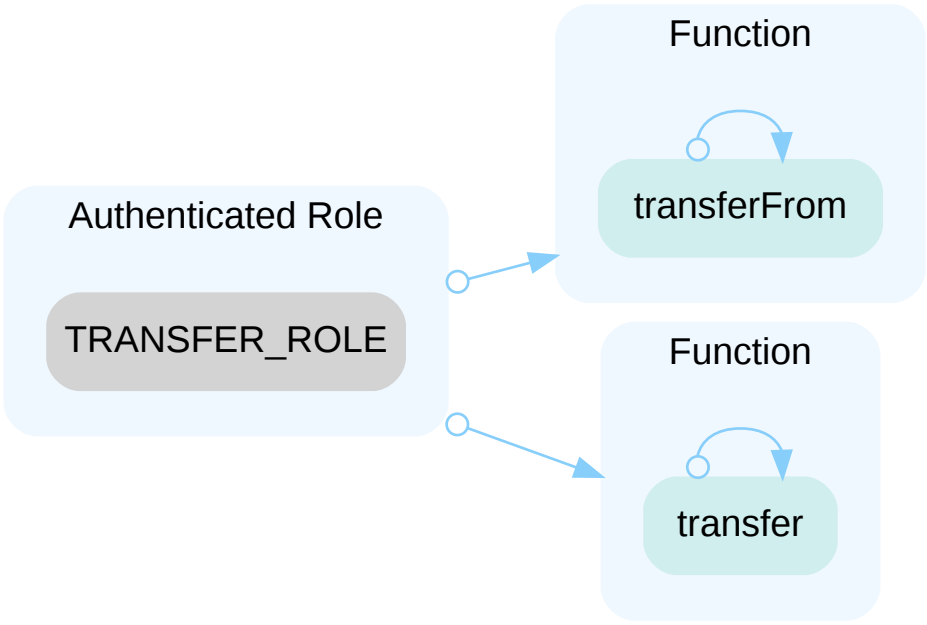
- function mint(): Accounts assigned the `MINTER_ROLE` can mint arbitrary amounts of `DAD` tokens to any account.



Any compromise to the `MINTER_ROLE` accounts may allow the hacker to take advantage of this authority and mint `DAD` tokens to any accounts.

In the contract `DADToken`, the role `TRANSFER_ROLE` has authority over the functions shown in the list and diagram below:

- function transfer(): Accounts assigned the `TRANSFER_ROLE` can transfer their own `DAD` tokens.
- function transferFrom(): Accounts that have been assigned the `TRANSFER_ROLE` can use their allowance to transfer `DAD` tokens belonging to other accounts.

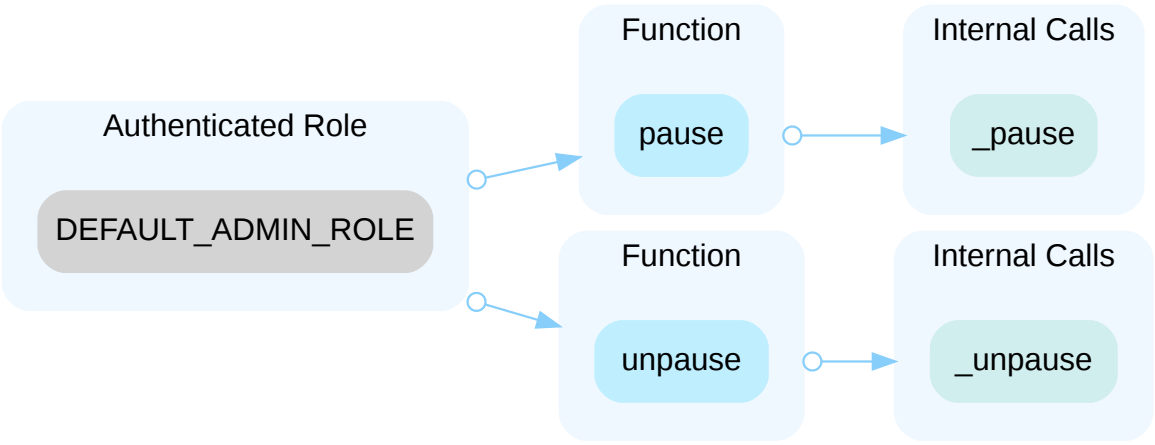


Any compromise to the `TRANSFER_ROLE` accounts may allow the hacker to take advantage of this authority and transfer `DAD` tokens.

In the contract `DADToken`, the role `DEFAULT_ADMIN_ROLE` inherited from `AccessControl` contract has authority over the functions shown below:

- function pause(): Accounts assigned the `DEFAULT_ADMIN_ROLE` can pause the contract, which will prevent the transfers or burning of `DAD` tokens.

- function unpause(): Accounts assigned the `DEFAULT_ADMIN_ROLE` can unpause the contract, which will allow the transfers or burning of `DAD` tokens.

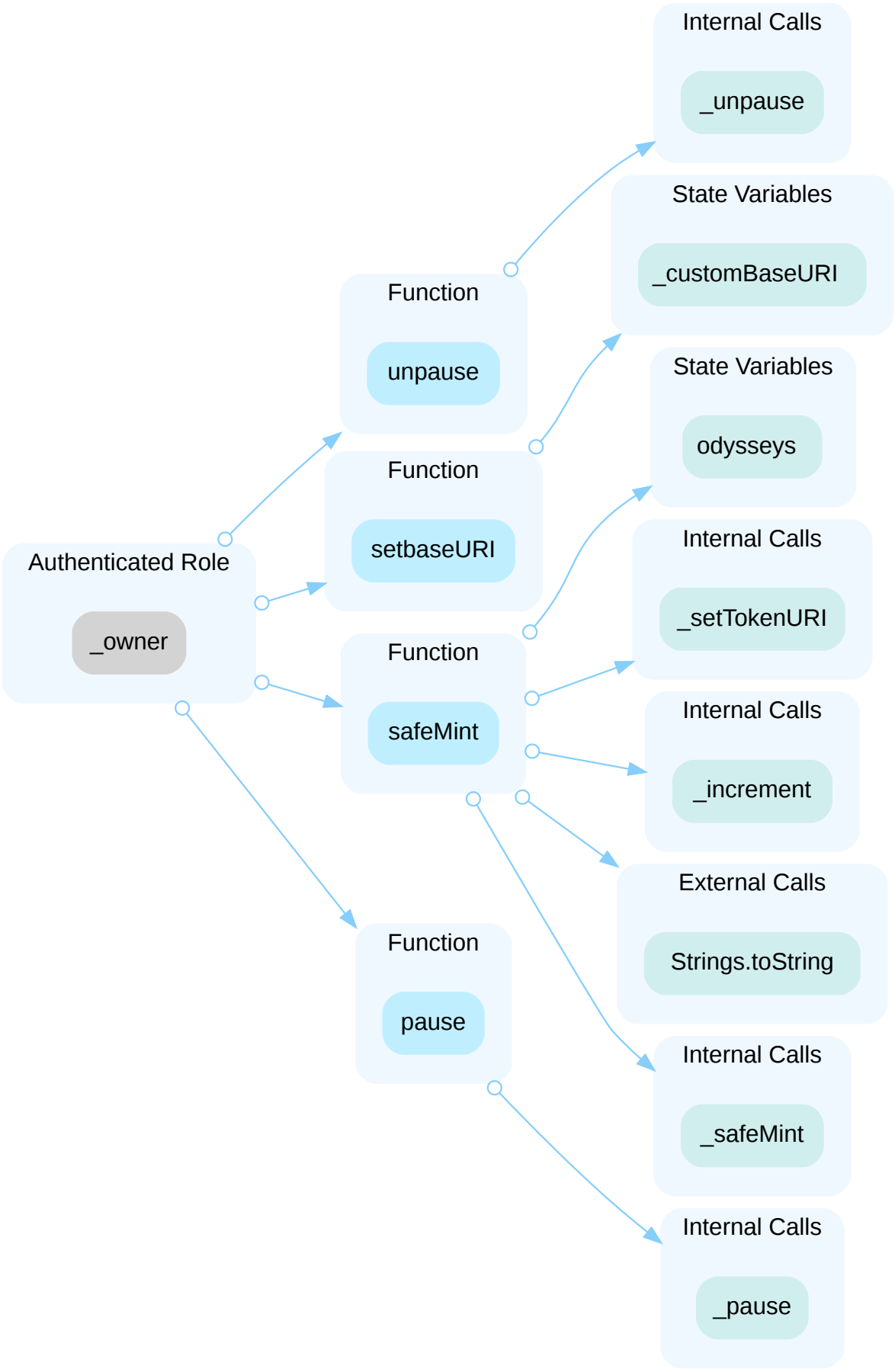


Accounts granted with `DEFAULT_ADMIN_ROLE` can grant or revoke other roles. By default, the deployer of `DADToken` contract has been granted all these roles.

- DEFAULT_ADMIN_ROLE
- TRANSFER_ROLE
- BURNER_ROLE
- MINTER_ROLE

Any compromise to the `DEFAULT_ADMIN_ROLE` accounts may allow the hacker to take advantage of this authority and pause/unpause `DAD` tokens to be transferred, mint large number of `DAD` tokens to any account, as well as operate the same actions as mentioned above.

In the contract `OdysseyNFT` the role `_owner` has authority over the functions shown in the diagram below.

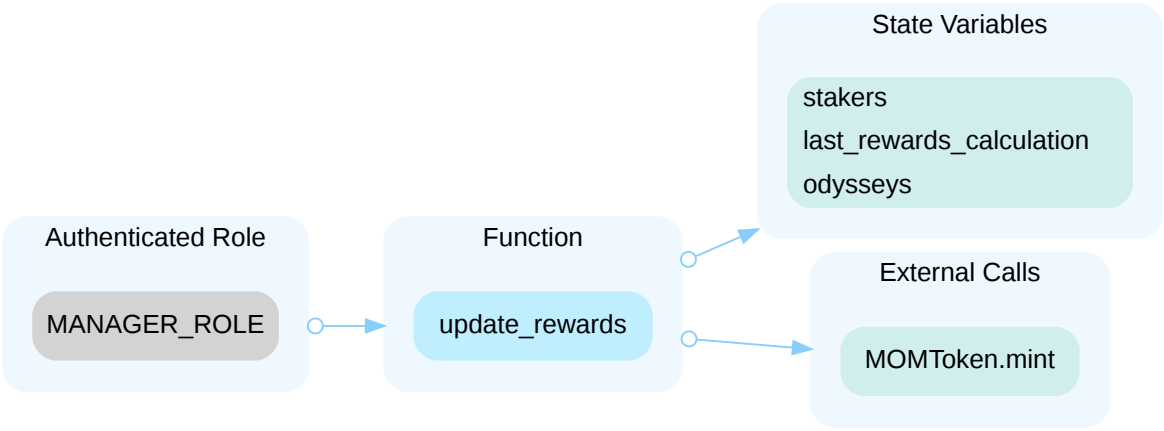

Internal Calls
_unpause
State Variables
_customBaseURI
Function
unpause
State Variables
odysseys
Function
setbaseURI
Internal Calls
_setTokenURI
Authenticated Role
_owner
Function
safeMint
Internal Calls
_increment
External Calls
Strings.toString
Function
pause
Internal Calls
_safeMint
Internal Calls
_pause

Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and execute the following actions:

- execute function `pause()` to permit the contract to mint or transfer;
- execute function `unpause()` to allow the contract to mint or transfer;
- execute function `safeMint()` to mint OdysseyNFT ;
- execute function `setbaseURI()` to set an unexpected base URI for OdysseyNFT;

In the contract `Staking` , the role `MANAGER_ROLE` has authority over the functions shown in the list and diagram below:

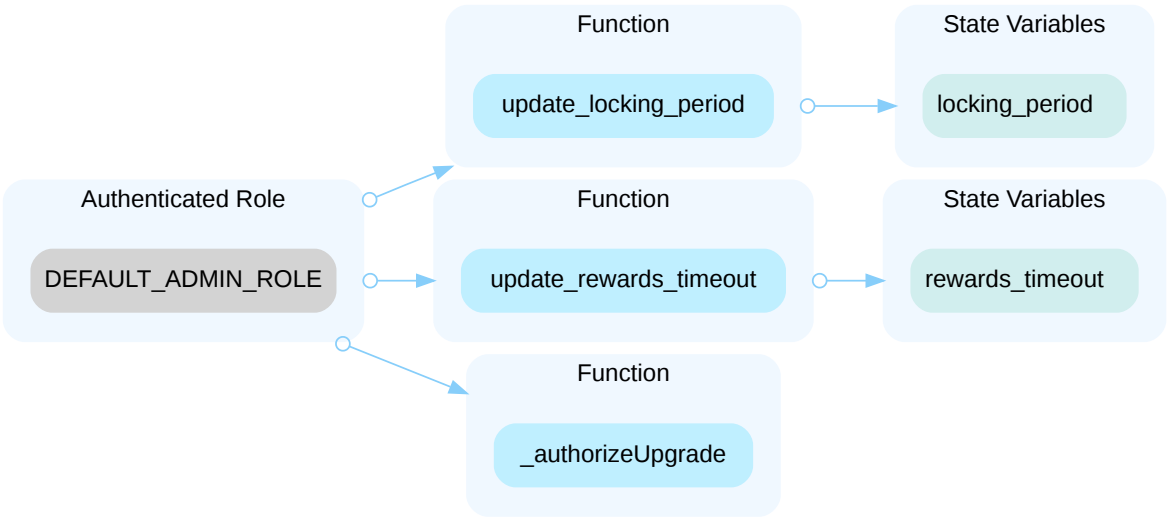
- function update_rewards(): Accounts assigned the `MANAGER_ROLE` can update rewards for users who staked in Odyssey and the owner of Odyssey NFT.



Any compromise to the `MANAGER_ROLE` accounts may allow the hacker to take advantage of this authority and update the rewards in an unexpected behavior.

In the contract `Staking` , the role `DEFAULT_ADMIN_ROLE` inherited from `AccessControl` contract has authority over the functions shown in the list and diagram below:

- function _authorizeUpgrade(): Accounts assigned the `DEFAULT_ADMIN_ROLE` can upgrade the implementation of this contract.
- function update_locking_period(): Accounts assigned the `MANAGER_ROLE` can update the value of `locking_period` .
- function update_rewards_timeout(): Accounts assigned the `MANAGER_ROLE` can update the value of `rewards_timeout` .



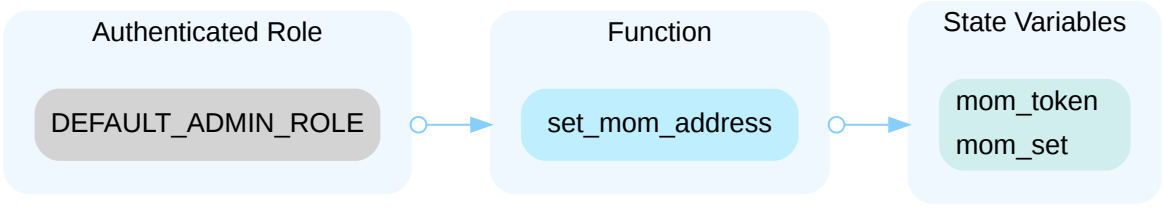
Accounts granted with `DEFAULT_ADMIN_ROLE` can grant or revoke other roles as mentioned above. By default, the deployer of `Staking` contract has been granted all these roles.

- DEFAULT_ADMIN_ROLE
- MANAGER_ROLE

Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and execute the above actions in an unexpected behavior.

In the contract `Vesting` the role `DEFAULT_ADMIN_ROLE` has authority over the functions shown in the list and diagram below.

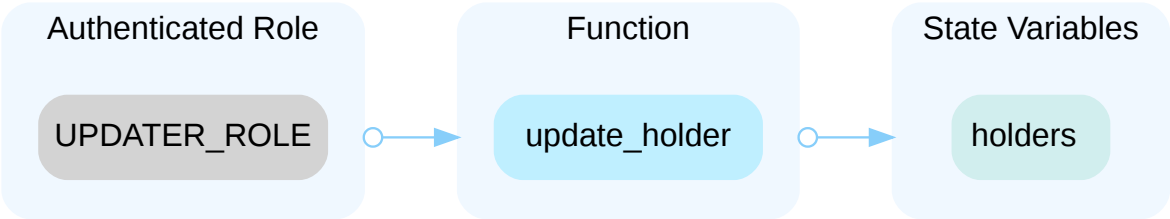
- function set_mom_address(): Accounts assigned the `DEFAULT_ADMIN_ROLE` can set the address of `mom_token` only once.



Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and set the unexpected address of `mom_token` .

In the contract `Vesting` the role `UPDATER_ROLE` has authority over the functions shown in the list and diagram below.

- function update_holder(): Accounts assigned the `UPDATER_ROLE` can update the vesting amount of `DAD` holders.



Any compromise to the `UPDATER_ROLE` account may allow the hacker to take advantage of this authority and update the vesting amount of `DAD` holders.

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

[Momentum Team, 07/12/2023]: Issue acknowledged. This issue will be fixed in the future, which will not be included in this audit engagement.

For now we will mitigate with different wallets for the contracts, but as soon as a multi-sign wallet/solution is available for the network, we will change the ownerhisp/admin to the 2/3 or 3/5 wallets.

[CertiK, 07/12/2023]: While this strategy will reduce the risk, it's crucial to note that it has not completely eliminated it. CertiK strongly encourages the project team periodically revisit the private key security management of all above-listed addresses.

MTH-01 | INITIAL TOKEN DISTRIBUTION

Category	Severity	Location	Status
Centralization / Privilege	Major	contracts/token/MomToken.sol: 55	Acknowledged

Description

All of the initial supply of `MOM` tokens are sent to the contract deployer when deploying the contract. This is a potential centralization risk as the deployer can distribute `MOM` tokens without the consensus of the community.

```
 55     constructor(uint256 initialSupply, address _vesting, address _dad) ERC20(
"Momentum", "MOM") {
 56         require(_vesting != address(0) && _dad != address(0),
"Address should not be 0");
 57         // assigning all roles to the deployer (owner)
 58         _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
 59         _grantRole(PAUSER_ROLE, msg.sender);
 60         _grantRole(MINTER_ROLE, msg.sender);
 61         _grantRole(BURNER_ROLE, msg.sender);
 62
 63         vesting = _vesting;
 64         dad = _dad;
 65
 66         // mint the initial supply
 67         _mint(msg.sender, initialSupply);
 68     }
```

Recommendation

We recommend transparency through providing a breakdown of the intended initial token distribution in a public location. We also recommend the team make an effort to restrict the access of the corresponding private key.

Alleviation

[Momentum Team, 06/08/2023]: The team attached the CAP-table shown as below providing a breakdown of the intended initial token distribution. This table will be part of the to be released lightpaper that will be made publicly available through our website.

Cap-table

Percentage	Amount	Allocation	Comment
45%	440,000,000.00	Users	Minting of tokens by users, according to activities, incentives for collaboration and achieving results.
2%	21,000,000.00	Ecosystem partners (Users)	Loyalty airdrop for past and current clients, partners and winning teams. In case this is not claimed, the tokens will be allocated to the users.
7%	70,000,000.00	Liquidity provision for (decentralized) exchanges	Provide sufficient liquidity to trade on exchanges → enable market making.
10%	100,000,000.00	Founders / Team	For the founders/ team, seed and private sale token holders, there will be a one year lock-up period, starting from the Token Generation Event, when the mainnet goes live. After this first lock-up year there will be a vesting period of two years, unlocking pro rata per block (~4,16%).
10%	100,000,000.00	Seed sale	
10%	100,000,000.00	Private sale	
15%	150,000,000.00	Foundation	Treasury of the foundation, for future development and growth of the system and its ecosystem.
100%	***981,000,000.00***		1G

In terms of restricting access to the corresponding private key (or keys since this applies to all the keys) we have implemented a number of measures.

High risk roles are managed by a 3/5 multi-sig. All other roles are managed by a 2/3 multi-sig. Note: Exceptions are applicable for keys owned by smart contracts.

- Strict processes and procedures are implemented to use and manage and oversee the use of the keys.
- Multi-sig participants are chosen on the following factors:
    - The risk level induced should fit the job responsibility.
    - Individuals assigned must possess the necessary expertise to verify the validity of the proposed chance.
    - Organizational Structure
    - Regular evaluation
    - Due to the fact that Odyssey is not a large organization, the overall organizational structure and reporting lines should be evaluated each time when creating a new multi-sig or when organization changes take place. This ensures that roles are structured in a way that minimizes conflicts of interest and prevents a single individual from having excessive control or access
    - Momentum foundation
    - In order to compensate for not yet active decentralized governance, the Momentum foundation is incorporated. The goal of the Foundation is to govern the Momentum token ($MOM) ecosystem and the network as a permanent governing body. As such it will be actively involved in participating in the multi-sigs. The long-term goal of the Foundation is, once regulation allows, to become a DAO.

- The Momentum foundation is founded and located in The Netherlands. The foundation is established by drafting a deed of incorporation and articles of association by a civil-law notary and registered at the Chamber of Commerce. The foundation has a board, but no members and no shareholders. The Momentum foundation that does not aim to make a profit. In the event of any profits, it is necessary that these then are allocated to the cause or purpose of the foundation.

[CertiK, 06/08/2023]: While this strategy will reduce the risk, it's crucial to note that it has not completely eliminated it. CertiK strongly encourages the project team periodically revisit the private key security management of all above-listed addresses.

STK-05 | CENTRALIZED CONTROL OF CONTRACT UPGRADE

Category	Severity	Location	Status
Centralization	Major	contracts/staking/Staking.sol (938a1bdb7d02c98cc2afd097cb0d9981eb58499d): 18	Acknowledged

Description

The `Staking` contract is an upgradeable contract, meaning the owner can upgrade the contract without the community's commitment. If an attacker compromises the account, he can change the implementation of the contract and drain tokens from the contract.

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

[Momentum Team, 06/08/2023]: The team will have the following:

- Each Admin / Owner will be a separate multi sign wallet (2/3 or 3/5).
- Pauser and Burner for MOM will be removed, replaced by the ADMIN.
- Other functions than Admin/Owner are designed for contract.
- Staking is TRANSFER role for DAD, MINTER role for MOM.
- Staking MANAGER role is a backend software that will ONLY update the rewards.
- In the near future, we will have a Vesting contract, that will be BURNER of DAD.

For now, we do have not created the multi sign wallets, so we can't provide the mitigation.
No time-lock will be implemented at the moment.
In the future we have plans to became a DAO.

[CertiK, 06/08/2023]: While this strategy will reduce the risk, it's crucial to note that it has not completely eliminated it. CertiK strongly encourages the project team periodically revisit the private key security management of all above-listed addresses.

STN-01 | WRONG REWARDS CLAIMED FROM ODDYSEY

Category	Severity	Location	Status
Coding Issue	Major	contracts/staking/Staking.sol: 684, 688, 689	Resolved

Description

In the function `_claim_rewards(uint256 odyssey_id)` of contract `Staking`, the Odyssey owner can claim rewards from specific `OdysseyNFT`. The issue is that it retrieves the `dad_amount` and `mom_amount` rewards from the `stakers` mapping, which is not correct. These variables should be retrieved from the `odysseys` mapping instead, since the rewards are associated with the specific `OdysseyNFT` token with `odyssey_id`.

```
684     function _claim_rewards(uint256 odyssey_id) private {
685         require(odysseys[odyssey_id].total_rewards > 0, "No rewards available")
;
686         require(OdysseyNFT(odyssey_nfts).ownerOf(odyssey_id) == msg.sender,
"Not owner of that Odyssey");
687
688         uint256 dad_amount = stakers[msg.sender].dad_rewards;
689         uint256 mom_amount = stakers[msg.sender].mom_rewards;
690         odysseys[odyssey_id].total_rewards = 0;
691         odysseys[odyssey_id].dad_rewards = 0;
692         odysseys[odyssey_id].mom_rewards = 0;
693
694         if(dad_amount != 0) {
695             DADToken(dad_token).mint(payable(msg.sender), dad_amount);
696         }
697
698         if(mom_amount != 0) {
699             MOMToken(mom_token).mint(payable(msg.sender), mom_amount);
700         }
701
702         emit tru(odyssey_id, dad_amount, mom_amount);
703     }
```

This issue could potentially lead to incorrect reward distributions and cause confusion or disputes among NFT owners. It's important to ensure that the rewards are correctly associated with the specific `OdysseyNFT` token to avoid any potential issues.

Scenario

1. Owner_Odyssey_1 stakes 5000000000 MOM tokens in Odyssey#1
2. Owner_Odyssey_1 stakes 5000000000 DAD tokens in Odyssey#1
3. Owner_Odyssey_2 stakes 5000000000 MOM tokens in Odyssey#1

4. Owner_Odyssey_2 stakes 5000000000 MOM tokens in Odyssey#2
5. Manager updates rewards
6. Owner_Odyssey_1 claims rewards
7. Owner_Odyssey_2 claims rewards
8. Owner_Odyssey_1 claims rewards from Odyssey_1
9. Owner_Odyssey_2 claims rewards from Odyssey_2

Odyssey owners claim zero reward from Odyssey NFT as the rewards are retrieved incorrectly.

Proof of Concept

The following proof of concept uses Foundry to test the case the `OdysseyNFT` claims the wrong rewards.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "forge-std/Test.sol";
import "../contracts/staking/Staking.sol";
import "../contracts/token/DADToken.sol";
import "../contracts/token/MomToken.sol";
import "../contracts/nft/OdysseyNFT.sol";
import "@openzeppelin/contracts/proxy/ERC1967/ERC1967Proxy.sol";

contract UUPSProxy is ERC1967Proxy {
    constructor(address _implementation, bytes memory _data)
    ERC1967Proxy(_implementation, _data){}
}

contract StakingNewTest is Test {

    Staking public stakingImpl;
    UUPSProxy public stakingProxy;
    Staking public staking;

    MOMToken public mToken;
    DADToken public dToken;
    OdysseyNFT public odysseyNFT;
    Vesting public vesting;

    address private constant Bob = address(0x123);
    address private constant Tom = address(0x456);
    address private constant Owner_Odyssey_1 = address(0x789);
    address private constant Owner_Odyssey_2 = address(0x91011);
    uint256 private odyssey_1;
    uint256 private odyssey_2;
    address private treasury;

    uint256 private constant treasury_amount = 1e10;

    address[] addresses = [Bob, Tom];
    uint256[] odysseys_ids;
    uint256[] stakers_amounts_mom = [5e9, 5e9];
    uint256[] stakers_amounts_dad = [5e9, 5e9];
    uint256[] odysseys_amounts_mom = [5e9, 5e9];
    uint256[] odysseys_amounts_dad = [5e9, 5e9];
    address[] NFTOwners = [Owner_Odyssey_1, Owner_Odyssey_2];

    function setUp() public {
        treasury = makeAddr("treasury");
        dToken = new DADToken();
        vesting = new Vesting(address(dToken), block.timestamp + 1 days);
```

```
        dToken.grantRole(dToken.BURNER_ROLE(), address(vesting));
        mToken = new MOMToken(1e20, address(vesting), address(dToken));
        vesting.set_mom_address(address(mToken));
        // implementation of staking contract
        stakingImpl = new Staking();
        // deploy proxy contract and point it to implementation
        stakingProxy = new UUPSProxy(address(stakingImpl), "");
        // warp in ABI
        staking = Staking(address(stakingProxy));

        //staking = new Staking();
        odysseyNFT = new OdysseyNFT(
            "Odyssey_NFT",
            "ODS",
            "ipfs://odyssey.nft"
        );

        staking.initialize(
            address(mToken),
            address(dToken),
            address(odysseyNFT),
            treasury
        );

        mToken.transfer(Bob, 2e10);
        mToken.transfer(Tom, 2e10);
        dToken.mint(Bob, 2e10);
        dToken.mint(Tom, 2e10);

        vm.startPrank(Bob);
        mToken.approve(address(staking), 2e20);
        dToken.approve(address(staking), 2e20);
        dToken.approve(address(vesting), 2e20);
        vm.stopPrank();

        vm.startPrank(Tom);
        mToken.approve(address(staking), 2e20);
        dToken.approve(address(staking), 2e20);
        dToken.approve(address(vesting), 2e20);
        vm.stopPrank();

        dToken.grantRole(keccak256("TRANSFER_ROLE"), address(staking));
        mToken.grantRole(keccak256("MINTER_ROLE"), address(staking));
        dToken.grantRole(keccak256("MINTER_ROLE"), address(staking));
        vesting.grantRole(vesting.UPDATER_ROLE(), address(mToken));

        odysseyNFT.safeMint(Owner_Odyssey_1);
        odyssey_1 = odysseyNFT.currentId();
        odysseyNFT.safeMint(Owner_Odyssey_2);
```

```
        odyssey_2 = odysseyNFT.currentId();
        odysseys_ids = [odyssey_1, odyssey_2];

        mToken.mint(Owner_Odyssey_1, 2e20);
        mToken.mint(Owner_Odyssey_2, 2e20);
        dToken.mint(Owner_Odyssey_1, 2e20);
        dToken.mint(Owner_Odyssey_2, 2e20);

        vm.startPrank(Owner_Odyssey_1);
        mToken.approve(address(staking), 2e20);
        dToken.approve(address(staking), 2e20);
        dToken.approve(address(vesting), 2e20);
        vm.stopPrank();

        vm.startPrank(Owner_Odyssey_2);
        mToken.approve(address(staking), 2e20);
        dToken.approve(address(staking), 2e20);
        dToken.approve(address(vesting), 2e20);
        vm.stopPrank();

        vm.label(Bob, "Bob");
        vm.label(Tom, "Tom");
        vm.label(Owner_Odyssey_1, "Owner_Odyssey_1");
        vm.label(Owner_Odyssey_2, "Owner_Odyssey_2");

    }

    function printTotalStaked() private view {
        console2.log("Currently total staked is %d", staking.total_staked());
    }

    function printStakerFor(address user) private {
        Staking.Staker memory staker;
        (staker.user, staker.total_rewards,staker.dad_rewards, staker.mom_rewards,
staker.total_staked, staker.dad_amount, staker.mom_amount) = staking.stakers(user);
        console2.log("--------------------Staker info for user : %s-----------------
-", vm.getLabel(user));
        console2.log("user=%s, total_rewards=%d, total_staked=%d,",
            staker.user, staker.total_rewards, staker.total_staked
        );
        console2.log("dad_amount=%d, mom_amount=%d",
            staker.dad_amount, staker.mom_amount
        );
        console2.log("dad_rewards=%d, mom_rewards=%d",
            staker.dad_rewards, staker.mom_rewards
        );
    }

    function printOdysseyFor(uint256 odyssey_id) private view {
```

```
        Staking.Odyssey memory odyssey;
        (odyssey.odyssey_id, odyssey.total_staked_into, odyssey.total_stakers,
odyssey.total_rewards,odyssey.dad_rewards, odyssey.mom_rewards,
odyssey.staked_odysseys_index) = staking.odysseys(odyssey_id);
        console2.log("-------------------Odyssey info for odyssey_id : %d-----------
--------", odyssey_id);
        console2.log("odyssey_id= %d, total_staked_into= %d, total_stakers= %d,",
            odyssey.odyssey_id, odyssey.total_staked_into, odyssey.total_stakers
        );
        console2.log("total_rewards= %d, staked_odysseys_index= %d",
            odyssey.total_rewards, odyssey.staked_odysseys_index
        );
        console2.log("dad_rewards= %d, mom_rewards= %d",
            odyssey.dad_rewards, odyssey.mom_rewards
        );
    }

    function printStakedBy(uint256 odyssey_id) private {
        Staking.StakedBy memory stakedBy;
        Staking.StakedBy[] memory stakedBys = staking.get_staked_by(odyssey_id);
        for (uint i = 1; i < stakedBys.length; i++) {
            stakedBy = stakedBys[i];
            console2.log("-------------------StakedBy info for user: %s in
odyssey_id : %d-------------------",  vm.getLabel(stakedBy.user), odyssey_id);
            console2.log("user= %s, total_amount= %d, dad_amount= %d,",
                vm.getLabel(stakedBy.user), stakedBy.total_amount,
stakedBy.dad_amount
            );
            console2.log("mom_amount= %d, timestamp= %d, effective_timestamp_mom=
%d",
                stakedBy.mom_amount, stakedBy.timestamp,
stakedBy.effective_timestamp_mom
            );
            console2.log("effective_timestamp_dad= %d",
stakedBy.effective_timestamp_dad
            );
        }

    }

    function printStakedOdysseys() private view {
        uint256[] memory odysseys =  staking.get_staked_odysseys();
        console2.log("Length of `staked_odysseys`= %d", odysseys.length);
    }

    function printStatistic(address[] memory users) private {
        console2.log("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Print Info
Start~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
        printTotalStaked();
        printStakedOdysseys();
```

```
        for (uint i = 0; i < users.length; i++) {
            printStakerFor(users[i]);
        }
        for (uint i = 0; i < odysseys_ids.length; i++) {
            printOdysseyFor(odysseys_ids[i]);
            printStakedBy(odysseys_ids[i]);
        }
        printBalances(users);
        console2.log("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Print Info
End~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
    }

    function printBalances(address[] memory users) private {
        console2.log("--------------------Show Balances--------------------");
        for(uint i = 0; i < users.length; i++) {
            console2.log("%s's   balances: MOM = %d, DAD = %d",
vm.getLabel(users[i]) ,mToken.balanceOf(users[i]), dToken.balanceOf(users[i]));
        }
        console2.log("Staking balances: MOM = %d, DAD = %d",
mToken.balanceOf(address(staking)), dToken.balanceOf(address(staking)));
    }

    function test_NFTOwner_claimRewards() public {
        console2.log("Owner_Odyssey_1 stakes 5000000000 MOM tokens in Odyssey#1");
        vm.prank(Owner_Odyssey_1);
        staking.stake(odyssey_1, 5e9, Staking.Token.MOM);

        vm.warp(block.timestamp + 3 days);
        console2.log("Owner_Odyssey_1 stakes 5000000000 DAD tokens in Odyssey#1");
        vm.prank(Owner_Odyssey_1);
        staking.stake(odyssey_1, 5e9, Staking.Token.DAD);

        console2.log("Owner_Odyssey_2 stakes 5000000000 MOM tokens in Odyssey#1");
        vm.prank(Owner_Odyssey_2);
        staking.stake(odyssey_1, 5e9, Staking.Token.MOM);
        console2.log("Owner_Odyssey_2 stakes 5000000000 MOM tokens in Odyssey#2");
        vm.prank(Owner_Odyssey_2);
        staking.stake(odyssey_2, 5e9, Staking.Token.MOM);

        console2.log("Manager updates rewards");
        staking.update_rewards(NFTOwners, stakers_amounts_mom, stakers_amounts_dad,
odysseys_ids, odysseys_amounts_mom,odysseys_amounts_dad,treasury_amount,
block.timestamp - 2 minutes);

        console2.log("Owner_Odyssey_1 claims rewards");
        vm.prank(Owner_Odyssey_1);
        staking.claim_rewards();
        console2.log("Owner_Odyssey_2 claims rewards");
        vm.prank(Owner_Odyssey_2);
        staking.claim_rewards();
```

```
        printStatistic(NFTOwners);

        console2.log("Owner_Odyssey_1 claims rewards from Odyssey_1");
        vm.prank(Owner_Odyssey_1);
        staking.claim_rewards(odyssey_1);
        console2.log("Owner_Odyssey_2 claims rewards from Odyssey_2");
        vm.prank(Owner_Odyssey_2);
        staking.claim_rewards(odyssey_2);

        printStatistic(NFTOwners);
    }
}
```

```
Running 1 test for test/StakingNewTest.t.sol:StakingNewTest
[PASS] test_NFTOwner_claimRewards() (gas: 1511164)
Logs:
  Owner_Odyssey_1 stakes 5000000000 MOM tokens in Odyssey#1
  Owner_Odyssey_1 stakes 5000000000 DAD tokens in Odyssey#1
  Owner_Odyssey_2 stakes 5000000000 MOM tokens in Odyssey#1
  Owner_Odyssey_2 stakes 5000000000 MOM tokens in Odyssey#2
  Manager updates rewards
  Owner_Odyssey_1 claims rewards
  Owner_Odyssey_2 claims rewards
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Print Info Start~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  Currently total staked is 20000000000
  Length of `staked_odysseys`= 2
  --------------------Staker info for user : Owner_Odyssey_1--------------------
  user=0x0000000000000000000000000000000000000789, total_rewards=0,
total_staked=10000000000,
  dad_amount=5000000000, mom_amount=5000000000
  dad_rewards=0, mom_rewards=0
  --------------------Staker info for user : Owner_Odyssey_2--------------------
  user=0x0000000000000000000000000000000000091011, total_rewards=0,
total_staked=10000000000,
  dad_amount=0, mom_amount=10000000000
  dad_rewards=0, mom_rewards=0
  --------------------Odyssey info for odyssey_id : 604472133179351442128897---------
----------
  odyssey_id= 604472133179351442128897, total_staked_into= 15000000000,
total_stakers= 2,
  total_rewards= 10000000000, staked_odysseys_index= 0
  dad_rewards= 5000000000, mom_rewards= 5000000000
  --------------------StakedBy info for user: Owner_Odyssey_1 in odyssey_id :
604472133179351442128897--------------------
  user= Owner_Odyssey_1, total_amount= 10000000000, dad_amount= 5000000000,
  mom_amount= 5000000000, timestamp= 259201, effective_timestamp_mom= 1
  effective_timestamp_dad= 129601
  --------------------StakedBy info for user: Owner_Odyssey_2 in odyssey_id :
604472133179351442128897--------------------
  user= Owner_Odyssey_2, total_amount= 5000000000, dad_amount= 0,
  mom_amount= 5000000000, timestamp= 259201, effective_timestamp_mom= 259201
  effective_timestamp_dad= 259201
  --------------------Odyssey info for odyssey_id : 604472133179351442128898---------
----------
  odyssey_id= 604472133179351442128898, total_staked_into= 5000000000,
total_stakers= 1,
  total_rewards= 10000000000, staked_odysseys_index= 1
  dad_rewards= 5000000000, mom_rewards= 5000000000
  --------------------StakedBy info for user: Owner_Odyssey_2 in odyssey_id :
604472133179351442128898--------------------
  user= Owner_Odyssey_2, total_amount= 5000000000, dad_amount= 0,
  mom_amount= 5000000000, timestamp= 259201, effective_timestamp_mom= 259201
  effective_timestamp_dad= 259201
```

```
  ------------------Show Balances------------------
  Owner_Odyssey_1's   balances: MOM = 200000000000000000000, DAD =
200000000000000000000
  Owner_Odyssey_2's   balances: MOM = 199999999995000000000, DAD =
200000000005000000000
  Staking balances: MOM = 15000000000, DAD = 5000000000
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Print Info End~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  Owner_Odyssey_1 claims rewards from Odyssey_1
  Owner_Odyssey_2 claims rewards from Odyssey_2
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Print Info Start~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  Currently total staked is 20000000000
  Length of `staked_odysseys`= 2
  ------------------Staker info for user : Owner_Odyssey_1------------------
  user=0x0000000000000000000000000000000000000789, total_rewards=0,
total_staked=10000000000,
  dad_amount=5000000000, mom_amount=5000000000
  dad_rewards=0, mom_rewards=0
  ------------------Staker info for user : Owner_Odyssey_2------------------
  user=0x0000000000000000000000000000000000091011, total_rewards=0,
total_staked=10000000000,
  dad_amount=0, mom_amount=10000000000
  dad_rewards=0, mom_rewards=0
  ------------------Odyssey info for odyssey_id : 604472133179351442128897---------
----------
  odyssey_id= 604472133179351442128897, total_staked_into= 15000000000,
total_stakers= 2,
  total_rewards= 0, staked_odysseys_index= 0
  dad_rewards= 0, mom_rewards= 0
  ------------------StakedBy info for user: Owner_Odyssey_1 in odyssey_id :
604472133179351442128897------------------
  user= Owner_Odyssey_1, total_amount= 10000000000, dad_amount= 5000000000,
  mom_amount= 5000000000, timestamp= 259201, effective_timestamp_mom= 1
  effective_timestamp_dad= 129601
  ------------------StakedBy info for user: Owner_Odyssey_2 in odyssey_id :
604472133179351442128897------------------
  user= Owner_Odyssey_2, total_amount= 5000000000, dad_amount= 0,
  mom_amount= 5000000000, timestamp= 259201, effective_timestamp_mom= 259201
  effective_timestamp_dad= 259201
  ------------------Odyssey info for odyssey_id : 604472133179351442128898---------
----------
  odyssey_id= 604472133179351442128898, total_staked_into= 5000000000,
total_stakers= 1,
  total_rewards= 0, staked_odysseys_index= 1
  dad_rewards= 0, mom_rewards= 0
  ------------------StakedBy info for user: Owner_Odyssey_2 in odyssey_id :
604472133179351442128898------------------
  user= Owner_Odyssey_2, total_amount= 5000000000, dad_amount= 0,
  mom_amount= 5000000000, timestamp= 259201, effective_timestamp_mom= 259201
  effective_timestamp_dad= 259201
  ------------------Show Balances------------------
```

```
  Owner_Odyssey_1's   balances: MOM = 200000000000000000000, DAD =
200000000000000000000
  Owner_Odyssey_2's   balances: MOM = 199999999995000000000, DAD =
200000000005000000000
  Staking balances: MOM = 15000000000, DAD = 5000000000
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Print Info End~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Test result: ok. 1 passed; 0 failed; finished in 7.61ms
```

Recommendation

It is recommended to modify the code to retrieve the rewards from the `odysseys` mapping as following:

```
function _claim_rewards(uint256 odyssey_id) private {
    require(odysseys[odyssey_id].total_rewards > 0, "No rewards available");
    require(OdysseyNFT(odyssey_nfts).ownerOf(odyssey_id) == msg.sender, "Not owner
of that Odyssey");

    uint256 dad_amount = odysseys[odyssey_id].dad_rewards;
    uint256 mom_amount = odysseys[odyssey_id].mom_rewards;
    odysseys[odyssey_id].total_rewards = 0;
    odysseys[odyssey_id].dad_rewards = 0;
    odysseys[odyssey_id].mom_rewards = 0;

    if(dad_amount != 0) {
        DADToken(dad_token).mint(payable(msg.sender), dad_amount);
    }

    if(mom_amount != 0) {
        MOMToken(mom_token).mint(payable(msg.sender), mom_amount);
    }

    emit OdysseyRewardsClaimed(odyssey_id, dad_amount, mom_amount);
}
```

Alleviation

[Momentum Team, 07/12/2023]: The team has fixed the issue in commit bfbe050cc7cf1454a304f36cf066049346509f84 by using the correct mapping.

OFT-02 FUNCTION `transferFrom` OF CONTRACT `OdysseyNFT` DOES NOT FAIL WHEN PAUSED

Category	Severity	Location	Status
Logical Issue	Medium	contracts/nft/OdysseyNFT.sol: 16	Resolved

Description

It is expected that any call of the form `transferFrom(from, to, tokenId)` to a paused contract must fail. However, in ERC721, some invocations of `transferFrom(from, to, tokenId)` succeed even when the contract is paused.

Recommendation

Ensure that the `transferFrom()` function reverts when the contract is paused.

Alleviation

[Momentum Team, 07/12/2023]: The team has fixed the issue in commits 8188a73c48bd16aad9fcab71dc1117eac5ce9115 and 8b5d331e81e75a28eeb953999c3a0b61182f154f by not allowing token transfers when the contract is paused.

STN-02 POSSIBLE FOR STAKES TO BE LOCKED IN THE CONTRACT

Category	Severity	Location	Status
Design Issue	Medium	contracts/staking/Staking.sol: 496, 566	Acknowledged

Description

The function `_unstake()` requires the `odyssey_id` to exist while unstaking.

```
451     function _unstake(uint256 odyssey_id, Token token) private
onlyMintedOdyssey(odyssey_id) {
```

Similarly, the function `_restake()` requires the `from_odyssey_id` to exist when removing a staking amount.

```
513     function _restake(uint256 from_odyssey_id, uint256 to_odyssey_id, uint256
 amount, Token token) private onlyMintedOdyssey(to_odyssey_id) onlyMintedOdyssey(
from_odyssey_id) {
```

However, Odysseys may be burned, meaning that a previously existed ID may not longer exist in the future. In this situation, users' stakes will be locked in the contract.

Recommendation

It is recommended to not require the an Odyssey ID to exist when removing stakes from the Odyssey.

Alleviation

[Momentum Team, 06/15/2023]: Yes, that is the intended behavior. We do not plan any burning of NFT's at the moment, but if an Odyssey NFT is burned, the user's deposit will be locked, since he/she staked in a NFT that was burned. That's the plan for now.

STN-03 | INACCURATE CALCULATION ON EFFECTIVE TIMESTAMP

Category	Severity	Location	Status
Coding Issue	Medium	contracts/staking/Staking.sol: 473~477, 480~484	Resolved

Description

In the function `_do_stake()` of contract `Staking`, the issue is that it increases the `dad_amount` or `mom_amount` before calculating the effective timestamp. This means that the effective timestamp is calculated based on the updated amount, rather than the original amount. This can result in incorrect effective timestamps being calculated, which can affect the distribution of rewards.

```
471             if(token == Token.DAD) {
472                 staked_by[odyssey_id][index].dad_amount += amount;
473                 staked_by[odyssey_id][index].effective_timestamp_dad =
474                     calculate_effective_timestamp(staked_by[odyssey_id][index].
timestamp,
475                         staked_by[odyssey_id][index].dad_amount,
476                         amount,
477                         true);
478             } else {
479                 staked_by[odyssey_id][index].mom_amount += amount;
480                 staked_by[odyssey_id][index].effective_timestamp_mom =
481                     calculate_effective_timestamp(staked_by[odyssey_id][index].
timestamp,
482                         staked_by[odyssey_id][index].mom_amount,
483                         amount,
484                         true);
485             }
```

Recommendation

It is recommended to modify the code to first calculate the effective timestamp, and then increase the `dad_amount` or `mom_amount`. For example:

```
471             if(token == Token.DAD) {
472                 staked_by[odyssey_id][index].effective_timestamp_dad =
473                     calculate_effective_timestamp(staked_by[odyssey_id][index].
timestamp,
474                         staked_by[odyssey_id][index].dad_amount,
475                         amount,
476                         true);
477                 staked_by[odyssey_id][index].dad_amount += amount;
478             } else {
479                 staked_by[odyssey_id][index].effective_timestamp_mom =
480                     calculate_effective_timestamp(staked_by[odyssey_id][index].
timestamp,
481                         staked_by[odyssey_id][index].mom_amount,
482                         amount,
483                         true);
484                 staked_by[odyssey_id][index].mom_amount += amount;
485             }
```

Alleviation

[Momentum Team, 07/12/2023]: The team has fixed the issue in commit 630be403faf6def215da2d9966773a5cecfab6a6 by using the correct values when calculating the effective timestamp.

VET-04 | POSSIBLE FOR LAST CLAIM DATE TO GO BACK IN TIME

Category	Severity	Location	Status
Logical Issue	Medium	contracts/vesting/Vesting.sol: 108	Resolved

Description

When updating a holder, the holder's `last_claim_date` is either set to the current timestamp or the starting timestamp.

```
 108         holders[holder].last_claim_date = holders[holder].last_claim_date == 0
&& current_timestamp >= starting_date
 109             ? current_timestamp
 110             : starting_date;
```

In the situation when a holder has their `last_claim_date` set already and the vesting period has started, then updating the holder will change their `last_claim_date` to be `starting_date`, which is always less than or equal to the old `last_claim_date`.

This allows the holder to immediately redeem tokens instead of waiting a vesting period.

Recommendation

When updating an already existing holder after the start of the vesting period, it is recommended to redeem all tokens for the holder and then update the holder's `last_claim_date` to the current timestamp.

Alleviation

[Momentum Team, 07/12/2023]: The team has fixed the issue in commit 344f4a8e7e6e932e6396142c73a3712abf864757 by only allowing the `last_claim_date` to be set once.

Note that if the function is called again for the same holder, the `last_claim_date` will not change and this is the intended design.

OFT-01 FUNCTION `burn()` OF `OdysseyNFT` CONTRACT CANNOT BE PAUSED

Category	Severity	Location	Status
Coding Issue	Minor	contracts/nft/OdysseyNFT.sol: 146	Resolved

Description

The `burn()` function in the `OdysseyNFT` contract is not impacted by contract suspension. This means that even if the contract is paused, the owner of the non-fungible token can still burn it in the contract.

```
146     function burn(uint256 tokenId) public {
147         _requireMinted(tokenId);
148         require(_isApprovedOrOwner(_msgSender(), tokenId),
"caller is not token owner or approved");
149         odysseys--;
150         _burn(tokenId);
151     }
```

Recommendation

It is recommended to disallow burning when the contract is paused.

Alleviation

[Momentum Team, 07/12/2023]: The team has fixed the issue in commit 07d7cec02b4911ad662496a99a4a13500d7dfbc6 by disallowing burning when the contract is paused.

STN-04 | INCOMPATIBILITY WITH DEFLATIONARY TOKENS

Category	Severity	Location	Status
Logical Issue	Minor	contracts/staking/Staking.sol: 355, 415, 418, 480~484, 762	Acknowledged

Description

When transferring deflationary ERC20 tokens, the input amount may not be equal to the received amount due to the charged transaction fee. For example, if a user sends 100 deflationary tokens (with a 10% transaction fee), only 90 tokens actually arrived to the contract. However, a failure to discount such fees may allow the same user to withdraw 100 tokens from the contract, which causes the contract to lose 10 tokens in such a transaction.

```
355         _stake(odyssey_id, amount, token);
```

- Transferring tokens by `amount`.
- This function call executes the following operation.
- In `Staking._stake`,
  - `IERC20(mom_token).safeTransferFrom(payable(msg.sender), address(this), amount);`

```
355         _stake(odyssey_id, amount, token);
```

- This function call executes the following operation.
- In `Staking._stake`,
  - `_do_stake(odyssey_id, amount, token);`
- In `Staking._do_stake`,
  - `staked_by[odyssey_id][index].effective_timestamp_mom = calculate_effective_timestamp(staked_by[odyssey_id][index].timestamp, staked_by[odyssey_id][index].mom_amount, amount, true);`
- In `Staking.calculate_effective_timestamp`,
  - `uint256 amount_diff = current_amount - amount > 0 ? current_amount - amount : 1;`
- The `amount` appears to be used for bookkeeping purposes without compensating the potential transfer fees.

In this project, the `mom_token` address should be ideally set to the address of deployed `MOMToken` contract which doesn't

support transferring fee.

Recommendation

We advise the client to regulate tokens and ensure the `mom_token` or `dad_token` addresses which should be set to the addresses of deployed `MOMToken` and `DADToken` contracts.

Alleviation

[Momentum Team, 07/12/2023]: The team has acknowledged this issue. `mom_toke` n and `dad_token` will be set to the respective addresses of the deployed MOM and DAD tokens on deployment.

STN-06 USAGE OF `effective_timestamp` IN FUNCTION `calculate_effective_timestamp()`

Category	Severity	Location	Status
Design Issue	Minor	contracts/staking/Staking.sol: 753	Resolved

Description

The `calculate_effective_timestamp()` function is used to calculate the effective timestamp for a specific token type (MOM or DAD) based on the latest timestamp and the amount of tokens being staked or unstaked. Given the usages as below:

```
580             staked_by_from.effective_timestamp_dad =
calculate_effective_timestamp(staked_by_from.timestamp,
581
                                                        staked_by_from.
dad_amount,
582                                                         amount,
583                                                         false);
```

```
588             staked_by_from.effective_timestamp_mom =
calculate_effective_timestamp(staked_by_from.timestamp,
589
                                                        staked_by_from.
mom_amount,
590                                                         amount,
591                                                         false);
```

Here the `staked_by_from.timestamp` refers to the last timestamp of staking on the Odyssey, considering the following case:

1. Bob stakes 1000 `MOM` on Odyssey#1 in July 8
2. Bob stakes 1000 `DAD` on Odyssey#1 in July 10
3. Bob restakes 500 `MOM` from Odyssey#1 to Odyssey#2 in July 11, while calculating the `staked_by_from.effective_timestamp_mom`, it will use the latest staking time `July 10`.

In the example given, if Bob restakes 500 `MOM` tokens from Odyssey#1 to Odyssey#2, the `staked_by_from.timestamp` parameter used to calculate the effective timestamp for the `MOM` tokens will be set to the last timestamp when Bob staked `DAD` tokens on any Odyssey, which is July 10. This means that the effective timestamp for the restaked MOM tokens will be based on the July 10 timestamp, rather than the July 8 timestamp when the original MOM tokens were staked on Odyssey#1.

Recommendation

If the intent is to calculate the effective timestamp based on the original staking or unstaking action for a specific token type, then the `staked_by_from.timestamp` should be updated to reflect the original staking or unstaking action, rather than the latest one.

Alleviation

[Momentum Team, 07/12/2023]: The team has fixed the issue in commit b436b4253dc6cbfb634874b9aee93df36b9abab1 by using the correct timestamp.

STN-07 | POSSIBLE ETHER LOCKED IN `Staking` CONTRACT

Category	Severity	Location	Status
Language-Specific	Minor	contracts/staking/Staking.sol: 354	Resolved

Description

The `stake()` function in the `Staking` contract is marked as `payable`, which means that users can transfer Ether along with their token stake. However, there is no function provided for the owner of the contract to withdraw Ether from the contract. This can result in Ether becoming trapped within the contract if someone accidentally sends Ether while calling the `stake()` function.

```
 354     function stake(uint256 odyssey_id, uint256 amount, Token token) public
payable {
 355         _stake(odyssey_id, amount, token);
 356     }
```

Since the project does not require Ether, it is recommended to remove the `payable` modifier from the `stake()` function to prevent users from accidentally sending Ether to the contract.

Alternatively, if it is necessary to accept Ether for some reason, a separate function should be added to allow the owner of the contract to withdraw any accumulated Ether from the contract. This can be achieved by adding a new function with the `onlyRole` modifier that transfers the Ether balance of the contract to the owner's address.

Recommendation

To prevent accidental locking of Ether in the contract, it is recommended to remove the `payable` modifier from the `stake()` function if Ether is not required. Alternatively, a separate function should be added to allow the owner of the contract to withdraw any accumulated Ether from the contract.

Alleviation

[Momentum Team, 07/12/2023]: The team has fixed the issue in commit 9aa1404ec692dfe281829ed532b484c7c2c58be7 by removing the `payable` keyword.

VET-02 | CHECK-EFFECTS-INTERACTIONS PATTERN VIOLATION

Category	Severity	Location	Status
Logical Issue	Minor	contracts/vesting/Vesting.sol: 140~141	Resolved

Description

In the `Vesting` contract, function `_redeem_tokens()` violates the Check-Effects-Interactions pattern. This pattern is a best practice for writing secure smart contracts that involves performing all state changes before making any external function calls.

External call(s)

```
140         dad_contract.burnFrom(msg.sender, total_to_redeem);
```

State variables written after the call(s)

```
141         holder.total_tokens = holder.total_tokens - total_to_redeem;
```

An external function call to the `dad_contract.burnFrom()` function is made before the state changes are made. This creates a potential vulnerability where an attacker could manipulate the contract's state during the external function call, which could lead to loss of funds or other malicious actions.

To prevent such potential issue, it is important to ensure that all external function calls are made after the state changes have been made.

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attacks.

Alleviation

[Momentum Team, 07/12/2023]: The team has fixed the issue in commit 6614640b7f3e3b03b52693b4c98ca23b454c8738 by adding a reentrancy guard as well as using the checks-effects-interactions pattern.

OFT-03 | UNUSED EVENT

Category	Severity	Location	Status
Coding Issue	Informational	contracts/nft/OdysseyNFT.sol: 45	Resolved

Description

Some events are never emitted, which can lead to confusion and code maintainability issues.

```
45      event StateUpdated(string indexed state, uint256 from, uint256 to);
```

- `StateUpdated` is declared in `OdysseyNFT` but never emitted.

Recommendation

It is recommended to remove the unused events or emit them in the intended functions to improve code clarity and maintainability.

Alleviation

[Momentum Team, 07/12/2023]: The team has fixed the issue in commit e1b4c74807d3703e8424daa4cfa5b3493a351661 by removing the unused event.

VET-01 | POTENTIALLY ARITHMETIC UNDERFLOW

Category	Severity	Location	Status
Coding Issue	Informational	contracts/vesting/Vesting.sol: 131~134	Resolved

Description

In the `Vesting` contract, there is a potential arithmetic underflow issue in function `_redeem_tokens` and it is actually caused by the subtraction of `holder.last_claim_date` from `current_timestamp` in the calculation of `total_to_redeem`.

```
131         uint256 total_to_redeem = current_timestamp < end_date
132             ? (holder.total_tokens * (current_timestamp - holder.
last_claim_date)) / (end_date - holder.last_claim_date)
133             :  holder.total_tokens;
```

The `holder.last_claim_date` is determined from `update_holder` function in which it initializes the holder's `last_claim_date` to the current timestamp if it has not been set yet and the current timestamp is greater than or equal to the `starting_date`, otherwise, `last_claim_date` is set to `starting_date`.

```
108         holders[holder].last_claim_date = holders[holder].last_claim_date == 0
&& current_timestamp >= starting_date
109             ? current_timestamp
110             : starting_date;
```

If `current_timestamp` is less than `holder.last_claim_date`, the result of the subtraction will be negative, leading to an arithmetic underflow. This can result in unexpected behavior.

Proof of Concept

The following proof of concept uses Foundry to test this issue.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "forge-std/Test.sol";
import "../contracts/staking/Staking.sol";
import "../contracts/token/DADToken.sol";
import "../contracts/token/MomToken.sol";
import "../contracts/nft/OdysseyNFT.sol";

contract VestingTest is Test {

    Vesting public vesting;
    MOMToken public mToken;
    DADToken public dToken;
    address private constant Bob = address(0x123);
    address private constant Tom = address(0x456);
    address[] users = [Bob, Tom];

    function setUp() public {
        dToken = new DADToken();
        vesting = new Vesting(address(dToken), block.timestamp + 1 days);
        dToken.grantRole(dToken.BURNER_ROLE(), address(vesting));
        mToken = new MOMToken(10000 * 1e18, address(vesting), address(dToken));
        vesting.set_mom_address(address(mToken));
        uint256 initialDADToken = 1000 * 1e18;
        for(uint i = 0; i < users.length; i++) {
            dToken.mint(users[i], initialDADToken);
            vm.prank(users[i]);
            dToken.approve(address(vesting), initialDADToken);
        }
        mToken.transfer(address(vesting), 5000 * 1e18);
        vesting.grantRole(vesting.UPDATER_ROLE(), address(this));
    }

    function test_updateHolder_beforeStartDate() internal {
        uint256 amount = 100 * 1e18;
        vesting.update_holder(Bob, amount);
    }


    function test_redeemTokens_beforeStartDate_reverts() public {
        test_updateHolder_beforeStartDate();
        vm.prank(Bob);
        vm.expectRevert();
        vesting.redeem_tokens();
    }
}
```

```
[PASS] test_redeemTokens_beforeStartDate_reverts() (gas: 68638)

Traces:
  [68638] VestingTest::test_redeemTokens_beforeStartDate_reverts()
    ├─ [0] console::log(currentTimestamp = %d, 1) [staticcall]
    │   └─ ← ()
    ├─ [51403] Vesting::update_holder(0x0000000000000000000000000000000000000123,
10000000000000000000 [1e20])
    │   ├─ emit HolderUpdated(holder: 0x0000000000000000000000000000000000000123,
amount: 10000000000000000000 [1e20], last_claim_date: 86401 [8.64e4])
    │   └─ ← ()
    ├─ [0] VM::prank(0x0000000000000000000000000000000000000123)
    │   └─ ← ()
    ├─ [0] VM::expectRevert()
    │   └─ ← ()
    ├─ [5050] Vesting::redeem_tokens()
    │   └─ ← "Arithmetic over/underflow"
    └─ ← ()

Test result: ok. 1 passed; 0 failed; finished in 4.19ms
```

Recommendation

To prevent this issue, a simple solution would be to add a check at the beginning of the function to ensure that `current_timestamp` is greater than `holder.last_claim_date` before performing the calculation. Alternatively, if it is expected that `current_timestamp` may be less than `holder.last_claim_date`, then the calculation should be modified to handle this case appropriately. One way to do this would be to set `total_to_redeem` to 0 if `current_timestamp` is less than `holder.last_claim_date`.

Alleviation

[Momentum Team, 07/12/2023]: The team has fixed the issue in commit f9a502c7c86d5fc0365a30c0fd3f9e5319fd9ba5 by adding a check that `current_timestamp` is greater than `holder.last_claim_date`.

VET-05 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Coding Style	Informational	contracts/vesting/Vesting.sol: 93	Resolved

Description

There should always be events emitted in the sensitive functions that are controlled by centralization roles.

Recommendation

It is recommended emitting events for the sensitive functions that are controlled by centralization roles.

Alleviation

[Momentum Team, 07/12/2023]: The team has fixed the issue in commit dbc6f6dc2a8bf118cacd7349d4bfa1119658f5a6 by adding and emitting an event.

OPTIMIZATIONS | MOMENTUM - ADDENDUM

ID	Title	Category	Severity	Status
CON-04	Variables That Could Be Declared As Immutable	Gas Optimization	Optimization	Resolved
STN-05	Inefficient Memory Parameter	Inconsistency	Optimization	Resolved
VET-03	Potentially Redeem Zero Amount	Gas Optimization	Optimization	Resolved

CON-04 | VARIABLES THAT COULD BE DECLARED AS IMMUTABLE

Category	Severity	Location	Status
Gas Optimization	Optimization	contracts/token/MomToken.sol: 35, 40; contracts/vesting/Vesting.sol: 33, 43, 48	Resolved

Description

The linked variables assigned in the constructor can be declared as `immutable`. Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since they will not be stored in storage.

Recommendation

We recommend declaring these variables as immutable. Please note that the `immutable` keyword only works in Solidity version `v0.6.5` and up.

Alleviation

[Momentum Team, 07/12/2023]: The team has fixed the issue in commit 95afb1c573cb2070194c83d9f49ad13ae796289e by declaring the variables as immutable.

STN-05 | INEFFICIENT MEMORY PARAMETER

Category	Severity	Location	Status
Inconsistency	Optimization	contracts/staking/Staking.sol: 276, 276, 276, 277, 277, 277	Resolved

Description

One or more parameters with `memory` data location are never modified in their functions and those functions are never called internally within the contract. Thus, their data location can be changed to `calldata` to avoid the gas consumption copying from calldata to memory.

```
276     function update_rewards(address[] memory addresses, uint256[] memory
stakers_amount_mom, uint256[] memory stakers_amount_dad,
```

`update_rewards` has memory location parameters: `addresses`, `stakers_amount_mom`, `stakers_amount_dad`, `odysseys_ids`, `odysseys_amount_mom`, `odysseys_amount_dad`.

Recommendation

We recommend changing the parameter's data location to `calldata` to save gas.

- For Solidity versions prior to 0.6.9, since public functions are not allowed to have calldata parameters, the function visibility also needs to be changed to `external`.
- For Solidity versions prior to 0.5.0, since parameter data location is implicit, changing the function visibility to `external` will change the parameter's data location to calldata as well.

Alleviation

[Momentum Team, 07/12/2023]: The team has fixed the issue in commit 990795204554a945f6499cfd28f6796ccb6fe61f by using `calldata`.

VET-03 | POTENTIALLY REDEEM ZERO AMOUNT

Category	Severity	Location	Status
Gas Optimization	Optimization	contracts/vesting/Vesting.sol: 127	Resolved

Description

In the `_redeem_tokens` function of `Vesting` contract, there is a possibility that holders of the `DAD` token may attempt to redeem zero `MOM` tokens.

```
127     function _redeem_tokens() private {
128         require(mom_set, "MOM address is not set yet");
129         Holder storage holder = holders[msg.sender];
130         uint current_timestamp = block.timestamp;
131         uint256 total_to_redeem = current_timestamp < end_date
132             ? (holder.total_tokens * (current_timestamp - holder.
last_claim_date)) / (end_date - holder.last_claim_date)
133             :  holder.total_tokens;
134         DADToken dad_contract = DADToken(dad_token);
135 
136         require(holder.total_tokens > 0, "No tokens to redeem");
137         require(dad_contract.balanceOf(msg.sender) >= total_to_redeem,
"Not enough balance to burn");
138         require(dad_contract.allowance(msg.sender, address(this)) >=
 total_to_redeem, "Allowance is needed");
139 
140         dad_contract.burnFrom(msg.sender, total_to_redeem);
141         holder.total_tokens = holder.total_tokens - total_to_redeem;
142         IERC20(mom_token).safeTransfer(payable(msg.sender), total_to_redeem);
143 
144         emit Redeemed(msg.sender, total_to_redeem);
145     }
```

Since redeeming zero tokens has no practical effect, this can result in unnecessary gas consumption without any benefit to the holder.

Proof of Concept

The following proof of concept uses Foundry to test the case `DAD` holder redeems zero `MOM` token.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "forge-std/Test.sol";
import "../contracts/staking/Staking.sol";
import "../contracts/token/DADToken.sol";
import "../contracts/token/MomToken.sol";
import "../contracts/nft/OdysseyNFT.sol";

contract VestingTest is Test {

    Vesting public vesting;
    MOMToken public mToken;
    DADToken public dToken;
    address private constant Bob = address(0x123);
    address private constant Tom = address(0x456);
    address[] users = [Bob, Tom];

    function setUp() public {
        dToken = new DADToken();
        vesting = new Vesting(address(dToken), block.timestamp + 1 days);
        dToken.grantRole(dToken.BURNER_ROLE(), address(vesting));
        mToken = new MOMToken(10000 * 1e18, address(vesting), address(dToken));
        vesting.set_mom_address(address(mToken));
        uint256 initialDADToken = 1000 * 1e18;
        for(uint i = 0; i < users.length; i++) {
            dToken.mint(users[i], initialDADToken);
            vm.prank(users[i]);
            dToken.approve(address(vesting), initialDADToken);
        }
        mToken.transfer(address(vesting), 5000 * 1e18);
        vesting.grantRole(vesting.UPDATER_ROLE(), address(this));
    }

    function test_redeemTokens_zeroAmount() public  {
        uint256 amount = 100 * 1e18;
        vm.warp(vesting.starting_date());
        console2.log("update_holder at end_date");
        vesting.update_holder(Bob, amount);
        vm.prank(Bob);
        vesting.redeem_tokens();
    }

}
```

```
[PASS] test_redeemTokens_zeroAmount() (gas: 110018)
Logs:
  update_holder at end_date

Traces:
  [110018] VestingTest::test_redeemTokens_zeroAmount()
    ├─ [2384] Vesting::starting_date() [staticcall]
    │   └─ ← 86401 [8.64e4]
    ├─ [0] VM::warp(86401 [8.64e4])
    │   └─ ← ()
    ├─ [0] console::log(update_holder at end_date) [staticcall]
    │   └─ ← ()
    ├─ [49293] Vesting::update_holder(0x0000000000000000000000000000000000000123,
10000000000000000000 [1e20])
    │   ├─ emit HolderUpdated(holder: 0x0000000000000000000000000000000000000123,
amount: 10000000000000000000 [1e20], last_claim_date: 86401 [8.64e4])
    │   └─ ← ()
    ├─ [0] VM::prank(0x0000000000000000000000000000000000000123)
    │   └─ ← ()
    ├─ [45765] Vesting::redeem_tokens()
    │   ├─ [2585] DADToken::balanceOf(0x0000000000000000000000000000000000000123)
[staticcall]
    │   │   └─ ← 100000000000000000000 [1e21]
    │   ├─ [2848] DADToken::allowance(0x0000000000000000000000000000000000000123,
Vesting: [0x2e234DAe75C793f67A35089C9d99245E1C58470b]) [staticcall]
    │   │   └─ ← 100000000000000000000 [1e21]
    │   ├─ [12088] DADToken::burnFrom(0x0000000000000000000000000000000000000123, 0)
    │   │   ├─ emit Approval(owner: 0x0000000000000000000000000000000000000123,
spender: Vesting: [0x2e234DAe75C793f67A35089C9d99245E1C58470b], value:
100000000000000000000 [1e21])
    │   │   ├─ emit Transfer(from: 0x0000000000000000000000000000000000000123, to:
0x0000000000000000000000000000000000000000, value: 0)
    │   │   └─ ← ()
    │   ├─ [9313] MOMToken::transfer(0x0000000000000000000000000000000000000123, 0)
    │   │   ├─ emit Transfer(from: Vesting:
[0x2e234DAe75C793f67A35089C9d99245E1C58470b], to:
0x0000000000000000000000000000000000000123, value: 0)
    │   │   └─ ← true
    │   ├─ emit Redeemed(holder: 0x0000000000000000000000000000000000000123, amount:
0)
    │   └─ ← ()
    └─ ← ()

Test result: ok. 1 passed; 0 failed; finished in 4.30ms

| contracts/vesting/Vesting.sol:Vesting contract |                 |       |
|       |         |
|------------------------------------------------|-----------------|-------|-------
|-------|---------|
```

```
| Deployment Cost                                  | Deployment Size |       |
|       |         |
| 1002075                                          | 5119            |       |
|       |         |
| Function Name                                    | min             | avg   | median
| max   | # calls |
| UPDATER_ROLE                                     | 240             | 240   | 240
| 240   | 1       |
| grantRole                                        | 27498           | 27498 | 27498
| 27498 | 1       |
| redeem_tokens                                    | 45765           | 45765 | 45765
| 45765 | 1       |
| set_mom_address                                  | 43175           | 43175 | 43175
| 43175 | 1       |
| starting_date                                    | 2384            | 2384  | 2384
| 2384  | 1       |
| update_holder                                    | 49293           | 49293 | 49293
| 49293 | 1       |
```

Recommendation

It is recommended to add a check for the `total_to_redeem` to ensure it is not zero. This would prevent unnecessary gas consumption when holders attempt to redeem zero tokens.

Alleviation

[Momentum Team, 07/12/2023]: The team has fixed the issue in commit 493d47b8b1990136c3cfe7099c2df6ecede8aa40 by adding a check to ensure `total_to_redeem` is positive.

FORMAL VERIFICATION | MOMENTUM - ADDENDUM

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 automated formal verification (symbolic model checking) to prove that well-known functions in the smart contracts adhere to their expected behavior.

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 Compliance with Pausable ERC-721

We verified the properties of the public interface of those token contracts that implement the pausable ERC-721 interface.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
erc721pausable-supportsinterface-correct-erc721	`supportsInterface` Signals Support for `ERC721`
erc721pausable-balanceof-succeed-normal	`balanceOf` Succeeds on Admissible Inputs
erc721pausable-balanceof-correct-count	`balanceOf` Returns the Correct Value
erc721pausable-balanceof-revert	`balanceOf` Fails on the Zero Address
erc721pausable-balanceof-no-change-state	`balanceOf` Does Not Change the Contract's State
erc721pausable-ownerof-succeed-normal	`ownerOf` Succeeds For Valid Tokens
erc721pausable-transferfrom-succeed-normal	`transferFrom` Succeeds on Admissible Inputs
erc721pausable-ownerof-correct-owner	`ownerOf` Returns the Correct Owner
erc721pausable-ownerof-revert	`ownerOf` Fails On Invalid Tokens
erc721pausable-ownerof-no-change-state	`ownerOf` Does Not Change the Contract's State
erc721pausable-getapproved-succeed-normal	`getApproved` Succeeds For Valid Tokens
erc721pausable-transferfrom-revert-pause	`transferFrom` Fails when Paused
erc721pausable-getapproved-correct-value	`getApproved` Returns Correct Approved Address

Property Name	Title
erc721pausable-getapproved-revert-zero	`getApproved` Fails on Invalid Tokens
erc721pausable-isapprovedforall-succeed-normal	`isApprovedForAll` Always Succeeds
erc721pausable-getapproved-change-state	`getApproved` Does Not Change the Contract's State
erc721pausable-isapprovedforall-correct	`isApprovedForAll` Returns Correct Approvals
erc721pausable-isapprovedforall-change-state	`isApprovedForAll` Does Not Change the Contract's State
erc721pausable-approve-set-correct	`approve` Sets Approval
erc721pausable-approve-succeed-normal	`approve` Returns for Admissible Inputs
erc721pausable-approve-revert-invalid-token	`approve` Fails For Calls with Invalid Tokens
erc721pausable-approve-revert-not-allowed	`approve` Prevents Unpermitted Approvals
erc721pausable-setapprovalforall-succeed-normal	`setApprovalForAll` Returns for Admissible Inputs
erc721pausable-approve-change-state	`approve` Has No Unexpected State Changes
erc721pausable-setapprovalforall-set-correct	`setApprovalForAll` Approves Operator
erc721pausable-setapprovalforall-multiple	`setApprovalForAll` Can Set Multiple Operators
erc721pausable-setapprovalforall-change-state	`setApprovalForAll` Has No Unexpected State Changes
erc721pausable-transferfrom-correct-increase	`transferFrom` Transfers the Complete Token in Non-self Transfers
erc721pausable-transferfrom-correct-approval	`transferFrom` Updates the Approval Correctly
erc721pausable-transferfrom-correct-one-token-self	`transferFrom` Performs Self Transfers Correctly
erc721pausable-transferfrom-correct-owner-from	`transferFrom` Removes Token Ownership of From
erc721pausable-transferfrom-correct-owner-to	`transferFrom` Transfers Ownership
erc721pausable-transferfrom-correct-balance	`transferFrom` Sum of Balances is Constant
erc721pausable-transferfrom-correct-state-balance	`transferFrom` Keeps Balances Constant Except for From and To
erc721pausable-transferfrom-correct-state-owner	`transferFrom` Has Expected Ownership Changes

Property Name	Title
erc721pausable-transferfrom-revert-invalid	`transferFrom` Fails for Invalid Tokens
erc721pausable-transferfrom-revert-from-zero	`transferFrom` Fails for Transfers From the Zero Address
erc721pausable-transferfrom-correct-state-approval	`transferFrom` Has Expected Approval Changes
erc721pausable-transferfrom-revert-to-zero	`transferFrom` Fails for Transfers To the Zero Address
erc721pausable-supportsinterface-metadata	`supportsInterface` Signals that ERC721Metadata is Implemented
erc721pausable-supportsinterface-succeed-always	`supportsInterface` Always Succeeds
erc721pausable-supportsinterface-correct-erc165	`supportsInterface` Signals Support for ERC165
erc721pausable-supportsinterface-correct-false	`supportsInterface` Returns `False` for Id 0xffffffff
erc721pausable-supportsinterface-no-change-state	`supportsInterface` Does Not Change the Contract's State
erc721pausable-transferfrom-revert-not-owned	`transferFrom` Fails if `From` Is Not Token Owner
erc721pausable-transferfrom-revert-exceed-approval	`transferFrom` Fails for Token Transfers without Approval

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:

Property Name	Title
erc20-transfer-succeed-normal	`transfer` Succeeds on Admissible Non-self Transfers
erc20-transfer-succeed-self	`transfer` Succeeds on Admissible Self Transfers
erc20-transfer-revert-zero	`transfer` Prevents Transfers to the Zero Address
erc20-transfer-correct-amount	`transfer` Transfers the Correct Amount in Non-self Transfers
erc20-transfer-correct-amount-self	`transfer` Transfers the Correct Amount in Self Transfers

Property Name	Title
erc20-transfer-exceed-balance	`transfer` Fails if Requested Amount Exceeds Available Balance
erc20-transfer-change-state	`transfer` Has No Unexpected State Changes
erc20-transfer-recipient-overflow	`transfer` Prevents Overflows in the Recipient's Balance
erc20-transfer-false	If `transfer` Returns `false`, the Contract State Is Not Changed
erc20-transfer-never-return-false	`transfer` Never Returns `false`
erc20-transferfrom-revert-from-zero	`transferFrom` Fails for Transfers From the Zero Address
erc20-transferfrom-revert-to-zero	`transferFrom` Fails for Transfers To the Zero Address
erc20-transferfrom-succeed-normal	`transferFrom` Succeeds on Admissible Non-self Transfers
erc20-transferfrom-succeed-self	`transferFrom` Succeeds on Admissible Self Transfers
erc20-transferfrom-correct-amount	`transferFrom` Transfers the Correct Amount in Non-self Transfers
erc20-transferfrom-correct-amount-self	`transferFrom` Performs Self Transfers Correctly
erc20-transferfrom-change-state	`transferFrom` Has No Unexpected State Changes
erc20-transferfrom-correct-allowance	`transferFrom` Updated the Allowance Correctly
erc20-transferfrom-fail-exceed-balance	`transferFrom` Fails if the Requested Amount Exceeds the Available Balance
erc20-transferfrom-fail-exceed-allowance	`transferFrom` Fails if the Requested Amount Exceeds the Available Allowance
erc20-transferfrom-fail-recipient-overflow	`transferFrom` Prevents Overflows in the Recipient's Balance
erc20-transferfrom-never-return-false	`transferFrom` Never Returns `false`
erc20-transferfrom-false	If `transferFrom` Returns `false`, the Contract's State Is Unchanged
erc20-totalsupply-succeed-always	`totalSupply` Always Succeeds
erc20-balanceof-succeed-always	`balanceOf` Always Succeeds
erc20-totalsupply-correct-value	`totalSupply` Returns the Value of the Corresponding State Variable
erc20-totalsupply-change-state	`totalSupply` Does Not Change the Contract's State

Property Name	Title
erc20-balanceof-correct-value	`balanceOf` Returns the Correct Value
erc20-allowance-succeed-always	`allowance` Always Succeeds
erc20-balanceof-change-state	`balanceOf` Does Not Change the Contract's State
erc20-allowance-correct-value	`allowance` Returns Correct Value
erc20-allowance-change-state	`allowance` Does Not Change the Contract's State
erc20-approve-succeed-normal	`approve` Succeeds for Admissible Inputs
erc20-approve-revert-zero	`approve` Prevents Approvals For the Zero Address
erc20-approve-correct-amount	`approve` Updates the Approval Mapping Correctly
erc20-approve-change-state	`approve` Has No Unexpected State Changes
erc20-approve-false	If `approve` Returns `false`, the Contract's State Is Unchanged
erc20-approve-never-return-false	`approve` Never Returns `false`

Verification Results

In the remainder of this section, we list all contracts where model checking of at least one property was not successful. There are several reasons why this could happen:

- Model checking reports a counterexample that violates the property. Depending on the counterexample,this occurs if
  - The specification of the property is too generic and does not accurately capture the intended behavior of the smart contract. In that case, the counterexample does not indicate a problem in the underlying smart contract. We report such instances as being "inapplicable".
  - The property is applicable to the smart contract. In that case, the counterexample showcases a problem in the smart contract and a correspond finding is reported separately in the Findings section of this report. In the following tables, we report such instances as "invalid". The distinction between spurious and actual counterexamples is done manually by the auditors.
- The model checking result is inconclusive. Such a result does not indicate a problem in the underlying smart contract. An inconclusive result may occur if
  - The model checking engine fails to construct a proof. This can happen if the logical deductions necessary are beyond the capabilities of the automated reasoning tool. It is a technical limitation of all proof engines and cannot be avoided in general.

- The model checking engine runs out of time or memory and did not produce a result. This can happen if automatic abstraction techniques are ineffective or of the state space is too big.

Detailed Results For Contract OdysseyNFT (contracts/nft/OdysseyNFT.sol) In Commit e10687248f1af24907587d5ece38abfb0754a94f

Verification of Compliance with Pausable ERC-721

Detailed results for function `supportsInterface`

Property Name	Final Result	Remarks
erc721pausable-supportsinterface-correct-erc721	True	
erc721pausable-supportsinterface-metadata	True	
erc721pausable-supportsinterface-succeed-always	True	
erc721pausable-supportsinterface-correct-erc165	True	
erc721pausable-supportsinterface-correct-false	True	
erc721pausable-supportsinterface-no-change-state	True	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc721pausable-balanceof-succeed-normal	True	
erc721pausable-balanceof-correct-count	True	
erc721pausable-balanceof-revert	True	
erc721pausable-balanceof-no-change-state	True	

Detailed results for function `ownerOf`

Property Name	Final Result	Remarks
erc721pausable-ownerof-succeed-normal	True	
erc721pausable-ownerof-correct-owner	True	
erc721pausable-ownerof-revert	True	
erc721pausable-ownerof-no-change-state	True	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc721pausable-transferfrom-succeed-normal	True	
erc721pausable-transferfrom-revert-pause	False	OFT-02 : Function `transferFrom` of Contract `OdysseyNFT` does NOT Fail Even Paused
erc721pausable-transferfrom-correct-increase	True	
erc721pausable-transferfrom-correct-approval	True	
erc721pausable-transferfrom-correct-one-token-self	True	
erc721pausable-transferfrom-correct-owner-from	True	
erc721pausable-transferfrom-correct-owner-to	True	
erc721pausable-transferfrom-correct-balance	True	
erc721pausable-transferfrom-correct-state-balance	True	
erc721pausable-transferfrom-correct-state-owner	True	
erc721pausable-transferfrom-revert-invalid	True	
erc721pausable-transferfrom-revert-from-zero	True	
erc721pausable-transferfrom-correct-state-approval	True	
erc721pausable-transferfrom-revert-to-zero	True	
erc721pausable-transferfrom-revert-not-owned	True	
erc721pausable-transferfrom-revert-exceed-approval	True	

Detailed results for function `getApproved`

Property Name	Final Result	Remarks
erc721pausable-getapproved-succeed-normal	True	
erc721pausable-getapproved-correct-value	True	
erc721pausable-getapproved-revert-zero	True	
erc721pausable-getapproved-change-state	True	

Detailed results for function `isApprovedForAll`

Property Name	Final Result	Remarks
erc721pausable-isapprovedforall-succeed-normal	True	
erc721pausable-isapprovedforall-correct	True	
erc721pausable-isapprovedforall-change-state	True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc721pausable-approve-set-correct	True	
erc721pausable-approve-succeed-normal	True	
erc721pausable-approve-revert-invalid-token	True	
erc721pausable-approve-revert-not-allowed	True	
erc721pausable-approve-change-state	True	

Detailed results for function `setApprovalForAll`

Property Name	Final Result	Remarks
erc721pausable-setapprovalforall-succeed-normal	True	
erc721pausable-setapprovalforall-set-correct	True	
erc721pausable-setapprovalforall-multiple	True	
erc721pausable-setapprovalforall-change-state	True	

Detailed Results For Contract DADToken (contracts/token/DADToken.sol) In Commit e10687248f1af24907587d5ece38abfb0754a94f

Verification of ERC-20 Compliance

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-succeed-normal	Inconclusive	
erc20-transfer-succeed-self	Inconclusive	
erc20-transfer-revert-zero	Inconclusive	
erc20-transfer-correct-amount	Inconclusive	
erc20-transfer-correct-amount-self	Inconclusive	
erc20-transfer-exceed-balance	Inconclusive	
erc20-transfer-change-state	Inconclusive	
erc20-transfer-recipient-overflow	Inconclusive	
erc20-transfer-false	Inconclusive	
erc20-transfer-never-return-false	Inconclusive	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	Inconclusive	
erc20-transferfrom-revert-to-zero	Inconclusive	
erc20-transferfrom-succeed-normal	Inconclusive	
erc20-transferfrom-succeed-self	Inconclusive	
erc20-transferfrom-correct-amount	Inconclusive	
erc20-transferfrom-correct-amount-self	Inconclusive	
erc20-transferfrom-change-state	Inconclusive	
erc20-transferfrom-correct-allowance	Inconclusive	
erc20-transferfrom-fail-exceed-balance	Inconclusive	
erc20-transferfrom-fail-exceed-allowance	Inconclusive	
erc20-transferfrom-fail-recipient-overflow	Inconclusive	
erc20-transferfrom-never-return-false	Inconclusive	
erc20-transferfrom-false	Inconclusive	

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 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 `approve`

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

Detailed Results For Contract MOMToken (contracts/token/MomToken.sol) In Commit e10687248f1af24907587d5ece38abfb0754a94f

Verification of ERC-20 Compliance

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	True	
erc20-transfer-correct-amount	True	
erc20-transfer-succeed-normal	False	Context not considered
erc20-transfer-succeed-self	False	Context not considered
erc20-transfer-correct-amount-self	True	
erc20-transfer-change-state	True	
erc20-transfer-false	True	
erc20-transfer-exceed-balance	True	
erc20-transfer-never-return-false	True	
erc20-transfer-recipient-overflow	False	Context not considered

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	True	
erc20-transferfrom-revert-to-zero	True	
erc20-transferfrom-correct-amount-self	True	
erc20-transferfrom-correct-amount	True	
erc20-transferfrom-succeed-normal	False	Context not considered
erc20-transferfrom-succeed-self	False	Context not considered
erc20-transferfrom-correct-allowance	True	
erc20-transferfrom-change-state	True	
erc20-transferfrom-fail-exceed-balance	True	
erc20-transferfrom-fail-exceed-allowance	True	
erc20-transferfrom-false	True	
erc20-transferfrom-never-return-false	True	
erc20-transferfrom-fail-recipient-overflow	False	Context not considered

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 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 `approve`

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

APPENDIX | MOMENTUM - ADDENDUM

Finding Categories

Categories	Description
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Coding Style	Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable.
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
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.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Details on Formal Verification

Technical description

Some Solidity smart contracts from this project have been formally verified using symbolic model checking. Each such contract was compiled into a mathematical model which 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 model also formalizes a simplified execution environment of the Ethereum blockchain and a verification harness that performs the initialization of the contract and all possible interactions with the contract. Initially, the contract state is initialized non-deterministically (i.e. by arbitrary values) and over-approximates the reachable state space of the contract throughout

any actual deployment on chain. All valid results thus carry over to the contract's behavior in arbitrary states after it has been deployed.

Assumptions and simplifications

The following assumptions and simplifications apply to our model:

- Gas consumption is not taken into account, i.e. we assume that executions do not terminate prematurely because they run out of gas.
- The contract's state variables are non-deterministically initialized before invocation of any of those functions. That ignores contract invariants and may lead to false positives. It is, however, a safe over-approximation.
- The verification engine reasons about unbounded integers. Machine arithmetic is modeled as operations on the congruence classes arising from the bit-width of the underlying numeric type. This ensures that over- and underflow characteristics are faithfully represented.
- Certain low-level calls and inline assembly are not supported and may lead to an ERC-20 token 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 definitions

All properties are expressed in linear temporal logic (LTL). For that matter, we treat each invocation of and each return from a public or an external function as a discrete time steps. Our analysis reasons about the contract's state upon entering and upon leaving public or external functions.

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:

- `started(f, [cond])` Indicates an invocation of contract function `f` within a state satisfying formula `cond`.
- `willSucceed(f, [cond])` Indicates an invocation of contract function `f` within a state satisfying formula `cond` and considers only those executions that do not revert.
- `finished(f, [cond])` Indicates that execution returns from contract function `f` in a state satisfying formula `cond`. Here, formula `cond` may refer to the contract's state variables and to the value they had upon entering the function (using the `old` function).
- `reverted(f, [cond])` Indicates that execution of contract function `f` was interrupted by an exception in a contract state satisfying formula `cond`.

The verification performed in this audit operates on a harness that non-deterministically invokes a function of the contract's public or external interface. All formulas are analyzed w.r.t. the trace that corresponds to this function invocation.

Description of ERC-20 Properties

The specifications are designed such that they capture the desired and admissible behaviors of the ERC-20 functions `transfer`, `transferFrom`, `approve`, `allowance`, `balanceOf`, and `totalSupply`.

In the following, we list those property specifications.

Properties for ERC-20 function `transfer`

erc20-transfer-revert-zero

Function `transfer` Prevents Transfers to the Zero Address.

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

Specification:

```
[](started(contract.transfer(to, value), to == address(0))
   ==> <>(reverted(contract.transfer) || finished(contract.transfer(to, value),
          !return)))
```

erc20-transfer-succeed-normal

Function `transfer` Succeeds on Admissible Non-self Transfers.

All invocations of the form `transfer(recipient, amount)` must succeed and return `true` if

- the `recipient` address is not the zero address,
- `amount` does not exceed the balance of address `msg.sender`,
- transferring `amount` to the `recipient` address does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call.

Specification:

```
[](started(contract.transfer(to, value), to != address(0)
         && to != msg.sender && value >= 0 && value <= _balances[msg.sender]
         && _balances[to] + value <= type(uint256).max && _balances[to] >= 0
         && _balances[msg.sender] <= type(uint256).max)
         ==> <>(finished(contract.transfer(to, value), return)))
```

erc20-transfer-succeed-self

Function `transfer` Succeeds on Admissible Self Transfers.

All self-transfers, i.e. invocations of the form `transfer(recipient, amount)` where the `recipient` address equals the address in `msg.sender` must succeed and return `true` if

- the value in `amount` does not exceed the balance of `msg.sender` and

- the supplied gas suffices to complete the call.

Specification:

```
[](started(contract.transfer(to, value), to != address(0)
          && to == msg.sender && value >= 0 && value <= _balances[msg.sender]
          && _balances[msg.sender] >= 0
          && _balances[msg.sender] <= type(uint256).max)
          ==> <>(finished(contract.transfer(to, value), return)))
```

erc20-transfer-correct-amount

Function `transfer` Transfers the Correct Amount in Non-self Transfers.

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:

```
[](willSucceed(contract.transfer(to, value), to != msg.sender
   && _balances[to] >= 0 && value >= 0
   && _balances[to] + value <= type(uint256).max
   && _balances[msg.sender] >= 0 && _balances[msg.sender] <= type(uint256).max)
      ==> <>(finished(contract.transfer(to, value), return
             ==> _balances[msg.sender] == old(_balances[msg.sender]) - value
                 && _balances[to] == old(_balances[to]) + value)))
```

erc20-transfer-correct-amount-self

Function `transfer` Transfers the Correct Amount in Self Transfers.

All non-reverting invocations of `transfer(recipient, amount)` that return `true` and where the `recipient` address equals `msg.sender` (i.e. self-transfers) must not change the balance of address `msg.sender` .

Specification:

```
[](willSucceed(contract.transfer(to, value), to == msg.sender
  && _balances[to] >= 0 && _balances[to] <= type(uint256).max)
     ==> <>(finished(contract.transfer(to, value), return
         ==> _balances[to] == old(_balances[to]))))
```

erc20-transfer-change-state

Function `transfer` Has No Unexpected State Changes.

All non-reverting invocations of `transfer(recipient, amount)` that return `true` must only modify the balance entries of the `msg.sender` and the `recipient` addresses.

Specification:

```
[](willSucceed(contract.transfer(to, value), p1 != msg.sender && p1 != to)
  ==> <>(finished(contract.transfer(to, value), return
      ==> (_totalSupply == old(_totalSupply) && _allowances == old(_allowances)
           && _balances[p1] == old(_balances[p1])  ))))
```

erc20-transfer-exceed-balance

Function `transfer` Fails if Requested Amount Exceeds Available Balance.

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

Specification:

```
[](started(contract.transfer(to, value), value > _balances[msg.sender]
   && _balances[msg.sender] >= 0 && value <= type(uint256).max)
   ==> <>(reverted(contract.transfer) || finished(contract.transfer(to, value),
          !return)))
```

erc20-transfer-recipient-overflow

Function `transfer` Prevents Overflows in the Recipient's Balance.

Any invocation of `transfer(recipient, amount)` must fail if it causes the balance of the `recipient` address to overflow.

Specification:

```
[](started(contract.transfer(to, value), to != msg.sender
   && _balances[to] + value > type(uint256).max
   && _balances[to] >= 0 && _balances[to] <= type(uint256).max
   && _balances[msg.sender] <= type(uint256).max
   && value > 0 && value <= _balances[msg.sender])
   ==> <>(reverted(contract.transfer) || finished(contract.transfer(to, value),
          !return) || finished(contract.transfer(to, value), _balances[to]
                      > old(_balances[to]) + value - type(uint256).max - 1)))
```

erc20-transfer-false

If Function `transfer` Returns `false` , the Contract State Has Not Been Changed.

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

Specification:

```
[](willSucceed(contract.transfer(to, value))
   ==> <>(finished(contract.transfer(to, value), !return]
   ==> (_balances == old(_balances) && _totalSupply == old(_totalSupply)
                                    && _allowances == old(_allowances)  ))))
```

erc20-transfer-never-return-false

Function `transfe` Never Returns `false`.

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

Specification:

```
[](!(finished(contract.transfer, !return)))
```

Properties for ERC-20 function `transferFrom`

erc20-transferfrom-revert-from-zero

Function `transferFrom` Fails for Transfers From the Zero Address.

All calls of the form `transferFrom(from, dest, amount)` where the `from` address is zero, must fail.

Specification:

```
[](started(contract.transferFrom(from, to, value), from == address(0))
   ==> <>(reverted(contract.transferFrom) || finished(contract.transferFrom,
          !return)))
```

erc20-transferfrom-revert-to-zero

Function `transferFrom` Fails for Transfers To the Zero Address.

All calls of the form `transferFrom(from, dest, amount)` where the `dest` address is zero, must fail.

Specification:

```
[](started(contract.transferFrom(from, to, value), to == address(0))
   ==> <>(reverted(contract.transferFrom) || finished(contract.transferFrom,
          !return)))
```

erc20-transferfrom-succeed-normal

Function `transferFrom` Succeeds on Admissible Non-self Transfers. All invocations of `transferFrom(from, dest, amount)` must succeed and return `true` if

- the value of `amount` does not exceed the balance of address `from`,

- the value of `amount` does not exceed the allowance of `msg.sender` for address `from`,
- transferring a value of `amount` to the address in `dest` does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call.

Specification:

```
[](started(contract.transferFrom(from, to, value), from != address(0)
   && to != address(0) && from != to && value <= _balances[from]
   && value <= _allowances[from][msg.sender]
   && _balances[to] + value <= type(uint256).max
   && value >= 0 && _balances[to] >= 0 && _balances[from] >= 0
   && _balances[from] <= type(uint256).max
   && _allowances[from][msg.sender] >= 0
   && _allowances[from][msg.sender] <= type(uint256).max)
      ==> <>(finished(contract.transferFrom(from, to, value), return)))
```

erc20-transferfrom-succeed-self

Function `transferFrom` Succeeds on Admissible Self Transfers.

All invocations of `transferFrom(from, dest, amount)` where the `dest` address equals the `from` address (i.e. self-transfers) must succeed and return `true` if:

- The value of `amount` does not exceed the balance of address `from`,
- the value of `amount` does not exceed the allowance of `msg.sender` for address `from`, and
- the supplied gas suffices to complete the call.

Specification:

```
[](started(contract.transferFrom(from, to, value), from != address(0)
   && from == to && value <= _balances[from]
   && value <= _allowances[from][msg.sender]
   && value >= 0 && _balances[from] <= type(uint256).max
   && _allowances[from][msg.sender] <= type(uint256).max)
     ==> <>(finished(contract.transferFrom(from, to, value), return)))
```

erc20-transferfrom-correct-amount

Function `transferFrom` Transfers the Correct Amount in Non-self Transfers.

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:

```
[](willSucceed(contract.transferFrom(from, to, value), from != to && value >= 0
&& _balances[from] >= 0 && _balances[from] <= type(uint256).max
&& _balances[to] >= 0 && _balances[to] + value <= type(uint256).max)
   ==> <>(finished(contract.transferFrom(from, to, value), return
          ==> _balances[from] == old(_balances[from]) - value
              && _balances[to] == old(_balances[to] + value))))
```

erc20-transferfrom-correct-amount-self

Function `transferFrom` Performs Self Transfers Correctly.

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` and where the address in `from` equals the address in `dest` (i.e. self-transfers) do not change the balance entry of the `from` address (which equals `dest`).

Specification:

```
[](willSucceed(contract.transferFrom(from, to, value), from == to
   && value >= 0 && value <= type(uint256).max && _balances[from] >= 0
   && _balances[from] <= type(uint256).max)
      ==> <>(finished(contract.transferFrom(from, to, value), return
             ==> _balances[from] == old(_balances[from]))))
```

erc20-transferfrom-correct-allowance

Function `transferFrom` Updated the Allowance Correctly.

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:

```
[](willSucceed(contract.transferFrom(from, to, value), value >= 0
   && value <= type(uint256).max && _balances[from] >= 0
   && _balances[from] <= type(uint256).max && _balances[to] >= 0
   && _balances[to] <= type(uint256).max && _allowances[from][msg.sender] >= 0
   && _allowances[from][msg.sender] <= type(uint256).max)
      ==> <>(finished(contract.transferFrom(from, to, value), return
          ==> ((_allowances[from][msg.sender]
                == old(_allowances[from][msg.sender]) - value)
               || (_allowances[from][msg.sender]
                   == old(_allowances[from][msg.sender])
                      && (from == msg.sender
                          || old(_allowances[from][msg.sender])
                             == type(uint256).max))))))
```

erc20-transferfrom-change-state

Function `transferFrom` Has No Unexpected State Changes.

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` may only modify the following state variables:

- The balance entry for the address in `dest`,
- The balance entry for the address in `from`,
- The allowance for the address in `msg.sender` for the address in `from`. Specification:

```
[](willSucceed(contract.transferFrom(from, to, amount), p1 != from && p1 != to
   && (p2 != from || p3 != msg.sender))
    ==> <>(finished(contract.transferFrom(from, to, amount), return
     ==> (_totalSupply == old(_totalSupply) && _balances[p1] == old(_balances[p1])
         && _allowances[p2][p3] == old(_allowances[p2][p3])  ))))
```

erc20-transferfrom-fail-exceed-balance

Function `transferFrom` Fails if the Requested Amount Exceeds the Available 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:

```
[](started(contract.transferFrom(from, to, value), value > _balances[from]
   && _balances[from] >= 0 && _balances[from] <= type(uint256).max)
       ==> <>(reverted(contract.transferFrom)
              || finished(contract.transferFrom, !return)))
```

erc20-transferfrom-fail-exceed-allowance

Function `transferFrom` Fails if the Requested Amount Exceeds the Available 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:

```
  [](started(contract.transferFrom(from, to, value), value > _allowances[from]
[msg.sender]
    && _allowances[from][msg.sender] >= 0 && value <= type(uint256).max)
    ==> <>(reverted(contract.transferFrom)
          || finished(contract.transferFrom(from, to, value), !return)
          || finished(contract.transferFrom(from, to, value), return
             && (msg.sender == from
                 || _allowances[from][msg.sender] == type(uint256).max))))
```

erc20-transferfrom-fail-recipient-overflow

Function `transferFrom` Prevents Overflows in the Recipient's Balance.

Any call of `transferFrom(from, dest, amount)` with a value in `amount` whose transfer would cause an overflow of the balance of address `dest` must fail.

Specification:

```
[](started(contract.transferFrom(from, to, value), from != to
   && _balances[to] + value > type(uint256).max && value <= type(uint256).max
   && _balances[to] >= 0 && _balances[to] <= type(uint256).max)
      ==> <>(reverted(contract.transferFrom)
          || finished(contract.transferFrom(from, to, value), !return)
          || finished(contract.transferFrom(from, to, value), _balances[to]
              > old(_balances[to]) + value - type(uint256).max - 1)))
```

erc20-transferfrom-false

If Function `transferFrom` Returns `false`, the Contract's State Has Not Been Changed.

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

Specification:

```
[](willSucceed(contract.transfer(to, value))
   ==> <>(finished(contract.transfer(to, value), !return
   ==> (_balances == old(_balances) && _totalSupply == old(_totalSupply)
                                    && _allowances == old(_allowances)  ))))
```

erc20-transferfrom-never-return-false

Function `transferFrom` Never Returns `false`.

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

Specification:

```
[](!(finished(contract.transferFrom, !return)))
```

Properties related to function `totalSupply`

erc20-totalsupply-succeed-always

Function `totalSupply` Always Succeeds.

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

Specification:

```
[](started(contract.totalSupply) ==> <>(finished(contract.totalSupply)))
```

erc20-totalsupply-correct-value

Function `totalSupply` Returns the Value of the Corresponding State Variable.

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

Specification:

```
[](willSucceed(contract.totalSupply)
   ==> <>(finished(contract.totalSupply, return == _totalSupply)))
```

erc20-totalsupply-change-state

Function `totalSupply` Does Not Change the Contract's State.

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

Specification:

```
[](willSucceed(contract.totalSupply)
   ==> <>(finished(contract.totalSupply, _totalSupply == old(_totalSupply)
          && _balances == old(_balances) && _allowances == old(_allowances)  )))
```

Properties related to function `balanceOf`

erc20-balanceof-succeed-always

Function `balanceOf` Always Succeeds.

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

Specification:

```
[](started(contract.balanceOf) ==> <>(finished(contract.balanceOf)))
```

erc20-balanceof-correct-value

Function `balanceOf` Returns the Correct Value.

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

Specification:

```
[](willSucceed(contract.balanceOf)
   ==> <>(finished(contract.balanceOf(owner), return == _balances[owner])))
```

erc20-balanceof-change-state

Function `balanceOf` Does Not Change the Contract's State.

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

Specification:

```
[](willSucceed(contract.balanceOf)
   ==> <>(finished(contract.balanceOf(owner), _totalSupply == old(_totalSupply)
          && _balances == old(_balances)
          && _allowances == old(_allowances)  )))
```

Properties related to function `allowance`

erc20-allowance-succeed-always

Function `allowance` Always Succeeds.

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

Specification:

```
[](started(contract.allowance) ==> <>(finished(contract.allowance)))
```

erc20-allowance-correct-value

Function `allowance` Returns Correct Value.

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

Specification:

```
[](willSucceed(contract.allowance(owner, spender))
   ==> <>(finished(contract.allowance(owner, spender),
           return == _allowances[owner][spender])))
```

erc20-allowance-change-state

Function `allowance` Does Not Change the Contract's State.

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

Specification:

```
[](willSucceed(contract.allowance(owner, spender))
   ==> <>(finished(contract.allowance(owner, spender),
   _totalSupply == old(_totalSupply) && _balances == old(_balances)
               && _allowances == old(_allowances)  )))
```

Properties related to function `approve`

erc20-approve-revert-zero

Function `approve` Prevents Giving Approvals For the Zero Address.

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

Specification:

```
[](started(contract.approve(spender, value), spender == address(0))
   ==> <>(reverted(contract.approve)
          || finished(contract.approve(spender, value), !return)))
```

erc20-approve-succeed-normal

Function `approve` Succeeds for Admissible Inputs.

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:

```
[](started(contract.approve(spender, value), spender != address(0))
   ==> <>(finished(contract.approve(spender, value), return)))
```

erc20-approve-correct-amount

Function `approve` Updates the Approval Mapping Correctly.

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:

```
[](willSucceed(contract.approve(spender, value), spender != address(0)
   && value >= 0 && value <= type(uint256).max)
       ==> <>(finished(contract.approve(spender, value), return
              ==> _allowances[msg.sender][spender] == value)))
```

erc20-approve-change-state

Function `approve` Has No Unexpected State Changes.

All calls of the form `approve(spender, amount)` must only update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount` and incur no other state changes.

Specification:

```
[](willSucceed(contract.approve(spender, value), spender != address(0)
   && (p1 != msg.sender || p2 != spender))
       ==> <>(finished(contract.approve(spender, value), return
              ==> _totalSupply == old(_totalSupply) && _balances == old(_balances)
                  && _allowances[p1][p2] == old(_allowances[p1][p2])  )))
```

erc20-approve-false

If Function `approve` Returns `false`, the Contract's State Has Not Been Changed.

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

Specification:

```
[](willSucceed(contract.approve(spender, value))
   ==> <>(finished(contract.approve(spender, value), !return
          ==> (_balances == old(_balances) && _totalSupply == old(_totalSupply)
               && _allowances == old(_allowances)  ))))
```

erc20-approve-never-return-false

Function `approve` Never Returns `false`.

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

Specification:

```
 [](!(finished(contract.approve, !return)))
```

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.

CertiK | Securing the Web3 World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading 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.