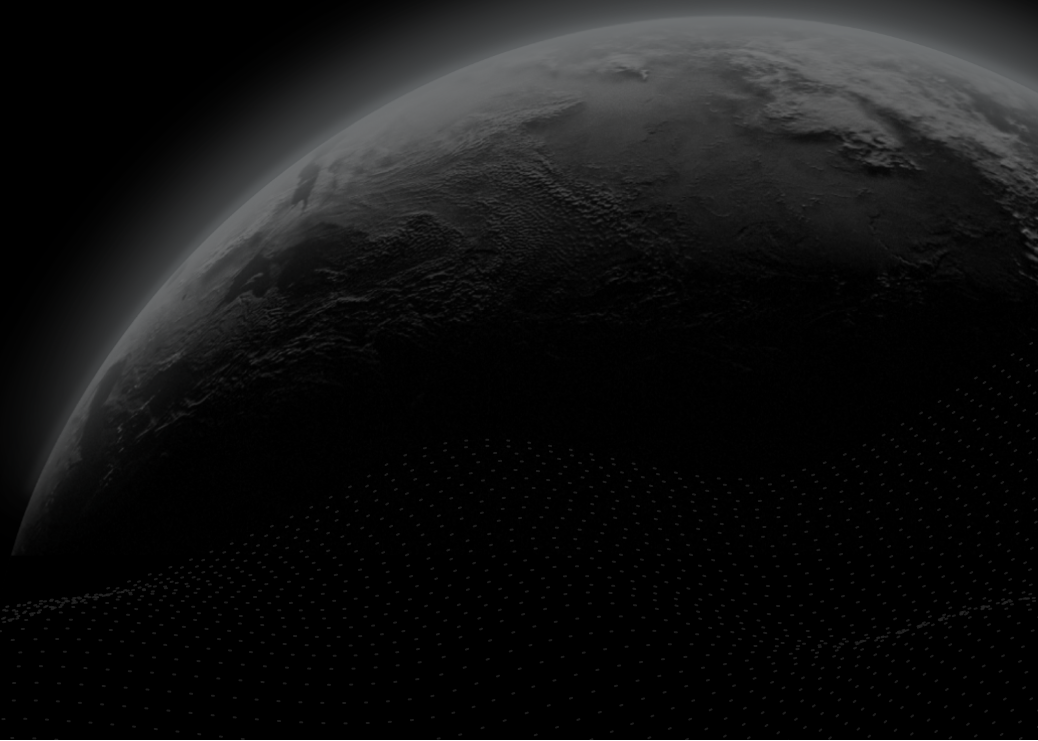
CERTIK

Security Assessment

Momentum

CertiK Assessed on Jul 4th, 2023



CertiK Assessed on Jul 4th, 2023

Momentum

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 07/04/2023	N/A

CODEBASE

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

View All in Codebase Page

COMMITS

- 938a1bdb7d02c98cc2afd097cb0d9981eb58499d
- 6fc42dd5ee8538436f14146d50912feb627edc98
- abc630afff3c58b2fb698612d47fa8d0b545c78d

View All in Codebase Page

Vulnerability Summary

23	19	0	0	4	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.
7	Major	4 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.
8	Medium	7 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	5 Resolved	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

CODEBASE | MOMENTUM

Repository

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

Commit

- 938a1bdb7d02c98cc2afd097cb0d9981eb58499d
- 6fc42dd5ee8538436f14146d50912feb627edc98
- abc630afff3c58b2fb698612d47fa8d0b545c78d

AUDIT SCOPE | MOMENTUM

8 files audited ● 5 files with Acknowledged findings ● 3 files without findings

ID	Repo	File	SHA256 Checksum
ONF	momentum-xyz/contracts	contracts/nft/OdysseyNFT.sol	917a47b2749d1d8cbfb9735355c607216af9934774c61830c7ec636b1ac5ab1b
STK	momentum-xyz/contracts	contracts/staking/Staking.sol	acc092bfa38cabf3bbce059a40fbd393ecffb377a32ce5dfd41e504b69fcd6e3
DAD	momentum-xyz/contracts	contracts/token/DADToken.sol	75670a5f5d56af01da541fbf95fe040cea0c67bf829ac39cc0da118cf79b5fe8
MTB	momentum-xyz/contracts	contracts/token/MomToken.sol	bd40ae7fec6c86f1313741e7d5f2af0a569eeb370e1a957179aed37e60c23f08
STI	momentum-xyz/contracts	contracts/staking/Staking.sol	867818535183f5e7cff2ac336796bf3c73447fbbaf8791fdfc5c0774dc538a47
ONT	momentum-xyz/contracts	contracts/nft/OdysseyNFT.sol	763896624b6a53c5061b666c5b91ce31e0e0525ab951b6ec31574e0071fd16c6
DAT	momentum-xyz/contracts	contracts/token/DADToken.sol	75670a5f5d56af01da541fbf95fe040cea0c67bf829ac39cc0da118cf79b5fe8
MTU	momentum-xyz/contracts	contracts/token/MomToken.sol	bd40ae7fec6c86f1313741e7d5f2af0a569eeb370e1a957179aed37e60c23f08

APPROACH & METHODS | MOMENTUM

This report has been prepared for Momentum to discover issues and vulnerabilities in the source code of the Momentum 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

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`. It has a maximum number of tokens and each holder can have a limited number of Odyssey NFT (both numbers could be reset by the owner). Users can stake/unstake tokens (`MOM` or `DAD` discussed below) on Odyssey NFT.
- **MOMToken**: The `MOMToken` contract is an `ERC20` token with the symbol `MOM`. It can be used as staked token by users to stake on Odyssey NFT to earn rewards. In addition, `MOM` is also the reward token that is minted to users or the owner of Odyssey NFT. It is worth noting that the any account with `MINTER_ROLE` role is authorized to mint any number of `MOM` to any account.
- **DADToken**: The `DADToken` contract is an `ERC20` token with the symbol `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 the `TRANSFER_ROLE` role granted. It is worth noting that the any accounts with `DEFAULT_ADMIN_ROLE` role is authorized to mint any number of `DAD` to any account.
- **Staking**: The `Staking` is the central contract of `Momentum` system and includes functions like staking, unstaking, updating rewards, and claiming rewards. Users can stake either `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 included in this contract. Once the rewards are updated, users can claim their rewards. The owner of Odyssey NFT can also receive rewards if the Odyssey NFT has been staked into. All these rewards are `MOM` tokens that are minted to users. In addition, all the staked tokens kept in the `Staking` contract can be withdrawn by uses when they unstake and then claim the unstaked tokens.

External Dependencies

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

- rewards calculated by third party: the logic to calculate the rewards is a black-box
- The addresses `mom_token`, `dad_token`, `odyssey_nfts`

It is assumed that the rewards calculation is 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-01 | 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

23	0	7	8	5	3
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Momentum. Through this audit, we have uncovered 23 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-01	**Centralization Related Risks**	**Centralization**	**Major**	**Acknowledged**
MTB-01	Initial Token Distribution	Centralization / Privilege	Major	Acknowledged
STI-01	User Cannot Stake After Unstaking Due To Index Out Of Bounds	Coding Issue	Major	Resolved
STI-02	Calculated Rewards Could Be Removed Out Of Stakers After Unstaking	Logical Issue	Major	Resolved
STK-01	**Centralized Control Of Contract Upgrade**	**Centralization**	**Major**	**Acknowledged**
STK-14	Double Counting When Unstaking	Logical Issue	Major	Resolved
STK-16	Faulty Reset Of User Tokens	Logical Issue	Major	Resolved
STI-03	`MOM` Or `DAD` Amount Of Staker Increased After Restaking	Coding Issue	Medium	Resolved
STI-04	Possible For Stakes To Be Locked In The Contract	Design Issue	Medium	Acknowledged
STK-03	Lack Of Storage Gap In Upgradeable Contract	Logical Issue	Medium	Resolved
STK-05	Incorrect Use Of `addresses` Length In Loop And Lack Of Length Check	Logical Issue	Medium	Resolved

ID	Title	Category	Severity	Status
STK-06	Potential Multiple Counting On `odyssey.total_stakers`	Logical Issue	Medium	Resolved
STK-17	Inaccurate Rise In The Total Staked Amount Of The User.	Logical Issue	Medium	Resolved
STK-18	`odyssey.total_staked_into` May Not Be Decreased	Logical Issue	Medium	Resolved
STK-19	User Potentially Cannot Unstake Caused By Underflow	Logical Issue	Medium	Resolved
STK-02	Potential Damage Due To Unprotected Initializer	Logical Issue	Minor	Resolved
STK-07	Missing Zero Address Validation	Volatile Code	Minor	Resolved
STK-08	Unchecked ERC-20 `transfer()` / `transferFrom()` Call	Volatile Code	Minor	Resolved
STK-15	No Check To Ensure `odyssey` Exists	Logical Issue	Minor	Resolved
STK-20	Potential Risks Associated With Changing Staking Tokens	Logical Issue	Minor	Resolved
CON-02	Missing Emit Events	Coding Style	Informational	Resolved
STK-09	Missing Error Messages	Coding Style	Informational	Resolved
STK-13	Lack Of Usage Of `effective_timestamp`	Logical Issue	Informational	Resolved

CON-01 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization	Major	contracts/nft/OdysseyNFT.sol (938a1bdb7d02c98cc2afd097cb0d9981eb58499d): 72, 79, 87, 103, 127, 151; contracts/staking/Staking.sol (938a1bdb7d02c98cc2afd097cb0d9981eb58499d): 225, 235, 243, 251, 263, 288, 296; contracts/token/DADToken.sol (938a1bdb7d02c98cc2afd097cb0d9981eb58499d): 35, 40, 50, 67, 76, 82, 91; contracts/token/MomToken.sol (938a1bdb7d02c98cc2afd097cb0d9981eb58499d): 46, 54, 64, 82, 92	Acknowledged

Description

In the contract `MOMToken`, the role `PAUSER_ROLE` has authority over the functions shown below:

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

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

In the contract `MOMToken`, the role `MINTER_ROLE` has authority over the functions shown below:

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

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

In the contract `MOMToken`, the role `BURNER_ROLE` has authority over the functions shown below:

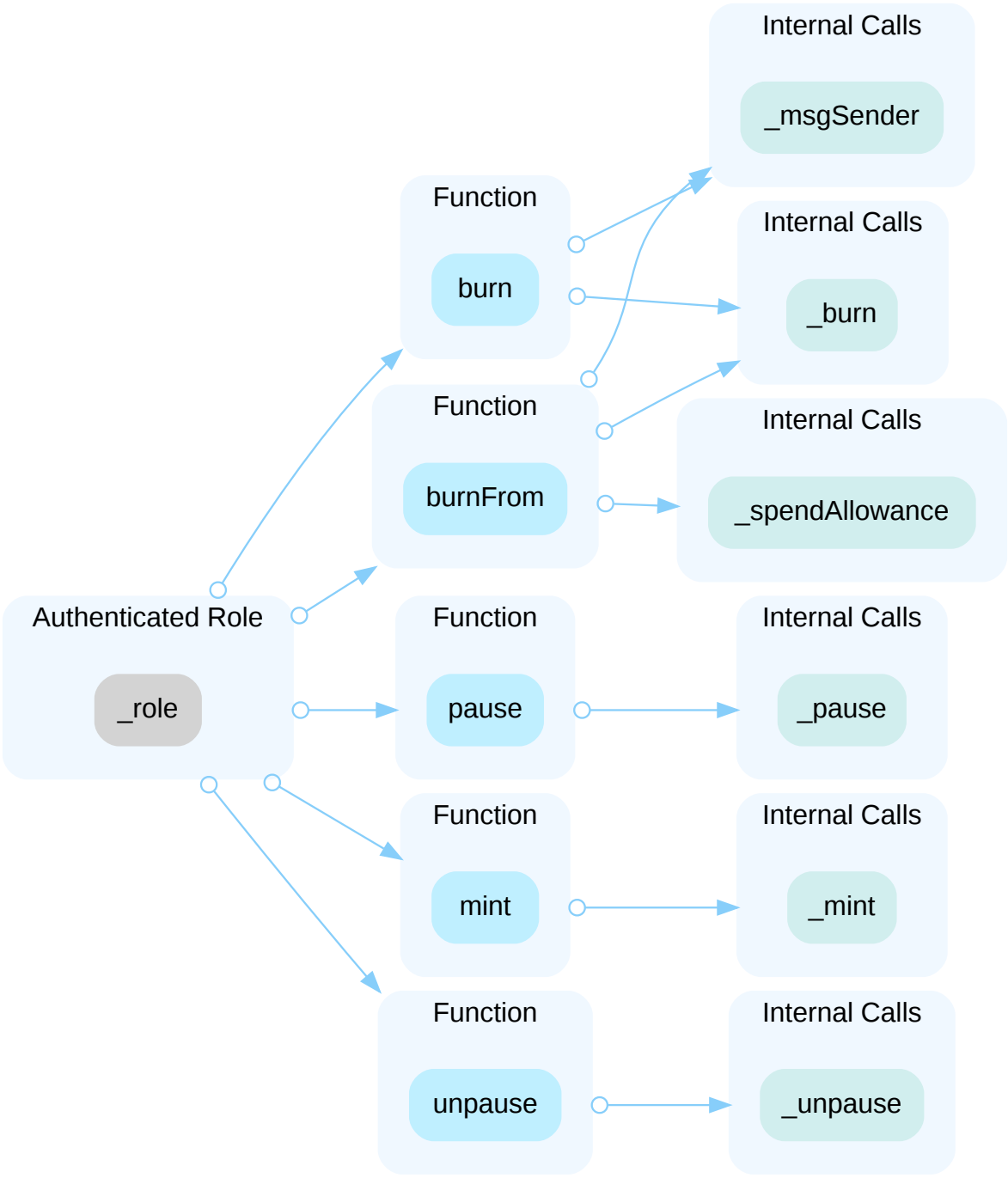
- function burn(): Accounts assigned the `BURNER_ROLE` can burn their own `MOM` tokens.
- function burnFrom(): Accounts that have been assigned the `BURNER_ROLE` can use their allowance to burn `MOM` tokens belonging to other accounts.

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

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 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 randomly.

In the contract `DADToken`, the role `TRANSFER_ROLE` has authority over the functions shown 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 randomly.

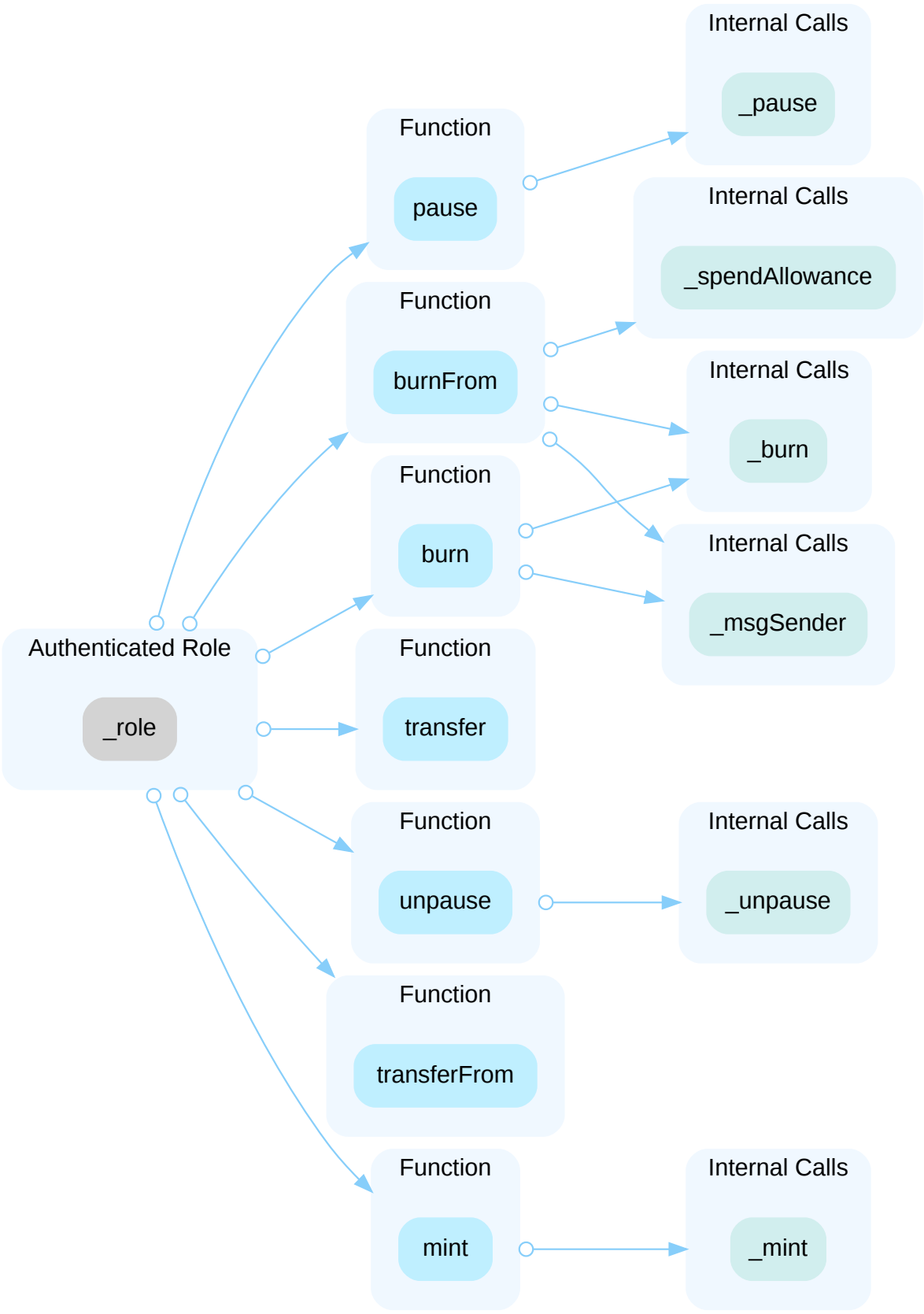
In the contract `DADToken`, the role `DEFAULT_ADMIN_ROLE` inherited from `AccessControl` contract has authority over the functions shown below:

- function pause(): Accounts assigned the `DEFAULT_ADMIN_ROLE` can pause the contract, which will prevent the transfer of `DAD` tokens.
- function unpause(): Accounts assigned the `DEFAULT_ADMIN_ROLE` can unpause the contract, which will allow the transfer of `DAD` tokens.
- function mint(): Accounts assigned the `DEFAULT_ADMIN_ROLE` can mint arbitrary amounts of `DAD` tokens to any account.

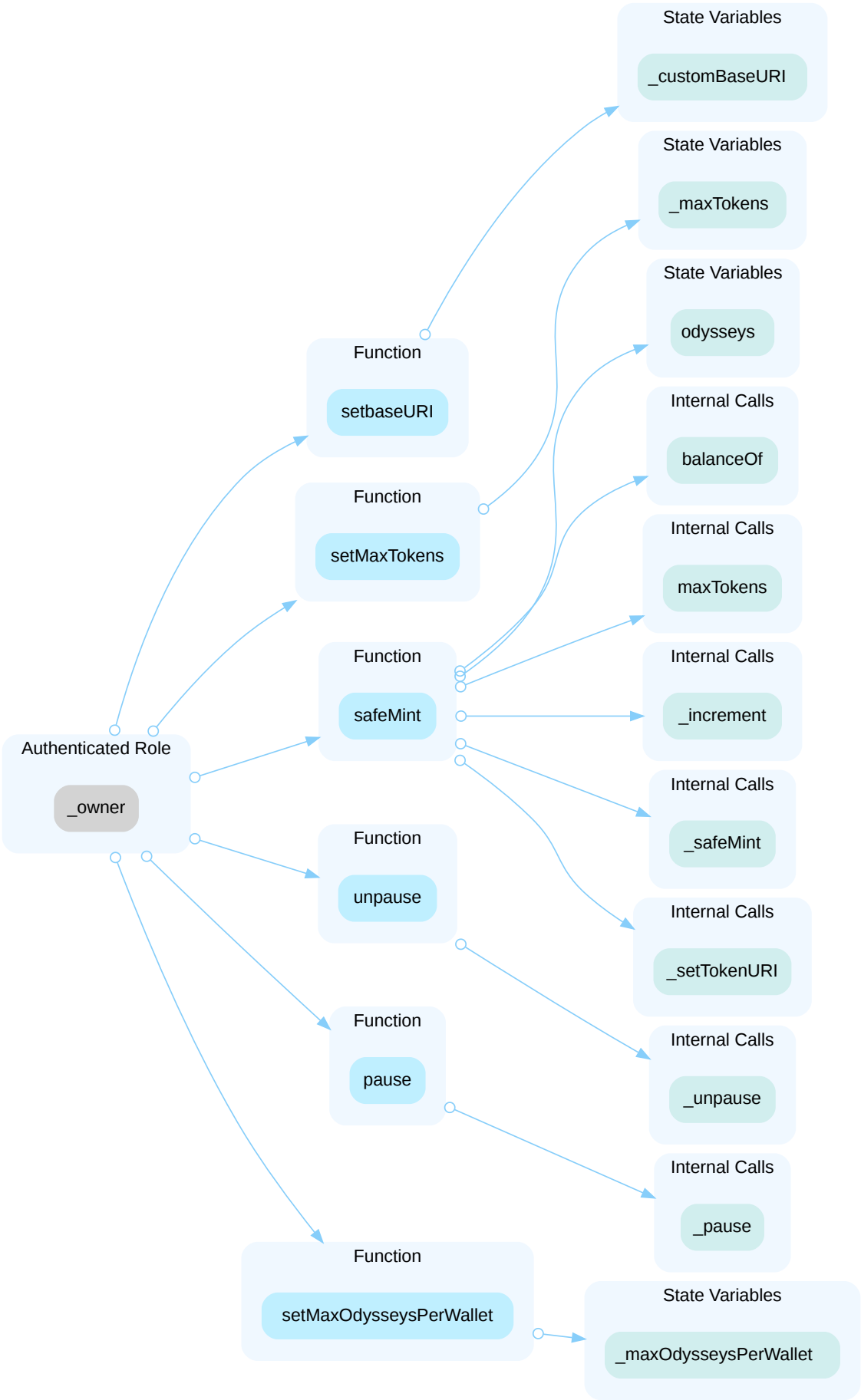
Accounts granted with `DEFAULT_ADMIN_ROLE` can grant or revoke other roles. By default, the deployer of `DADToken` contract has been granted all these roles.

- DEFAULT_ADMIN_ROLE
- TRANSFER_ROLE
- BURNER_ROLE

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.


State Variables
_customBaseURI
State Variables
_maxTokens
State Variables
odysseys
Function
setbaseURI
Internal Calls
balanceOf
Function
setMaxTokens
Internal Calls
maxTokens
Function
safeMint
Internal Calls
_increment
Authenticated Role
_owner
Internal Calls
_safeMint
Function
unpause
Internal Calls
_setTokenURI
Function
pause
Internal Calls
_unpause
Internal Calls
_pause
Function
setMaxOdysseysPerWallet
State Variables
_maxOdysseysPerWallet

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 `setMaxTokens()` to set an unexpected maximum number as `_maxTokens`;
- execute function `setMaxOdysseysPerWallet()` to set an unexpected maximum number as `_maxOdysseysPerWallet`;
- 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 below:

- function update_mom_token_contract(): Accounts assigned the `MANAGER_ROLE` can update the address of `mom_token`.
- function update_dad_token_contract(): Accounts assigned the `MANAGER_ROLE` can update the address of `dad_token`.
- function update_odyssey_nfts_contract(): Accounts assigned the `MANAGER_ROLE` can update the address of `odyssey_nfts`.
- function update_rewards(): Accounts assigned the `MANAGER_ROLE` can update rewards for users who staked in Odyssey and the owner of Odyssey NFT.
- 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`.

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

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

- function _authorizeUpgrade(): Accounts assigned the `DEFAULT_ADMIN_ROLE` can upgrade this contract.

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.

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 not have 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]: In the remediation commit 6fc42dd5ee8538436f14146d50912feb627edc98, the following privileged functions were removed in `Staking` contract :

- function update_mom_token_contract(): Accounts assigned the `MANAGER_ROLE` can update the address of `mom_token` .
- function update_dad_token_contract(): Accounts assigned the `MANAGER_ROLE` can update the address of `dad_token` .
- function update_odyssey_nfts_contract(): Accounts assigned the `MANAGER_ROLE` can update the address of `odyssey_nfts` .

MTB-01 | INITIAL TOKEN DISTRIBUTION

Category	Severity	Location	Status
Centralization / Privilege	Major	contracts/token/MomToken.sol (938a1bdb7d02c98cc2afd097cb0d9981eb58499d): 38~39	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.

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.

STI-01 USER CANNOT STAKE AFTER UNSTAKING DUE TO INDEX OUT OF BOUNDS

Category	Severity	Location	Status
Coding Issue	Major	contracts/staking/Staking.sol (update_0608): 414, 433~438	Resolved

Description

An `Index out of bounds` error may prevent users from staking tokens after they have unstaked.

```
433             staked_by[odyssey_id][index].effective_timestamp =
434                     calculate_effective_timestamp(staked_by[odyssey_id][index].
timestamp,
435                         staked_by[odyssey_id][index].total_amount,
436                         amount,
437                         true);
438             staked_by[odyssey_id][index].total_amount += amount;
```

The issue stems from the fact that, during the unstaking process, the `remove_staked_by()` function does not reset the index stored in `staked_by_indexes` to zero when the staker is removed from the `staked_by` mapping.

```
637     function remove_staked_by(uint256 odyssey_id, address staker) private {
638         if(staked_by[odyssey_id].length == 2) {
639             delete staked_by[odyssey_id];
640             staked_by_indexes[odyssey_id][staker] = 0;
641             return;
642         }
643         StakedBy storage last_item = staked_by[odyssey_id][staked_by[odyssey_id
].length-1];
644         staked_by_indexes[odyssey_id][last_item.user] = staked_by_indexes[
odyssey_id][staker];
645         staked_by[odyssey_id][staked_by_indexes[odyssey_id][staker]] =
 last_item;
646         staked_by[odyssey_id].pop();
647     }
```

If the same user stakes in the same Odyssey again, the index obtained from `staked_by_indexes` will be the old value, which may exceed the length of the `staked_by[odyssey_id]` array.

```
414         uint256 index = staked_by_indexes[odyssey_id][msg.sender];
415         // First stake of the user
416         if (staked_by[odyssey_id].length == 0) {
417             staked_by[odyssey_id].push();
418         }
419         // The user is not staking on the odyssey
420         if(index == 0) {
```

Scenario

- Bob stakes 5000000000 MOM tokens in Odyssey#1
- Bob stakes 5000000000 DAD tokens in Odyssey#1
- Tom stakes 5000000000 MOM tokens in Odyssey#1
- Tom stakes 5000000000 MOM tokens in Odyssey#2
- Bob unstakes 5000000000 MOM tokens in Odyssey#1
- Tom unstakes 5000000000 MOM tokens in Odyssey#1
- Tom unstakes 5000000000 MOM tokens in Odyssey#2
- Tom stakes 5000000000 MOM tokens in Odyssey#1 but failed

Proof of Concept

The following proof of concept is using Foundry to test the case user cannot stake tokens after unstaking in the same Odyssey caused by `Index out of bounds` error.

```
// 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 StakingTestNew is Test {

    Staking public staking;
    MOMToken public mToken;
    DADToken public dToken;
    OdysseyNFT public odysseyNFT;

    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[] addresses = [Bob, Tom];
    uint256[] stakers_amounts = [1e10, 1e10];
    uint256[] odysseys_ids;
    uint256[] odysseys_amounts = [1e10, 1e10];

    function setUp() public {
        mToken = new MOMToken(1e20);
        dToken = new DADToken();
        staking = new Staking();
        odysseyNFT = new OdysseyNFT(
            "Odyssey_NFT",
            "ODS",
            21000,
            150,
            "ipfs://odyssey.nft"
        );

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

        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);
        vm.stopPrank();

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

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

        odysseyNFT.safeMint(Owner_Odyssey_1);
        odyssey_1 = odysseyNFT.currentId();
        odysseyNFT.safeMint(Owner_Odyssey_2);
        odyssey_2 = odysseyNFT.currentId();
        odysseys_ids = [odyssey_1, odyssey_2];

    }

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

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

    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.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
        );
    }

    function printStakedBy(uint256 odyssey_id) private view {
        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--------------------",  stakedBy.user, odyssey_id);
            console2.log("user= %s, total_amount= %d, dad_amount= %d,",
                stakedBy.user, stakedBy.total_amount, stakedBy.dad_amount
            );
            console2.log("mom_amount= %d, timestamp= %d, effective_timestamp= %d",
                stakedBy.mom_amount, stakedBy.timestamp,
stakedBy.effective_timestamp
            );
        }

    }

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

    function printStatistic() private view {
        console2.log("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Print Info
Start~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
        printTotalStaked();
        printStakedOdysseys();
        for (uint i = 0; i < addresses.length; i++) {
            printStakerFor(addresses[i]);
        }
        for (uint i = 0; i < odysseys_ids.length; i++) {
            printOdysseyFor(odysseys_ids[i]);
            printStakedBy(odysseys_ids[i]);
        }
        console2.log("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Print Info
End~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
    }

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

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

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

        printStatistic();

        vm.warp(block.timestamp + 10 days);
        console2.log("Bob unstakes 5000000000 MOM tokens in Odyssey#1");
        vm.prank(Bob);
        staking.unstake(odyssey_1, Staking.Token.MOM);

        uint staked_by_index_Bob;
        uint staked_by_index_Tom;

        staked_by_index_Bob = staking.staked_by_indexes(odyssey_1, Bob);
        staked_by_index_Tom = staking.staked_by_indexes(odyssey_1, Tom);
        console2.log("%d, staked_by_index_Bob = %d, staked_by_index_Tom = %d",
odyssey_1, staked_by_index_Bob, staked_by_index_Tom);
        staked_by_index_Bob = staking.staked_by_indexes(odyssey_2, Bob);
        staked_by_index_Tom = staking.staked_by_indexes(odyssey_2, Tom);
        console2.log("%d, staked_by_index_Bob = %d, staked_by_index_Tom = %d",
odyssey_2, staked_by_index_Bob, staked_by_index_Tom);

        vm.warp(block.timestamp + 12 days);
        console2.log("Tom unstakes 5000000000 MOM tokens in Odyssey#1");
        vm.prank(Tom);
        staking.unstake(odyssey_1, Staking.Token.MOM);

        vm.warp(block.timestamp + 12 days);
        console2.log("Tom unstakes 5000000000 MOM tokens in Odyssey#2");
        vm.prank(Tom);
        staking.unstake(odyssey_2, Staking.Token.MOM);

        staked_by_index_Bob = staking.staked_by_indexes(odyssey_1, Bob);
        staked_by_index_Tom = staking.staked_by_indexes(odyssey_1, Tom);
        console2.log("%d, staked_by_index_Bob = %d, staked_by_index_Tom = %d",
odyssey_1, staked_by_index_Bob, staked_by_index_Tom);
        staked_by_index_Bob = staking.staked_by_indexes(odyssey_2, Bob);
        staked_by_index_Tom = staking.staked_by_indexes(odyssey_2, Tom);
        console2.log("%d, staked_by_index_Bob = %d, staked_by_index_Tom = %d",
odyssey_2, staked_by_index_Bob, staked_by_index_Tom);
```

```
        printStatistic();

        console2.log("Tom stakes 5000000000 MOM tokens in Odyssey#1 and needs to
revert");
        vm.expectRevert();
        vm.prank(Tom);
        staking.stake(odyssey_1, 5e9, Staking.Token.MOM);
    }
}
```

Execution command:

```
forge test --mc StakingTestNew --mt
testUserCannotStakeAfterUnstakingDueToIndexOutOfBounds -vvvv
```

The output is:

```
[⠊] Compiling...
[⠒] Compiling 1 files with 0.8.20
[⠢] Solc 0.8.20 finished in 3.69s
Compiler run successful!

Running 1 test for test/StakingTest.t.sol:StakingTestNew
[PASS] testUserCannotStakeAfterUnstakingDueToIndexOutOfBounds() (gas: 1196849)
Logs:
  Bob stakes 5000000000 MOM tokens in Odyssey#1
  Bob stakes 5000000000 DAD tokens in Odyssey#1
  Tom stakes 5000000000 MOM tokens in Odyssey#1
  Tom stakes 5000000000 MOM tokens in Odyssey#2
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Print Info Start~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  Currently total staked is 20000000000
  Length of `staked_odysseys`= 2
  --------------------Staker info for user :
0x0000000000000000000000000000000000000123--------------------
  user=0x0000000000000000000000000000000000000123, total_rewards=0,
total_staked=10000000000,
  dad_amount=5000000000, mom_amount=5000000000
  --------------------Staker info for user :
0x0000000000000000000000000000000000000456--------------------
  user=0x0000000000000000000000000000000000000456, total_rewards=0,
total_staked=10000000000,
  dad_amount=0, mom_amount=10000000000
  --------------------Odyssey info for odyssey_id : 604472133179351442128897---------
----------
  odyssey_id= 604472133179351442128897, total_staked_into= 15000000000,
total_stakers= 2,
  total_rewards= 0, staked_odysseys_index= 0
  --------------------StakedBy info for user:
0x0000000000000000000000000000000000000123 in odyssey_id : 604472133179351442128897-
------------------
  user= 0x0000000000000000000000000000000000000123, total_amount= 10000000000,
dad_amount= 5000000000,
  mom_amount= 5000000000, timestamp= 259201, effective_timestamp= 129601
  --------------------StakedBy info for user:
0x0000000000000000000000000000000000000456 in odyssey_id : 604472133179351442128897-
------------------
  user= 0x0000000000000000000000000000000000000456, total_amount= 5000000000,
dad_amount= 0,
  mom_amount= 5000000000, timestamp= 259201, effective_timestamp= 259201
  --------------------Odyssey info for odyssey_id : 604472133179351442128898---------
----------
  odyssey_id= 604472133179351442128898, total_staked_into= 5000000000,
total_stakers= 1,
  total_rewards= 0, staked_odysseys_index= 1
  --------------------StakedBy info for user:
0x0000000000000000000000000000000000000456 in odyssey_id : 604472133179351442128898-
------------------
```

```
  user= 0x0000000000000000000000000000000000000456, total_amount= 5000000000,
dad_amount= 0,
  mom_amount= 5000000000, timestamp= 259201, effective_timestamp= 259201
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Print Info End~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  Bob unstakes 5000000000 MOM tokens in Odyssey#1
  604472133179351442128897, staked_by_index_Bob = 1, staked_by_index_Tom = 2
  604472133179351442128898, staked_by_index_Bob = 0, staked_by_index_Tom = 1
  Tom unstakes 5000000000 MOM tokens in Odyssey#1
  Tom unstakes 5000000000 MOM tokens in Odyssey#2
  604472133179351442128897, staked_by_index_Bob = 1, staked_by_index_Tom = 2
  604472133179351442128898, staked_by_index_Bob = 0, staked_by_index_Tom = 0
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Print Info Start~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  Currently total staked is 5000000000
  Length of `staked_odysseys`= 1
  --------------------Staker info for user :
0x0000000000000000000000000000000000000123--------------------
  user=0x0000000000000000000000000000000000000123, total_rewards=0,
total_staked=5000000000,
  dad_amount=5000000000, mom_amount=0
  --------------------Staker info for user :
0x0000000000000000000000000000000000000456--------------------
  user=0x0000000000000000000000000000000000000000, total_rewards=0, total_staked=0,
  dad_amount=0, mom_amount=0
  --------------------Odyssey info for odyssey_id : 604472133179351442128897---------
----------
  odyssey_id= 604472133179351442128897, total_staked_into= 5000000000,
total_stakers= 1,
  total_rewards= 0, staked_odysseys_index= 0
  --------------------StakedBy info for user:
0x0000000000000000000000000000000000000123 in odyssey_id : 604472133179351442128897-
------------------
  user= 0x0000000000000000000000000000000000000123, total_amount= 5000000000,
dad_amount= 5000000000,
  mom_amount= 0, timestamp= 1123201, effective_timestamp= 604799
  --------------------Odyssey info for odyssey_id : 604472133179351442128898---------
----------
  odyssey_id= 0, total_staked_into= 0, total_stakers= 0,
  total_rewards= 0, staked_odysseys_index= 0
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Print Info End~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  Tom stakes 5000000000 MOM tokens in Odyssey#1 and needs to revert

Traces:
...
    ├─ [71680] Staking::stake(604472133179351442128897 [6.044e23], 5000000000 [5e9],
0)
    │   ├─ [577] OdysseyNFT::exists(604472133179351442128897 [6.044e23])
[staticcall]
    │   │   └─ ← true
    │   ├─ [6045] MOMToken::transferFrom(0x0000000000000000000000000000000000000456,
Staking: [0xF62849F9A0B5Bf2913b396098F7c7019b51A820a], 5000000000 [5e9])
```

```
    |   |   ├─ emit Approval(owner: 0x0000000000000000000000000000000000000456,
spender: Staking: [0xF62849F9A0B5Bf2913b396098F7c7019b51A820a], value:
19999999985000000000 [1.999e20])
    |   |   ├─ emit Transfer(from: 0x0000000000000000000000000000000000000456, to:
Staking: [0xF62849F9A0B5Bf2913b396098F7c7019b51A820a], value: 5000000000 [5e9])
    |   |   └─ ← 0x0000000000000000000000000000000000000000000000000000000000000001
    |   └─ ← "Index out of bounds"
    └─ ← ()

Test result: ok. 1 passed; 0 failed; finished in 4.92ms
```

Recommendation

We recommend to refactor the logic in function `remove_staked_by()` to ensure users can stake tokens after unstaking. The possible solution is as below:

```
637     function remove_staked_by(uint256 odyssey_id, address staker) private {
638         if(staked_by[odyssey_id].length == 2) {
639             delete staked_by[odyssey_id];
640             staked_by_indexes[odyssey_id][staker] = 0;
641             return;
642         }
643
       StakedBy storage last_item = staked_by[odyssey_id]
[staked_by[odyssey_id].length-1];
644
       staked_by_indexes[odyssey_id][last_item.user] =
staked_by_indexes[odyssey_id][staker];
645
       staked_by[odyssey_id][staked_by_indexes[odyssey_id][staker]] = last_item;
646         staked_by[odyssey_id].pop();
647         staked_by_indexes[odyssey_id][staker]=0; //added to fix
648     }
```

Alleviation

[Momentum Team, 06/15/2023]: The team heeded the advice to fix the issue in the `Staking` contract and changes were included in commit 79015fb53e58603e6f0137a6e44ded6f39e282d4.

STI-02 CALCULATED REWARDS COULD BE REMOVED OUT OF STAKERS AFTER UNSTAKING

Category	Severity	Location	Status
Logical Issue	Major	contracts/staking/Staking.sol (update_0608): 493	Resolved

Description

In the `_unstake()` function of `Staking` contract, a staker's records info, including staked amount and reward, could be deleted out of storage once they have no tokens staked.

```
492         } else {
493             delete stakers[msg.sender];
494             remove_staked_by(odyssey_id, msg.sender);
495             decrease_odyssey_total_stakers(odyssey_id, amount);
496         }
```

If a staker forgets to claim the rewards before unstaking tokens, the staker may not be able to claim the reward later as the calculated reward for this staker has been removed during the unstaking process.

Scenario

- Bob stakes 5000000000 MOM tokens in Odyssey#1
- Bob stakes 5000000000 DAD tokens in Odyssey#1
- Tom stakes 5000000000 MOM tokens in Odyssey#1
- Tom stakes 5000000000 MOM tokens in Odyssey#2
- Manager updates rewards
- Bob retakes 5000000000 MOM from Odyssey#1 to Odyssey#2
- Tom retakes 5000000000 MOM from Odyssey#2 to Odyssey#1
- Bob unstakes 5000000000 DAD tokens in Odyssey#1
- Bob unstakes 5000000000 MOM tokens in Odyssey#2
- Tom unstakes 10000000000 MOM tokens in Odyssey#1
- Bob claims unstaked tokens
- Tom claims unstaked tokens
- Bob claims rewards and reverts
- Tom claims rewards and reverts

Proof of Concept

The following proof of concept is using Foundry to test the case user cannot claim rewards after unstaking.

```
// 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 StakingTestNew is Test {

    Staking public staking;
    MOMToken public mToken;
    DADToken public dToken;
    OdysseyNFT public odysseyNFT;

    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[] addresses = [Bob, Tom];
    uint256[] odysseys_ids;
    uint256[] stakers_amounts = [1e10, 1e10];
    uint256[] odysseys_amounts = [1e10, 1e10];


    function setUp() public {
        mToken = new MOMToken(1e20);
        dToken = new DADToken();
        staking = new Staking();
        odysseyNFT = new OdysseyNFT(
            "Odyssey_NFT",
            "ODS",
            21000,
            150,
            "ipfs://odyssey.nft"
        );

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

        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);
        vm.stopPrank();

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

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

        odysseyNFT.safeMint(Owner_Odyssey_1);
        odyssey_1 = odysseyNFT.currentId();
        odysseyNFT.safeMint(Owner_Odyssey_2);
        odyssey_2 = odysseyNFT.currentId();
        odysseys_ids = [odyssey_1, odyssey_2];

    }

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

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

    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.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
        );
    }

    function printStakedBy(uint256 odyssey_id) private view {
        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-------------------",  stakedBy.user, odyssey_id);
            console2.log("user= %s, total_amount= %d, dad_amount= %d,",
                stakedBy.user, stakedBy.total_amount, stakedBy.dad_amount
            );
            console2.log("mom_amount= %d, timestamp= %d, effective_timestamp= %d",
                stakedBy.mom_amount, stakedBy.timestamp,
stakedBy.effective_timestamp
            );
        }

    }

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

    function printStatistic() private view {
        console2.log("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Print Info
Start~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
        printTotalStaked();
        printStakedOdysseys();
        for (uint i = 0; i < addresses.length; i++) {
            printStakerFor(addresses[i]);
        }
        for (uint i = 0; i < odysseys_ids.length; i++) {
            printOdysseyFor(odysseys_ids[i]);
            printStakedBy(odysseys_ids[i]);
        }
        printBalances();
        console2.log("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Print Info
End~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
    }

    function printBalances() private view {
        console2.log("-------------------Show Balances--------------------");
```

```
        console2.log("Bob's   balances: MOM = %d, DAD = %d", mToken.balanceOf(Bob),
dToken.balanceOf(Bob));
        console2.log("Tom's   balances: MOM = %d, DAD = %d", mToken.balanceOf(Tom),
dToken.balanceOf(Tom));
        console2.log("Staking balances: MOM = %d, DAD = %d",
mToken.balanceOf(address(staking)), dToken.balanceOf(address(staking)));
    }

    function testUserCannotClaimExistedRewardsAfterUnstaking() public {

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

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

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

        console2.log("Manager updates rewards");
        staking.update_rewards(addresses, stakers_amounts, odysseys_ids,
odysseys_amounts, block.timestamp - 2 minutes);

        printStatistic();

        vm.warp(block.timestamp + 10 days);
        console2.log("Bob retakes 5000000000 MOM from Odyssey#1 to Odyssey#2");
        vm.prank(Bob);
        staking.restake(odyssey_1, odyssey_2, 5e9 , Staking.Token.MOM);

        console2.log("Tom retakes 5000000000 MOM from Odyssey#2 to Odyssey#1");
        vm.prank(Tom);
        staking.restake(odyssey_2, odyssey_1, 5e9 , Staking.Token.MOM);

        printStatistic();

        vm.warp(block.timestamp + 10 days);
        console2.log("Bob unstakes 5000000000 DAD tokens in Odyssey#1");
        vm.prank(Bob);
        staking.unstake(odyssey_1, Staking.Token.DAD);

        console2.log("Bob unstakes 5000000000 MOM tokens in Odyssey#2");
        vm.prank(Bob);
```

```
        staking.unstake(odyssey_2, Staking.Token.MOM);

        vm.warp(block.timestamp + 12 days);
        console2.log("Tom unstakes 10000000000 MOM tokens in Odyssey#1");
        vm.prank(Tom);
        staking.unstake(odyssey_1, Staking.Token.MOM);

        vm.warp(block.timestamp + 22 days);
        console2.log("Bob claims unstaked tokens");
        vm.prank(Bob);
        staking.claim_unstaked_tokens();
        console2.log("Tom claims unstaked tokens");
        vm.prank(Tom);
        staking.claim_unstaked_tokens();

        console2.log("Bob claims rewards and reverts");
        vm.expectRevert();
        vm.prank(Bob);
        staking.claim_rewards();
        console2.log("Tom claims rewards and reverts");
        vm.expectRevert();
        vm.prank(Tom);
        staking.claim_rewards();

        printStatistic();

    }

}
```

Execution command:

```
forge test --mc StakingTestNew --mt testUserCannotClaimExistedRewardsAfterUnstaking
-vvvv
```

The output is:

```
[⠊] Compiling...
[⠒] Compiling 7 files with 0.8.18
[⠢] Solc 0.8.18 finished in 3.28s
Compiler run successful!

Running 1 test for test/StakingTest.t.sol:StakingTestNew
[PASS] testUserCannotClaimExistedRewardsAfterUnstaking() (gas: 1531455)
Logs:
  Bob stakes 5000000000 MOM tokens in Odyssey#1
  Bob stakes 5000000000 DAD tokens in Odyssey#1
  Tom stakes 5000000000 MOM tokens in Odyssey#1
  Tom stakes 5000000000 MOM tokens in Odyssey#2
  Manager updates rewards
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Print Info Start~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  Currently total staked is 20000000000
  Length of `staked_odysseys`= 2
  --------------------Staker info for user :
0x0000000000000000000000000000000000000123--------------------
  user=0x0000000000000000000000000000000000000123, total_rewards=10000000000,
total_staked=10000000000,
  dad_amount=5000000000, mom_amount=5000000000
  --------------------Staker info for user :
0x0000000000000000000000000000000000000456--------------------
  user=0x0000000000000000000000000000000000000456, total_rewards=10000000000,
total_staked=10000000000,
  dad_amount=0, mom_amount=10000000000
  --------------------Odyssey info for odyssey_id : 604472133179351442128897---------
----------
  odyssey_id= 604472133179351442128897, total_staked_into= 15000000000,
total_stakers= 2,
  total_rewards= 10000000000, staked_odysseys_index= 0
  --------------------StakedBy info for user:
0x0000000000000000000000000000000000000123 in odyssey_id : 604472133179351442128897-
----------
  user= 0x0000000000000000000000000000000000000123, total_amount= 10000000000,
dad_amount= 5000000000,
  mom_amount= 5000000000, timestamp= 259201, effective_timestamp= 129601
  --------------------StakedBy info for user:
0x0000000000000000000000000000000000000456 in odyssey_id : 604472133179351442128897-
----------
  user= 0x0000000000000000000000000000000000000456, total_amount= 5000000000,
dad_amount= 0,
  mom_amount= 5000000000, timestamp= 259201, effective_timestamp= 259201
  --------------------Odyssey info for odyssey_id : 604472133179351442128898---------
----------
  odyssey_id= 604472133179351442128898, total_staked_into= 5000000000,
total_stakers= 1,
  total_rewards= 10000000000, staked_odysseys_index= 1
```

```
  --------------------StakedBy info for user:
0x0000000000000000000000000000000000000456 in odyssey_id : 604472133179351442128898-
------------------
  user= 0x0000000000000000000000000000000000000456, total_amount= 5000000000,
dad_amount= 0,
  mom_amount= 5000000000, timestamp= 259201, effective_timestamp= 259201
  --------------------Show Balances--------------------
  Bob's   balances: MOM = 15000000000, DAD = 15000000000
  Tom's   balances: MOM = 10000000000, DAD = 20000000000
  Staking balances: MOM = 15000000000, DAD = 5000000000
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Print Info End~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  Bob retakes 5000000000 MOM from Odyssey#1 to Odyssey#2
  Tom retakes 5000000000 MOM from Odyssey#2 to Odyssey#1
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Print Info Start~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  Currently total staked is 20000000000
  Length of `staked_odysseys`= 2
  --------------------Staker info for user :
0x0000000000000000000000000000000000000123--------------------
  user=0x0000000000000000000000000000000000000123, total_rewards=10000000000,
total_staked=10000000000,
  dad_amount=5000000000, mom_amount=5000000000
  --------------------Staker info for user :
0x0000000000000000000000000000000000000456--------------------
  user=0x0000000000000000000000000000000000000456, total_rewards=10000000000,
total_staked=10000000000,
  dad_amount=0, mom_amount=10000000000
  --------------------Odyssey info for odyssey_id : 604472133179351442128897---------
----------
  odyssey_id= 604472133179351442128897, total_staked_into= 15000000000,
total_stakers= 2,
  total_rewards= 10000000000, staked_odysseys_index= 0
  --------------------StakedBy info for user:
0x0000000000000000000000000000000000000123 in odyssey_id : 604472133179351442128897-
------------------
  user= 0x0000000000000000000000000000000000000123, total_amount= 5000000000,
dad_amount= 5000000000,
  mom_amount= 0, timestamp= 1123201, effective_timestamp= 604799
  --------------------StakedBy info for user:
0x0000000000000000000000000000000000000456 in odyssey_id : 604472133179351442128897-
------------------
  user= 0x0000000000000000000000000000000000000456, total_amount= 10000000000,
dad_amount= 0,
  mom_amount= 10000000000, timestamp= 1123201, effective_timestamp= 691201
  --------------------Odyssey info for odyssey_id : 604472133179351442128898---------
----------
  odyssey_id= 604472133179351442128898, total_staked_into= 5000000000,
total_stakers= 1,
  total_rewards= 10000000000, staked_odysseys_index= 1
  --------------------StakedBy info for user:
0x0000000000000000000000000000000000000123 in odyssey_id : 604472133179351442128898-
------------------
```

```
  user= 0x0000000000000000000000000000000000000123, total_amount= 5000000000,
dad_amount= 0,
  mom_amount= 5000000000, timestamp= 1123201, effective_timestamp= 1123201
  -------------------Show Balances-------------------
  Bob's   balances: MOM = 15000000000, DAD = 15000000000
  Tom's   balances: MOM = 10000000000, DAD = 20000000000
  Staking balances: MOM = 15000000000, DAD = 5000000000
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Print Info End~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  Bob unstakes 5000000000 DAD tokens in Odyssey#1
  Bob unstakes 5000000000 MOM tokens in Odyssey#2
  Tom unstakes 10000000000 MOM tokens in Odyssey#1
  Bob claims unstaked tokens
  Tom claims unstaked tokens
  Bob claims rewards and reverts
  Tom claims rewards and reverts
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Print Info Start~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  Currently total staked is 0
  Length of `staked_odysseys`= 2
  -------------------Staker info for user :
0x0000000000000000000000000000000000000123--------------------
  user=0x0000000000000000000000000000000000000000, total_rewards=0, total_staked=0,
  dad_amount=0, mom_amount=0
  -------------------Staker info for user :
0x0000000000000000000000000000000000000456--------------------
  user=0x0000000000000000000000000000000000000000, total_rewards=0, total_staked=0,
  dad_amount=0, mom_amount=0
  -------------------Odyssey info for odyssey_id : 604472133179351442128897---------
----------
  odyssey_id= 604472133179351442128897, total_staked_into= 0, total_stakers= 0,
  total_rewards= 10000000000, staked_odysseys_index= 0
  -------------------Odyssey info for odyssey_id : 604472133179351442128898---------
----------
  odyssey_id= 604472133179351442128898, total_staked_into= 0, total_stakers= 0,
  total_rewards= 10000000000, staked_odysseys_index= 1
  -------------------Show Balances-------------------
  Bob's   balances: MOM = 20000000000, DAD = 20000000000
  Tom's   balances: MOM = 20000000000, DAD = 20000000000
  Staking balances: MOM = 0, DAD = 0
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Print Info End~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Traces:
  ...
    ├─ [0] console::log(Bob claims rewards and reverts) [staticcall]
    │   └─ ← ()
    ├─ [0] VM::expectRevert()
    │   └─ ← ()
    ├─ [0] VM::prank(0x0000000000000000000000000000000000000123)
    │   └─ ← ()
    ├─ [573] Staking::claim_rewards()
    │   └─ ← "No rewards available"
```

```
    ├─ [0] console::log(Tom claims rewards and reverts) [staticcall]
    │   └─ ← ()
    ├─ [0] VM::expectRevert()
    │   └─ ← ()
    ├─ [0] VM::prank(0x0000000000000000000000000000000000000456)
    │   └─ ← ()
    ├─ [573] Staking::claim_rewards()
    │   └─ ← "No rewards available"
```

Recommendation

We recommend reviewing the current design to ensure that it aligns with the original requirement regarding the rewards claimed by stakers. For example, in order to address this issue, the `_unstake` function can force users to claim reward **before** all the staking tokens are removed OR add corresponding logic to ensure the reward has been claimed before deleting the `stakers` variable.

Alleviation

[Momentum Team, 07/03/2023]: Issue acknowledged. Changes have been reflected in the commit abc630afff3c58b2fb698612d47fa8d0b545c78d.

STK-01 | CENTRALIZED CONTROL OF CONTRACT UPGRADE

Category	Severity	Location	Status
Centralization	Major	contracts/staking/Staking.sol (938a1bdb7d02c98cc2afd097cb0d9981eb58499d): 16	Acknowledged

Description

The `Staking` 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 not have 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.

STK-14 | DOUBLE COUNTING WHEN UNSTAKING

Category	Severity	Location	Status
Logical Issue	Major	contracts/staking/Staking.sol (938a1bdb7d02c98cc2afd097cb0d9981eb58499d): 476	Resolved

Description

When unstaking, the staker's `total_staked` value can be decreased twice, causing the `total_staked` variable to be lower than expected. This can lead to disruptions for future unstakes.

In the `staker.total_staked > amount` case, if `_staked_by.total_amount > amount`, then `staker.total_staked` is decreased by amount.

```
        if(staker.total_staked > amount) {
            if(_staked_by.total_amount > amount) {
                ...
                staker.total_staked -= amount;
```

However, `staker.total_staked` is again decreased on line 487, regardless if `_staked_by.total_amount > amount` is true or not.

```
        if(staker.total_staked > amount) {
            if(_staked_by.total_amount > amount) {
                ...
            } else {
                ...
            }
            staker.total_staked -= amount;
```

This can cause a double count when unstaking.

Proof of Concept

The following test written in foundry shows that a user can have an `total_staked` value of 0 while having a positive amount of staked tokens.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.0;

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

contract StakingTest is Test {
    Staking staking;
    OdysseyNFT odysseyNFT;
    DADToken dadToken;
    MOMToken momToken;

    function setUp() public {
        staking = new Staking();
        odysseyNFT = new OdysseyNFT("Odyssey", "ODSY", 1000e18, 1000e18, "");
//maxTokens, maxOdysseysPerWallet
        dadToken = new DADToken();
        momToken = new MOMToken(1000e18); //initialSupply
        staking.initialize(address(momToken), address(dadToken),
address(odysseyNFT));

        dadToken.grantRole(dadToken.TRANSFER_ROLE(), address(staking));
        momToken.grantRole(momToken.MINTER_ROLE(), address(staking));

        dadToken.mint(address(this), 1000e18);

        momToken.approve(address(staking), 1000e18);
        dadToken.approve(address(staking), 1000e18);
    }

    function testUnstakeDoubleCount() public {
        staking.stake(0, 100, Staking.Token.MOM);
        staking.stake(0, 100, Staking.Token.DAD);
        ( , , uint256 total_stake_before, , ) = staking.stakers(address(this));
        assert(total_stake_before == 200);

        staking.unstake(0, Staking.Token.MOM);
        ( , , uint256 total_stake_after , uint256 dad_amount, ) =
staking.stakers(address(this));
        assert(total_stake_after == 0);
        assert(dad_amount == 100);
    }
}
```

Recommendation

It is recommended to revisit the unstaking logic and make sure that `staker.total_staked` is only decreased once.

Alleviation

[Momentum Team, 06/08/2023]: The team heeded the advice to update the logic in function `_unstake()` of `Staking` contract and changes were made in commit 6fc42dd5ee8538436f14146d50912feb627edc98.

STK-16 | FAULTY RESET OF USER TOKENS

Category	Severity	Location	Status
Logical Issue	Major	contracts/staking/Staking.sol (938a1bdb7d02c98cc2afd097cb0d9981eb58499d): 484~487	Resolved

Description

If a user stakes tokens in two different odysseys and wants to unstake from one of them, the `Mom token` or `Dad token` amount is currently reset to `zero`. This occurs because of the check in `Line 466: if(staker.total_staked > amount)`. However, instead of resetting the token amount, the total staked amount should be decreased by the `unstaked amount`.

Proof of Concept

The following fuzz test written in foundry shows that a user token amount reset to zero.

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

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

contract TestStaking is Test {
    Staking public staking;
    MOMToken mToken;
    DADToken dToken;
    OdysseyNFT oNFT;

    function setUp() public {
        staking = new Staking();
        mToken = new MOMToken(1_000_000 * 10 * 18);
        dToken = new DADToken();
        oNFT = new OdysseyNFT("NFT", "N", 1000, 5, "abc.erc1");
        staking.initialize(address(mToken), address(dToken), address(oNFT));
    }

function test_if_unstake_works(uint256 amount) external {
        vm.assume(amount > 10);
        vm.assume(amount < mToken.balanceOf(address(this)));

        mToken.approve(address(staking), amount * 2);
        staking.stake(1, amount, Staking.Token.MOM);
        staking.stake(2, amount, Staking.Token.MOM);

        (,, uint256 total_staked,,) = staking.stakers(address(this));

        assertTrue((total_staked == amount + amount), "Total staked should be equal
to 2 * amount.");

        staking.unstake(1, Staking.Token.MOM);

        (,,,,uint momAmount) = staking.stakers(address(this));

        assertTrue((momAmount != 0), "The total amount should not be zero. ");
    }
}
```

Recommendation

We recommend the team revisit the logic that updates the token amount of users.

Alleviation

[Momentum Team, 06/08/2023]: The team heeded the advice to update the logic in function `_unstake()` of `Staking` contract and changes were made in commit 6fc42dd5ee8538436f14146d50912feb627edc98.

STI-03 `MOM` OR `DAD` AMOUNT OF STAKER INCREASED AFTER RESTAKING

Category	Severity	Location	Status
Coding Issue	Medium	contracts/staking/Staking.sol (update_0608): 520	Resolved

Description

In the `_restake()` function of `Staking` contract, it just reduces the `total_staked` of staker, but there is no logic to decrease staker's `MOM` or `DAD` token amount.

```
520  Staker storage staker = stakers[msg.sender];
```

```
552  staker.total_staked -= amount;
```

However, in the `_do_stake()` function, the amount of `MOM` or `DAD` tokens for the staker is increased. This can cause the total amount of `MOM` and `DAD` tokens to be greater than the `total_staked` amount for the staker, which is unexpected.

```
401          staker.total_staked += amount;
402
403          token == Token.DAD ? staker.dad_amount += amount : staker.mom_amount +=
amount;
404          total_staked += amount;
```

Scenario

- Bob stakes 5000000000 MOM tokens in Odyssey#1
- Bob stakes 5000000000 DAD tokens in Odyssey#1
- Tom stakes 5000000000 MOM tokens in Odyssey#1
- Tom stakes 5000000000 MOM tokens in Odyssey#2
- Bob retakes 5000000000 MOM from Odyssey#1 to Odyssey#2
- Tom retakes 5000000000 MOM from Odyssey#2 to Odyssey#1

After restaking, list the stakers info as below:

```
  -------------------Staker info for user :
0x0000000000000000000000000000000000000123--------------------
  user=0x0000000000000000000000000000000000000123, total_rewards=0,
total_staked=10000000000, dad_amount=5000000000, mom_amount=10000000000
  -------------------Staker info for user :
0x0000000000000000000000000000000000000456--------------------
  user=0x0000000000000000000000000000000000000456, total_rewards=0,
total_staked=10000000000, dad_amount=0, mom_amount=15000000000
```

Proof of Concept

The following proof of concept uses Foundry to show `MOM` or `DAD` amount of staker is increased after restaking.

```
// 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 StakingTestNew is Test {

    Staking public staking;
    MOMToken public mToken;
    DADToken public dToken;
    OdysseyNFT public odysseyNFT;

    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[] addresses = [Bob, Tom];
    uint256[] odysseys_ids;


    function setUp() public {
        mToken = new MOMToken(1e20);
        dToken = new DADToken();
        staking = new Staking();
        odysseyNFT = new OdysseyNFT(
            "Odyssey_NFT",
            "ODS",
            21000,
            150,
            "ipfs://odyssey.nft"
        );

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

        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);
        vm.stopPrank();

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

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

        odysseyNFT.safeMint(Owner_Odyssey_1);
        odyssey_1 = odysseyNFT.currentId();
        odysseyNFT.safeMint(Owner_Odyssey_2);
        odyssey_2 = odysseyNFT.currentId();
        odysseys_ids = [odyssey_1, odyssey_2];

    }

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

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

    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.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
        );
    }

    function printStakedBy(uint256 odyssey_id) private view {
        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-------------------",  stakedBy.user, odyssey_id);
            console2.log("user= %s, total_amount= %d, dad_amount= %d,",
                stakedBy.user, stakedBy.total_amount, stakedBy.dad_amount
            );
            console2.log("mom_amount= %d, timestamp= %d, effective_timestamp= %d",
                stakedBy.mom_amount, stakedBy.timestamp,
stakedBy.effective_timestamp
            );
        }

    }

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

    function printStatistic() private view {
        console2.log("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Print Info
Start~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
        printTotalStaked();
        printStakedOdysseys();
        for (uint i = 0; i < addresses.length; i++) {
            printStakerFor(addresses[i]);
        }
        for (uint i = 0; i < odysseys_ids.length; i++) {
            printOdysseyFor(odysseys_ids[i]);
            printStakedBy(odysseys_ids[i]);
        }
        printBalances();
        console2.log("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Print Info
End~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
    }

    function printBalances() private view {
        console2.log("-------------------Show Balances--------------------");
        console2.log("Bob's   balances: MOM = %d, DAD = %d", mToken.balanceOf(Bob),
dToken.balanceOf(Bob));
```

```
        console2.log("Tom's   balances: MOM = %d, DAD = %d", mToken.balanceOf(Tom),
dToken.balanceOf(Tom));
        console2.log("Staking balances: MOM = %d, DAD = %d",
mToken.balanceOf(address(staking)), dToken.balanceOf(address(staking)));
    }

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

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

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

        printStatistic();

        uint staked_by_index_Bob;
        uint staked_by_index_Tom;
        staked_by_index_Bob = staking.staked_by_indexes(odyssey_1, Bob);
        staked_by_index_Tom = staking.staked_by_indexes(odyssey_1, Tom);
        console2.log("%d, staked_by_index_Bob = %d, staked_by_index_Tom = %d",
odyssey_1, staked_by_index_Bob, staked_by_index_Tom);
        staked_by_index_Bob = staking.staked_by_indexes(odyssey_2, Bob);
        staked_by_index_Tom = staking.staked_by_indexes(odyssey_2, Tom);
        console2.log("%d, staked_by_index_Bob = %d, staked_by_index_Tom = %d",
odyssey_2, staked_by_index_Bob, staked_by_index_Tom);


        vm.warp(block.timestamp + 10 days);
        console2.log("Bob retakes 5000000000 MOM from Odyssey#1 to Odyssey#2");
        vm.prank(Bob);
        staking.restake(odyssey_1, odyssey_2, 5e9 , Staking.Token.MOM);

        console2.log("Tom retakes 5000000000 MOM from Odyssey#2 to Odyssey#1");
        vm.prank(Tom);
        staking.restake(odyssey_2, odyssey_1, 5e9 , Staking.Token.MOM);

        staked_by_index_Bob = staking.staked_by_indexes(odyssey_1, Bob);
        staked_by_index_Tom = staking.staked_by_indexes(odyssey_1, Tom);
        console2.log("%d, staked_by_index_Bob = %d, staked_by_index_Tom = %d",
odyssey_1, staked_by_index_Bob, staked_by_index_Tom);
```

```
        staked_by_index_Bob = staking.staked_by_indexes(odyssey_2, Bob);
        staked_by_index_Tom = staking.staked_by_indexes(odyssey_2, Tom);
        console2.log("%d, staked_by_index_Bob = %d, staked_by_index_Tom = %d",
odyssey_2, staked_by_index_Bob, staked_by_index_Tom);

        printStatistic();
    }
}
```

Execution command:

```
forge test --mc StakingTestNew --mt testUserRestake -vvv
```

The output is:

```
[⠊] Compiling...
[⠊] Compiling 7 files with 0.8.20
[⠊] Solc 0.8.20 finished in 7.03s
Compiler run successful!

Running 1 test for test/StakingTest.t.sol:StakingTestNew
[PASS] testUserRestake() (gas: 1295692)
Logs:
  Bob stakes 5000000000 MOM tokens in Odyssey#1
  Bob stakes 5000000000 DAD tokens in Odyssey#1
  Tom stakes 5000000000 MOM tokens in Odyssey#1
  Tom stakes 5000000000 MOM tokens in Odyssey#2
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Print Info Start~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  Currently total staked is 20000000000
  Length of `staked_odysseys`= 2
  --------------------Staker info for user :
0x0000000000000000000000000000000000000123--------------------
  user=0x0000000000000000000000000000000000000123, total_rewards=0,
total_staked=10000000000,
  dad_amount=5000000000, mom_amount=5000000000
  --------------------Staker info for user :
0x0000000000000000000000000000000000000456--------------------
  user=0x0000000000000000000000000000000000000456, total_rewards=0,
total_staked=10000000000,
  dad_amount=0, mom_amount=10000000000
  --------------------Odyssey info for odyssey_id : 604472133179351442128897---------
----------
  odyssey_id= 604472133179351442128897, total_staked_into= 15000000000,
total_stakers= 2,
  total_rewards= 0, staked_odysseys_index= 0
  --------------------StakedBy info for user:
0x0000000000000000000000000000000000000123 in odyssey_id : 604472133179351442128897-
------------------
  user= 0x0000000000000000000000000000000000000123, total_amount= 10000000000,
dad_amount= 5000000000,
  mom_amount= 5000000000, timestamp= 259201, effective_timestamp= 129601
  --------------------StakedBy info for user:
0x0000000000000000000000000000000000000456 in odyssey_id : 604472133179351442128897-
------------------
  user= 0x0000000000000000000000000000000000000456, total_amount= 5000000000,
dad_amount= 0,
  mom_amount= 5000000000, timestamp= 259201, effective_timestamp= 259201
  --------------------Odyssey info for odyssey_id : 604472133179351442128898---------
----------
  odyssey_id= 604472133179351442128898, total_staked_into= 5000000000,
total_stakers= 1,
  total_rewards= 0, staked_odysseys_index= 1
  --------------------StakedBy info for user:
0x0000000000000000000000000000000000000456 in odyssey_id : 604472133179351442128898-
------------------
```

```
  user= 0x0000000000000000000000000000000000000456, total_amount= 5000000000,
dad_amount= 0,
  mom_amount= 5000000000, timestamp= 259201, effective_timestamp= 259201
  -------------------Show Balances-------------------
  Bob's   balances: MOM = 15000000000, DAD = 15000000000
  Tom's   balances: MOM = 10000000000, DAD = 20000000000
  Staking balances: MOM = 15000000000, DAD = 5000000000
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Print Info End~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  604472133179351442128897, staked_by_index_Bob = 1, staked_by_index_Tom = 2
  604472133179351442128898, staked_by_index_Bob = 0, staked_by_index_Tom = 1
  Bob retakes 5000000000 MOM from Odyssey#1 to Odyssey#2
  Tom retakes 5000000000 MOM from Odyssey#2 to Odyssey#1
  604472133179351442128897, staked_by_index_Bob = 1, staked_by_index_Tom = 2
  604472133179351442128898, staked_by_index_Bob = 1, staked_by_index_Tom = 1
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Print Info Start~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  Currently total staked is 20000000000
  Length of `staked_odysseys`= 2
  -------------------Staker info for user :
0x0000000000000000000000000000000000000123-------------------
  user=0x0000000000000000000000000000000000000123, total_rewards=0,
total_staked=10000000000,
  dad_amount=5000000000, mom_amount=10000000000
  -------------------Staker info for user :
0x0000000000000000000000000000000000000456-------------------
  user=0x0000000000000000000000000000000000000456, total_rewards=0,
total_staked=10000000000,
  dad_amount=0, mom_amount=15000000000
  -------------------Odyssey info for odyssey_id : 604472133179351442128897---------
----------
  odyssey_id= 604472133179351442128897, total_staked_into= 15000000000,
total_stakers= 2,
  total_rewards= 0, staked_odysseys_index= 0
  -------------------StakedBy info for user:
0x0000000000000000000000000000000000000123 in odyssey_id : 604472133179351442128897-
------------------
  user= 0x0000000000000000000000000000000000000123, total_amount= 5000000000,
dad_amount= 5000000000,
  mom_amount= 0, timestamp= 1123201, effective_timestamp= 604799
  -------------------StakedBy info for user:
0x0000000000000000000000000000000000000456 in odyssey_id : 604472133179351442128897-
------------------
  user= 0x0000000000000000000000000000000000000456, total_amount= 10000000000,
dad_amount= 0,
  mom_amount= 10000000000, timestamp= 1123201, effective_timestamp= 691201
  -------------------Odyssey info for odyssey_id : 604472133179351442128898---------
----------
  odyssey_id= 604472133179351442128898, total_staked_into= 5000000000,
total_stakers= 1,
  total_rewards= 0, staked_odysseys_index= 1
```

```
  ------------------StakedBy info for user:
0x0000000000000000000000000000000000000123 in odyssey_id : 604472133179351442128898-
-----------------
  user= 0x0000000000000000000000000000000000000123, total_amount= 5000000000,
dad_amount= 0,
  mom_amount= 5000000000, timestamp= 1123201, effective_timestamp= 1123201
  ------------------Show Balances------------------
  Bob's   balances: MOM = 15000000000, DAD = 15000000000
  Tom's   balances: MOM = 10000000000, DAD = 20000000000
  Staking balances: MOM = 15000000000, DAD = 5000000000
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~Print Info End~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Test result: ok. 1 passed; 0 failed; finished in 8.85ms
```

Recommendation

We recommend to refactor the code in function `_restake()` to decrease `MOM` or `DAD` amount of the staker. The possible solution is as below:

```
552         staker.total_staked -= amount;
553         token == Token.DAD ? staker.dad_amount -= amount : staker.mom_amount -=
 amount; //added to fix
554         // Restake in the 'to' Odyssey
555         _do_stake(to_odyssey_id, amount, token);
```

Alleviation

[Momentum Team, 06/15/2023]: The team heeded the advice to fix the issue in the `Staking` contract and changes were included in commit 79015fb53e58603e6f0137a6e44ded6f39e282d4.

STI-04 | POSSIBLE FOR STAKES TO BE LOCKED IN THE CONTRACT

Category	Severity	Location	Status
Design Issue	Medium	contracts/staking/Staking.sol (update_0608): 451, 513	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.

STK-03 | LACK OF STORAGE GAP IN UPGRADEABLE CONTRACT

Category	Severity	Location	Status
Logical Issue	Medium	contracts/staking/Staking.sol (938a1bdb7d02c98cc2afd097cb0d9981eb58499d): 16	Resolved

Description

There is no storage gap preserved in the logic contract. Any logic contract that acts as a base contract that needs to be inherited by other upgradeable child should have a reasonable size of storage gap preserved for the new state variable introduced by the future upgrades.

Recommendation

We recommend having a storage gap of a reasonable size preserved in the logic contract in case that new state variables are introduced in future upgrades. For more information, please refer to: https://docs.openzeppelin.com/contracts/3.x/upgradeable#storage_gaps.

Alleviation

[Momentum Team, 06/08/2023]: The team heeded the advice to add storage gaps `__gap` in the `Staking` contract and changes were made in commit 6fc42dd5ee8538436f14146d50912feb627edc98.

STK-05 INCORRECT USE OF `addresses` LENGTH IN LOOP AND LACK OF LENGTH CHECK

Category	Severity	Location	Status
Logical Issue	Medium	contracts/staking/Staking.sol (938a1bdb7d02c98cc2afd097cb0d9981eb58499d): 272, 276	Resolved

Description

The for loop in the code below uses the length of the `addresses` array to iterate over the `odysseys_ids` and `odysseys_amounts` arrays. However, the length of the `addresses` array has no relationship with the lengths of `odysseys_ids` and `odysseys_amounts`. This can result in unexpected behavior or errors in the code.

```
276         for(uint i = 0; i < addresses.length; i++) {
277             odysseys[odysseys_ids[i]].total_rewards += odysseys_amounts[i];
278         }
```

Furthermore, the code does not check if the length of the `odysseys_ids` array is the same as the length of the `odysseys_amounts` array and that the length of the `addresses` array is the same as the length of the `stakers_amounts` array. If the length of one of these arrays is larger than that of the other array, the code may revert due to an out of bounds error.

Recommendation

We recommend to use length of `odysseys_ids` to iterate over the `odysseys_ids` and `odysseys_amounts` arrays directly. For example:

```
    require(odysseys_ids.length == odysseys_amounts.length, "Length is not
matched.");
    for(uint i = 0; i < odysseys_ids.length; i++) {
        odysseys[odysseys_ids[i]].total_rewards += odysseys_amounts[i];
    }
```

Alleviation

[Momentum Team, 06/08/2023]: The team heeded the advice to fix the loop length in function `update_rewards()` of `Staking` contract and changes were included in commit 6fc42dd5ee8538436f14146d50912feb627edc98.

STK-06 POTENTIAL MULTIPLE COUNTING ON `odyssey.total_stakers`

Category	Severity	Location	Status
Logical Issue	Medium	contracts/staking/Staking.sol (938a1bdb7d02c98cc2afd097cb0d9981eb58499d): 409, 644	Resolved

Description

In the `Staking` contract, every time a user stakes `MOM` or `DAD` tokens on a certain Odyssey, the `total_stakers` increases by 1. If the same user stakes multiple times on the same Odyssey, the `total_stakers` count is incremented each time. When a user unstakes their tokens from an Odyssey, the `total_stakers` decreases by 1 if there are no more staked tokens on that Odyssey. Once `total_stakers` reaches zero, the staked records will be removed out of contract states `staked_odysseys` and `odysseys`.

```
644     function decrease_odyssey_total_stakers(uint256 odyssey_id, uint256 amount)
 private {
645         Odyssey storage odyssey = odysseys[odyssey_id];
646         odyssey.total_stakers--;
647         odyssey.total_staked_into -= amount;
648         if(odyssey.total_stakers == 0 && odyssey.total_rewards == 0) {
649             uint256 last_item = staked_odysseys[staked_odysseys.length-1];
650             odysseys[last_item].staked_odysseys_index = odyssey.
staked_odysseys_index;
651             staked_odysseys[odyssey.staked_odysseys_index] = last_item;
652             staked_odysseys.pop();
653             delete odysseys[odyssey_id];
654         }
655     }
```

However, the `staked_odysseys` and `odysseys states` may not be removed if `total_stakers` is counted multiple times but is only decreased once. This can be an issue and may cause unexpected behavior.

Proof of Concept

The following proof of concept is using Foundry to test the case an user stakes multiple times , unstakes and last claims the unstaked tokens on the same Odyssey.

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

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

contract StakingTest is Test {

    Staking public staking;
    MOMToken public mToken;
    DADToken public dToken;
    OdysseyNFT public odysseyNFT;

    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 user;
    uint256 total_rewards;
    uint256 total_staked;

    uint256 odyssey_id;
    uint256 total_stakers;
    uint256 staked_odysseys_index;

    function setUp() public {
        mToken = new MOMToken(1e20);
        dToken = new DADToken();
        staking = new Staking();
        odysseyNFT = new OdysseyNFT(
            "Odyssey_NFT",
            "ODS",
            21000,
            150,
            "ipfs://"
        );

        staking.initialize(
            address(mToken),
            address(dToken),
            address(odysseyNFT)
```

```
        );

        mToken.transfer(Bob, 1e10);
        mToken.transfer(Tom, 1e10);
        dToken.mint(Bob, 1e10);
        dToken.mint(Tom, 1e10);

        vm.startPrank(Bob);
        mToken.approve(address(staking), 1e20);
        dToken.approve(address(staking), 1e20);
        vm.stopPrank();

        vm.startPrank(Tom);
        mToken.approve(address(staking), 1e20);
        dToken.approve(address(staking), 1e20);
        vm.stopPrank();

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

        odysseyNFT.safeMint(Owner_Odyssey_1);
        odyssey_1 = odysseyNFT.currentId();
        odysseyNFT.safeMint(Owner_Odyssey_2);
        odyssey_2 = odysseyNFT.currentId();

    }


    function testOdysseyRecordsNotRemoved() public {
        console2.log("Initial MOM balance of Staking: ",
mToken.balanceOf(address(staking)));
        vm.startPrank(Bob);
        console2.log("Bob stakes 5000000000 MOM tokens in Odyssey#1 for the first
time");
        staking.stake(
            odyssey_1,
            5e9,
            Staking.Token.MOM
        );
        (odyssey_id,,total_stakers,,staked_odysseys_index) = staking.odysseys(1);
        console2.log("After the first  staking, total_stakers = %d,
staked_odysseys_index = %d", total_stakers, staked_odysseys_index);
        console2.log("After the first  staking, MOM balance of Staking: ",
mToken.balanceOf(address(staking)));

        vm.warp(block.timestamp + 3 days);
        console2.log("Bob stakes 5000000000 MOM tokens in Odyssey#1 for the second
time");
        staking.stake(
            odyssey_1,
```

```
            5e9,
            Staking.Token.MOM
        );

        console2.log("After the second staking, MOM balance of Staking: ",
mToken.balanceOf(address(staking)));
        (odyssey_id,,total_stakers,,staked_odysseys_index) =
staking.odysseys(odyssey_1);
        console2.log("After the second staking, total_stakers = %d,
staked_odysseys_index = %d", total_stakers, staked_odysseys_index);

        vm.warp(block.timestamp + 4 days);
        console2.log("Bob unstakes MOM tokens in Odyssey#1");
        staking.unstake(
            odyssey_1,
            Staking.Token.MOM
        );
        vm.warp(block.timestamp + 7 days);
        console2.log("Bob claiming unstaked MOM tokens");
        staking.claim_unstaked_tokens();
        vm.stopPrank();
        console2.log("After claiming, MOM balance of Staking: ",
mToken.balanceOf(address(staking)));

        (user, total_rewards, total_staked,,) = staking.stakers(address(Bob));
        console2.log("After claiming, user = %s, total_rewards = %d, total_staked =
%d", user, total_rewards, total_staked);
        (odyssey_id,,total_stakers,,staked_odysseys_index) =
staking.odysseys(odyssey_1);
        console2.log("After claiming, odyssey_id = %d, total_stakers = %d
staked_odysseys_index = %d", odyssey_id, total_stakers, staked_odysseys_index);
        uint256[] memory odysseys =  staking.get_staked_odysseys();
        console2.log("After claiming, the length of `staked_odysseys` is ",
odysseys.length);
    }


}
```

The result is as below:

```
% forge test --match-contract=StakingTest -vvv
[⠒] Compiling...
[⠒] Compiling 1 files with 0.8.18
[⠢] Solc 0.8.18 finished in 1.61s
Compiler run successful!

Running 1 test for test/Staking.t.sol:StakingTest
[PASS] testOdysseyRecordsNotRemoved() (gas: 501510)
Logs:
  Initial MOM balance of Staking:  0
  Bob stakes 5000000000 MOM tokens in Odyssey#1 for the first time
  After the first  staking, total_stakers = 1, staked_odysseys_index = 0
  After the first  staking, MOM balance of Staking:  5000000000
  Bob stakes 5000000000 MOM tokens in Odyssey#1 for the second time
  After the second staking, MOM balance of Staking:  10000000000
  After the second staking, total_stakers = 2, staked_odysseys_index = 0
  Bob unstakes MOM tokens in Odyssey#1
  Bob claiming unstaked MOM tokens
  After claiming, MOM balance of Staking:  0
  After claiming, user = 0x0000000000000000000000000000000000000000, total_rewards =
0, total_staked = 0
  After claiming, odyssey_id = 1, total_stakers = 1 staked_odysseys_index = 0
  After claiming, the length of `staked_odysseys` is  1

Test result: ok. 1 passed; 0 failed; finished in 5.36ms
```

Recommendation

We recommend to double the logic and refactor it accordingly. For example, move the code `odyssey.total_stakers++;` in Line#409 to the `if` block starting from Line#418.

```
417         // The user is not staking on the odyssey
418         if(index == 0) {
419             odyssey.total_stakers++;
420             index = staked_by[odyssey_id].length;
421             staked_by_indexes[odyssey_id][msg.sender] = index;
422             staked_by[odyssey_id].push();
423             staked_by[odyssey_id][index].user = msg.sender;
424             staked_by[odyssey_id][index].total_amount = amount;
425             token == Token.DAD
426                     ?  staked_by[odyssey_id][index].dad_amount += amount
427                     :  staked_by[odyssey_id][index].mom_amount += amount;
428             staked_by[odyssey_id][index].timestamp = block.timestamp;
429             staked_by[odyssey_id][index].effective_timestamp = block.timestamp;
430         }
```

Alleviation

[Momentum Team, 06/08/2023]: The team heeded the advice to update the logic in function `_stake()` of `Staking` contract and changes were made in commit 6fc42dd5ee8538436f14146d50912feb627edc98.

STK-17 INACCURATE RISE IN THE TOTAL STAKED AMOUNT OF THE USER.

Category	Severity	Location	Status
Logical Issue	Medium	contracts/staking/Staking.sol (938a1bdb7d02c98cc2afd097cb0d9981eb58499d): 466	Resolved

Description

When the `restake()` function is invoked, it calls the `_do_stake()` function, which results in an increase in the total staked amount for a user (`total_staked`). However, this behavior is incorrect since the user is not adding new tokens but simply transferring them to another odyssey. Also, the total staked amount is also increased but should remain unchanged.

Scenario

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

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


contract TestStaking is Test {
    Staking public staking;
    MOMToken mToken;
    DADToken dToken;
    OdysseyNFT oNFT;


    function setUp() public {
        staking = new Staking();
        mToken = new MOMToken(1_000_000 * 10 * 18);
        dToken = new DADToken();
        oNFT = new OdysseyNFT("NFT", "N", 1000, 5, "abc.erc1");
        staking.initialize(address(mToken),address(dToken), address(oNFT));
    }

    // This fuzz test checks if during restake
    // operation, the total staked token amount
    // of user increases.
    function test_if_restake_works(uint256 amount) external {
        vm.assume(amount > 10);
        vm.assume(amount < mToken.balanceOf(address(this)));

        uint256 balance_before_deposit = mToken.balanceOf(address(this));
        mToken.approve(address(staking), amount);
        staking.stake(1, amount, Staking.Token.MOM);

        (,, uint256 total_staked,,) = staking.stakers(address(this));

        assertTrue((total_staked == amount), "Total staked should be equal to staked
amount.");

        staking.restake(1, 2, amount, Staking.Token.MOM);

        (,, total_staked,,) = staking.stakers(address(this));

        assertTrue((total_staked == amount), "The total amount should be equal to
the original amount.");
```

```
    }
}
```

Recommendation

We recommend the team to check the logic that updates the leftover token amount after unstake operation.

Alleviation

[Momentum Team, 06/08/2023]: The team heeded the advice to update the logic in function `_restake()` of `Staking` contract and changes were made in commit 6fc42dd5ee8538436f14146d50912feb627edc98.

STK-18 | `odyssey.total_staked_into` MAY NOT BE DECREASED

Category	Severity	Location	Status
Logical Issue	Medium	contracts/staking/Staking.sol (938a1bdb7d02c98cc2afd097cb0d9981eb58499d): 466	Resolved

Description

In the `_unstake()` function of contract `Staking`, `odyssey.total_staked_into` is only decreased when users staked just one type of token (`MOM` or `DAD`) on one Odyssey and then unstaked it. However, if the user staked on multiple Odysseys or staked two tokens on one Odyssey, while unstaking, the `odyssey.total_staked_into` will not be decreased.

The same issue also occurs in the function `_restake()`.

Recommendation

We recommend to refactor the logic to eunsure `odyssey.total_staked_into` is always decreased accordingly when unstaking.

Alleviation

[Momentum Team, 06/08/2023]: The team heeded the advice to update the logic in function `_unstake()` of `Staking` contract and changes were made in commit 6fc42dd5ee8538436f14146d50912feb627edc98.

STK-19 USER POTENTIALLY CANNOT UNSTAKE CAUSED BY UNDERFLOW

Category	Severity	Location	Status
Logical Issue	Medium	contracts/staking/Staking.sol (938a1bdb7d02c98cc2afd097cb0d9981eb58499d): 675	Resolved

Description

In the function `calculate_effective_timestamp()`, there is a potential underflow issue in Line#675.

```
673         uint new_effective_timestamp = is_stake
674                                         ? ( (current_amount *
actual_effective_timestamp) + (amount * current_timestamp) ) / (current_amount +
amount)
675                                         : ( (current_amount *
actual_effective_timestamp) - (amount * current_timestamp) ) / (current_amount -
amount);
```

The `actual_effective_timestamp` is the the max value between the last staked timestamp `effective_timestamp` and last reward calculation `last_rewards_calculation`. The `current_amount` is the number of tokens that user staked, which is not smaller than `amount`.

However, the `actual_effective_timestamp` is ideally smaller than the current timestamp `current_timestamp`. So `(current_amount * actual_effective_timestamp) - (amount * current_timestamp)` may cause an underflow issue, which will prevent users to unstake their tokens.

Proof of Concept

The following proof of concept is using Foundry to test the case an user cannot unstake due to the `underflow` issue.

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

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

contract StakingTest3 is Test {

    Staking public staking;
    MOMToken public mToken;
    DADToken public dToken;
    OdysseyNFT public odysseyNFT;

    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 user;
    uint256 total_rewards;
    uint256 total_staked;

    uint256 odyssey_id;
    uint256 total_stakers;
    uint256 total_staked_into;
    uint256 staked_odysseys_index;

    address[] addresses = [Bob, Tom];
    uint256[] stakers_amounts = [1e10, 1e10];
    uint256[] odysseys_ids;
    uint256[] odysseys_amounts = [1e10, 1e10];

    function setUp() public {
        mToken = new MOMToken(1e20);
        dToken = new DADToken();
        staking = new Staking();
        odysseyNFT = new OdysseyNFT(
            "Odyssey_NFT",
            "ODS",
            21000,
            150,
            "ipfs://"
```

```
        );

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

        mToken.transfer(Bob, 1e10);
        mToken.transfer(Tom, 1e10);
        dToken.mint(Bob, 1e10);
        dToken.mint(Tom, 1e10);

        vm.startPrank(Bob);
        mToken.approve(address(staking), 1e20);
        dToken.approve(address(staking), 1e20);
        vm.stopPrank();

        vm.startPrank(Tom);
        mToken.approve(address(staking), 1e20);
        dToken.approve(address(staking), 1e20);
        vm.stopPrank();

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

        odysseyNFT.safeMint(Owner_Odyssey_1);
        odyssey_1 = odysseyNFT.currentId();
        odysseyNFT.safeMint(Owner_Odyssey_2);
        odyssey_2 = odysseyNFT.currentId();
        odysseys_ids = [odyssey_1, odyssey_2];

    }


    function testCannotUnstakeCausedByUnderflow() public {
        console2.log("Initial MOM balance of Staking: ",
mToken.balanceOf(address(staking)));
        vm.startPrank(Bob);
        console2.log("Bob stakes 5000000000 MOM tokens in Odyssey#1 for the first
time");
        staking.stake(
            odyssey_1,
            5e9,
            Staking.Token.MOM
        );
        (odyssey_id,total_staked_into,total_stakers,,staked_odysseys_index) =
staking.odysseys(odyssey_1);
```

```
        console2.log("After the first  staking, total_staked_into = %d,
total_stakers = %d, staked_odysseys_index = %d", total_staked_into ,total_stakers,
staked_odysseys_index);
        console2.log("After the first  staking, MOM balance of Staking: ",
mToken.balanceOf(address(staking)));

        vm.warp(block.timestamp + 3 days);
        console2.log("Bob stakes 5000000000 DAD tokens in Odyssey#1 for the second
time after 3 days");
        staking.stake(
            odyssey_1,
            5e9,
            Staking.Token.DAD
        );

        console2.log("After the second staking, MOM balance of Staking: ",
mToken.balanceOf(address(staking)));
        (odyssey_id,total_staked_into,total_stakers,,staked_odysseys_index) =
staking.odysseys(odyssey_1);
        console2.log("After the second staking, total_staked_into = %d,
total_stakers = %d, staked_odysseys_index = %d",total_staked_into, total_stakers,
staked_odysseys_index);

        vm.warp(block.timestamp + 4 days);
        console2.log("Bob unstakes MOM tokens in Odyssey#1 after 4 days and it
reverts");
        vm.expectRevert();
        staking.unstake(
            odyssey_1,
            Staking.Token.MOM
        );
        vm.stopPrank();
    }

}
```

The result is as below:

```
% forge test --match-contract=StakingTest3 -vvv
[⠒] Compiling...
[⠒] Compiling 1 files with 0.8.18
[⠢] Solc 0.8.18 finished in 1.57s
Compiler run successful!

Running 1 test for test/Staking3.t.sol:StakingTest3
[PASS] testCannotUnstakeCausedByUnderflow() (gas: 626053)
Logs:
  Initial MOM balance of Staking:  0
  Bob stakes 5000000000 MOM tokens in Odyssey#1 for the first time
  After the first  staking, total_staked_into = 5000000000, total_stakers = 1,
staked_odysseys_index = 0
  After the first  staking, MOM balance of Staking:  5000000000
  Bob stakes 5000000000 DAD tokens in Odyssey#1 for the second time after 3 days
  After the second staking, MOM balance of Staking:  5000000000
  After the second staking, total_staked_into = 10000000000, total_stakers = 2,
staked_odysseys_index = 0
  Bob unstakes MOM tokens in Odyssey#1 after 4 days and it reverts

Test result: ok. 1 passed; 0 failed; finished in 7.18ms
```

Recommendation

We recommend to check this logic and fix the potential underflow issue.

Alleviation

[Momentum Team, 06/08/2023]: The team heeded the advice to update the logic in function `calculate_effective_timestamp()` of `Staking` contract and changes were made in commit 6fc42dd5ee8538436f14146d50912feb627edc98.

STK-02 | POTENTIAL DAMAGE DUE TO UNPROTECTED INITIALIZER

Category	Severity	Location	Status
Logical Issue	Minor	contracts/staking/Staking.sol (938a1bdb7d02c98cc2afd097cb0d9981eb58499d): 213	Resolved

Description

The function `initialize()` is `public` and can be called by anyone as long as the contract is deployed. This allows an attacker to initialize the implementation contract, which allows for possible phishing attacks where an attacker attempts to have users mistakingly interact with the implementation contract instead of the proxy.

```
213     function initialize(address _mom_token, address _dad_token, address
_odyssey_nfts) initializer public {
```

- `initialize` is an unprotected initializer function.

Recommendation

It is advised to call `_disableInitializers` in the constructor or give the constructor the initializer modifier to prevent the intializer from being called on the logic contract.

Reference: https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable#initializing_the_implementation_contract

Alleviation

[Momentum Team, 07/03/2023]: Issue acknowledged. Changes have been reflected in the commit abc630afff3c58b2fb698612d47fa8d0b545c78d.

STK-07 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	Minor	contracts/staking/Staking.sol (938a1bdb7d02c98cc2afd097cb0d9981eb58499d): 214, 215, 216, 236, 244, 252	Resolved

Description

Addresses are not validated before assignment or external calls, potentially allowing the use of zero addresses and leading to unexpected behavior or vulnerabilities. For example, transferring tokens to a zero address can result in a permanent loss of those tokens.

```
214         mom_token = _mom_token;
```

- `_mom_token` is not zero-checked before being used.

```
215         dad_token = _dad_token;
```

- `_dad_token` is not zero-checked before being used.

```
216         odyssey_nfts = _odyssey_nfts;
```

- `_odyssey_nfts` is not zero-checked before being used.

```
236         mom_token = _mom_token;
```

- `_mom_token` is not zero-checked before being used.

```
244         dad_token = _dad_token;
```

- `_dad_token` is not zero-checked before being used.

```
252         odyssey_nfts = _odyssey_nfts;
```

- `_odyssey_nfts` is not zero-checked before being used.

Recommendation

It is recommended to add a zero-check for the passed-in address value to prevent unexpected errors.

Alleviation

[Momentum Team, 06/08/2023]: The team heeded the advice to fix the issue in `Staking` contract and changes were made in commit 6fc42dd5ee8538436f14146d50912feb627edc98.

STK-08 | UNCHECKED ERC-20 `transfer()` / `transferFrom()` CALL

Category	Severity	Location	Status
Volatile Code	Minor	contracts/staking/Staking.sol (938a1bdb7d02c98cc2afd097cb0d9981eb58499d): 581, 584	Resolved

Description

The return values of the `transfer()` and `transferFrom()` calls in the smart contract are not checked. Some ERC-20 tokens' transfer functions return no values, while others return a bool value, they should be handled with care. If a function returns `false` instead of reverting upon failure, an unchecked failed transfer could be mistakenly considered successful in the contract.

```
581                 IERC20(mom_token).transfer(payable(msg.sender), moms_to_claim);
```

```
584                 IERC20(dad_token).transfer(payable(msg.sender), dads_to_claim);
```

Recommendation

It is advised to use the OpenZeppelin's `SafeERC20.sol` implementation to interact with the `transfer()` and `transferFrom()` functions of external ERC-20 tokens. The OpenZeppelin implementation checks for the existence of a return value and reverts if false is returned, making it compatible with all ERC-20 token implementations.

Alleviation

[Momentum Team, 06/08/2023]: The team heeded the advice to fix the issue in `Staking` contract and changes were made in commit 6fc42dd5ee8538436f14146d50912feb627edc98.

STK-15 | NO CHECK TO ENSURE `odyssey` EXISTS

Category	Severity	Location	Status
Logical Issue	Minor	contracts/staking/Staking.sol (938a1bdb7d02c98cc2afd097cb0d9981eb58499d): 321	Resolved

Description

During the staking process, there is currently no verification in place to ensure that the owner of the NFT has actually created the Odyssey. This means that users can stake by passing arbitrary IDs without any validation. However, according to the documentation, only the holder of the Odyssey NFT can create an odyssey, and supporters can create connections by staking.

This is particularly important as there is a function `_claim_rewards`, which verifies whether `msg.sender` is the `owner` of the `odyssey_id` by checking `IERC721(odyssey_nfts).ownerOf(odyssey_id) == msg.sender`. The `tokenId` for the `OdysseyNFT` begins at 1 during the minting process. However, despite this constraint, users are still able to stake to odysseyId 0.

Recommendation

It is recommended to check that an odyssey exists.

Alleviation

[Momentum Team, 06/08/2023]: The team heeded the advice to fix the issue in `Staking` contract and changes were made in commit 6fc42dd5ee8538436f14146d50912feb627edc98.

[CertiK, 06/12/2023]: Unstaking requires the associated NFT to exist, but since NFTs can be burned, users' deposits will be locked in the contract if they do not unstake prior to burning the NFT.

[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.

STK-20 POTENTIAL RISKS ASSOCIATED WITH CHANGING STAKING TOKENS

Category	Severity	Location	Status
Logical Issue	Minor	contracts/staking/Staking.sol (938a1bdb7d02c98cc2afd097cb0d9981eb58499d): 235	Resolved

Description

In the context of Staking, there is a possibility to change the addresses of the mom and dad tokens. However, this can pose a problem for users who have already staked their mom/dad tokens. The concern arises because if the token address is altered, there is a risk that the contract may not possess a sufficient number of tokens when users decide to unstake their holdings.

Recommendation

We recommend the team to prioritize transparency and provide clear communication to users when changing staking token addresses to mitigate potential risks.

Alleviation

[Momentum Team, 06/08/2023]: The team removed the related functions which were used to change `MOM` , `DAD` and `NFT` tokens in `Staking` contract and changes were made in commit 6fc42dd5ee8538436f14146d50912feb627edc98.

CON-02 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Coding Style	Informational	contracts/nft/OdysseyNFT.sol (938a1bdb7d02c98cc2afd097cb0d9981eb58499d): 87, 103, 151; contracts/staking/Staking.sol (938a1bdb7d02c98cc2afd097cb0d9981eb58499d): 235, 243, 251, 288, 296	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, 06/08/2023]: The team heeded the advice to emit events properly and changes were made in commit 6fc42dd5ee8538436f14146d50912feb627edc98.

STK-09 | MISSING ERROR MESSAGES

Category	Severity	Location	Status
Coding Style	Informational	contracts/staking/Staking.sol (938a1bdb7d02c98cc2afd097cb0d9981eb58499d): 375, 377, 457, 458	Resolved

Description

The **require** can be used to check for conditions and throw an exception if the condition is not met. It is better to provide a string message containing details about the error that will be passed back to the caller.

Recommendation

We advise adding error messages to the linked **require** statements.

Alleviation

[Momentum Team, 06/08/2023]: The team heeded the advice to add missing error messages in `Staking` contract and changes were made in commit 6fc42dd5ee8538436f14146d50912feb627edc98.

STK-13 | LACK OF USAGE OF `effective_timestamp`

Category	Severity	Location	Status
Logical Issue	Informational	contracts/staking/Staking.sol (938a1bdb7d02c98cc2afd097cb0d9981eb58499d): 668	Resolved

Description

The `Staking` contract has a function named `calculate_effective_timestamp()` which is responsible for calculating the new `effective_timestamp`. Whenever a user stakes multiple times or unstakes tokens , the `effective_timestamp` is recalculated and updated based on the last staking timestamp. However, it is unclear where the `effective_timestamp` is used within the contract.

It is likely that the `effective_timestamp` is used to calculate the rewards that each user is eligible to receive, but it is not stated as to what impact this value has.

Recommendation

It is recommended to remove unused functionality or create a use case for them.

Alleviation

[Momentum Team, 06/08/2023]: The `effective_timestamp` is used on the backend code to calculate the correct rewards, it's not used currently on the contract.

OPTIMIZATIONS | MOMENTUM

ID	Title	Category	Severity	Status
CON-03	Inefficient Memory Parameter	Logical Issue	Optimization	Resolved
ONF-01	State Variable Should Be Declared Constant	Gas Optimization	Optimization	Resolved
STK-12	Missing Input Validation	Logical Issue	Optimization	Resolved

CON-03 | INEFFICIENT MEMORY PARAMETER

Category	Severity	Location	Status
Logical Issue	Optimization	contracts/nft/OdysseyNFT.sol (938a1bdb7d02c98cc2afd097cb0d9981eb58499d): 151; contracts/staking/Staking.sol (938a1bdb7d02c98cc2afd097cb0d9981eb58499d): 263, 263, 263, 263	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.

```
151     function setbaseURI(string memory baseURI) public onlyOwner {
```

`setbaseURI` has memory location parameters: `baseURI`.

```
263     function update_rewards(address[] memory addresses, uint256[] memory
 stakers_amounts, uint256[] memory odysseys_ids, uint256[] memory odysseys_amounts,
uint timestamp ) public onlyRole(MANAGER_ROLE) {
```

`update_rewards` has memory location parameters: `addresses`, `stakers_amounts`, `odysseys_ids`, `odysseys_amounts`.

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, 06/08/2023]: The team heeded the advice to optimize the code and changes were made in commit 6fc42dd5ee8538436f14146d50912feb627edc98.

ONF-01 | STATE VARIABLE SHOULD BE DECLARED CONSTANT

Category	Severity	Location	Status
Gas Optimization	Optimization	contracts/nft/OdysseyNFT.sol (938a1bdb7d02c98cc2afd097cb0d9981eb58499d): 31	Resolved

Description

State variables that never change should be declared as `constant` to save gas.

```
31      uint256 private _UUIDMask = 0x8000800000000000000;
```

- `_UUIDMask` should be declared `constant`.

Recommendation

We recommend adding the `constant` attribute to state variables that never change.

Alleviation

[Momentum Team, 06/08/2023]: The team heeded the advice to optimize the code and changes were made in commit 6fc42dd5ee8538436f14146d50912feb627edc98.

STK-12 | MISSING INPUT VALIDATION

Category	Severity	Location	Status
Logical Issue	Optimization	contracts/staking/Staking.sol (938a1bdb7d02c98cc2afd097cb0d9981eb58499d): 321, 330, 341	Resolved

Description

In the `Staking` function, users can restake `MOM` or `DAD` token from one `Odyssey` to other `Odyssey`. However, there is no check to ensure `from_odyssey_id` is not equal to `to_odyssey_id`.

```
341     function restake(uint256 from_odyssey_id, uint256 to_odyssey_id, uint256
amount, Token token) public {
342         _restake(from_odyssey_id, to_odyssey_id, amount, token);
343     }
```

In addition, both `from_odyssey_id` and `to_odyssey_id` are not checked whether they are bigger than zero. According to the logic of `safeMint()` in `OdysseyNFT` contract, the `token_id` should start from `604472133179351442128897`. The same issue occurs in functions `stake()` and `unstake()`.

Recommendation

We recommend to ensure `from_odyssey_id` is different from `to_odyssey_id` to save some gas and check that `to_odyssey_id` is greater than zero in the functions aforementioned.

Alleviation

[Momentum Team, 06/08/2023]: The team added proper validations on `odyssey_id` in `Staking` contract and changes were made in commit 6fc42dd5ee8538436f14146d50912feb627edc98.

[CertiK, 06/12/2023]: The change requires `from_odyssey_id` to exist, which may not be true as NFTs can be burned. This can prevent users from restaking if the staked in an NFT that was later burned.

FORMAL VERIFICATION | MOMENTUM

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

Property Name	Title
erc721pausable-transferfrom-revert-pause	`transferFrom` Fails when Paused
erc721pausable-getapproved-change-state	`getApproved` Does Not Change the Contract's State
erc721pausable-isapprovedforall-succeed-normal	`isApprovedForAll` Always Succeeds
erc721pausable-isapprovedforall-correct	`isApprovedForAll` Returns Correct Approvals
erc721pausable-isapprovedforall-change-state	`isApprovedForAll` Does Not Change the Contract's State
erc721pausable-approve-succeed-normal	`approve` Returns for Admissible Inputs
erc721pausable-approve-set-correct	`approve` Sets Approval
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-multiple	`setApprovalForAll` Can Set Multiple Operators
erc721pausable-setapprovalforall-set-correct	`setApprovalForAll` Approves Operator
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-state-balance	`transferFrom` Keeps Balances Constant Except for From and To
erc721pausable-transferfrom-correct-balance	`transferFrom` Sum of Balances is Constant
erc721pausable-transferfrom-correct-state-owner	`transferFrom` Has Expected Ownership Changes

Property Name	Title
erc721pausable-transferfrom-correct-state-approval	`transferFrom` Has Expected Approval Changes
erc721pausable-transferfrom-revert-from-zero	`transferFrom` Fails for Transfers From the Zero Address
erc721pausable-transferfrom-revert-invalid	`transferFrom` Fails for Invalid Tokens
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-transferfrom-revert-not-owned	`transferFrom` Fails if `From` Is Not Token Owner
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-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-revert-zero	`transfer` Prevents Transfers to the Zero Address
erc20-transfer-succeed-normal	`transfer` Succeeds on Admissible Non-self Transfers
erc20-transfer-succeed-self	`transfer` Succeeds on Admissible Self Transfers
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-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-exceed-balance	`transfer` Fails if Requested Amount Exceeds Available Balance
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-correct-allowance	`transferFrom` Updated the Allowance Correctly
erc20-transferfrom-change-state	`transferFrom` Has No Unexpected State Changes
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-false	If `transferFrom` Returns `false`, the Contract's State Is Unchanged
erc20-transferfrom-never-return-false	`transferFrom` Never Returns `false`
erc20-totalsupply-succeed-always	`totalSupply` Always Succeeds
erc20-totalsupply-correct-value	`totalSupply` Returns the Value of the Corresponding State Variable
erc20-balanceof-succeed-always	`balanceOf` Always Succeeds
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-balanceof-change-state	`balanceOf` Does Not Change the Contract's State
erc20-allowance-correct-value	`allowance` Returns Correct Value
erc20-allowance-succeed-always	`allowance` Always Succeeds
erc20-allowance-change-state	`allowance` Does Not Change the Contract's State
erc20-approve-revert-zero	`approve` Prevents Approvals For the Zero Address
erc20-approve-correct-amount	`approve` Updates the Approval Mapping Correctly
erc20-approve-succeed-normal	`approve` Succeeds for Admissible Inputs
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 6fc42dd5ee8538436f14146d50912feb627edc98

Verification of Compliance with Pausable ERC-721

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 `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 `transferFrom`

Property Name	Final Result	Remarks
erc721pausable-transferfrom-succeed-normal	True	
erc721pausable-transferfrom-revert-pause	False	
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-state-balance	True	
erc721pausable-transferfrom-correct-balance	True	
erc721pausable-transferfrom-correct-state-owner	True	
erc721pausable-transferfrom-correct-state-approval	True	
erc721pausable-transferfrom-revert-from-zero	True	
erc721pausable-transferfrom-revert-invalid	True	
erc721pausable-transferfrom-revert-to-zero	True	
erc721pausable-transferfrom-revert-not-owned	True	
erc721pausable-transferfrom-revert-exceed-approval	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 `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-succeed-normal	True	
erc721pausable-approve-set-correct	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-multiple	True	
erc721pausable-setapprovalforall-set-correct	True	
erc721pausable-setapprovalforall-change-state	True	

Detailed Results For Contract DADToken (contracts/token/DADToken.sol) In Commit 6fc42dd5ee8538436f14146d50912feb627edc98

Verification of ERC-20 Compliance

Detailed results for function `transfer`

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

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	True	
erc20-approve-correct-amount	True	
erc20-approve-succeed-normal	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 6fc42dd5ee8538436f14146d50912feb627edc98

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-self	False	
erc20-transfer-succeed-normal	False	
erc20-transfer-correct-amount-self	True	
erc20-transfer-change-state	True	
erc20-transfer-exceed-balance	True	
erc20-transfer-false	True	
erc20-transfer-never-return-false	True	
erc20-transfer-recipient-overflow	False	

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	True	
erc20-transferfrom-correct-amount-self	True	
erc20-transferfrom-succeed-normal	False	
erc20-transferfrom-succeed-self	False	
erc20-transferfrom-correct-allowance	True	
erc20-transferfrom-fail-exceed-balance	True	
erc20-transferfrom-change-state	True	
erc20-transferfrom-fail-exceed-allowance	True	
erc20-transferfrom-false	True	
erc20-transferfrom-never-return-false	True	
erc20-transferfrom-fail-recipient-overflow	False	

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 OdysseyNFT (contracts/nft/OdysseyNFT.sol) In Commit 938a1bdb7d02c98cc2afd097cb0d9981eb58499d

Verification of Compliance with Pausable ERC-721

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 `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 `transferFrom`

Property Name	Final Result	Remarks
erc721pausable-transferfrom-succeed-normal	True	
erc721pausable-transferfrom-revert-pause	False	
erc721pausable-transferfrom-correct-increase	True	
erc721pausable-transferfrom-correct-one-token-self	True	
erc721pausable-transferfrom-correct-approval	True	
erc721pausable-transferfrom-correct-owner-from	True	
erc721pausable-transferfrom-correct-owner-to	True	
erc721pausable-transferfrom-correct-state-balance	True	
erc721pausable-transferfrom-correct-balance	True	
erc721pausable-transferfrom-correct-state-owner	True	
erc721pausable-transferfrom-correct-state-approval	True	
erc721pausable-transferfrom-revert-invalid	True	
erc721pausable-transferfrom-revert-from-zero	True	
erc721pausable-transferfrom-revert-to-zero	True	
erc721pausable-transferfrom-revert-not-owned	True	
erc721pausable-transferfrom-revert-exceed-approval	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 `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-succeed-normal	True	
erc721pausable-approve-set-correct	True	
erc721pausable-approve-revert-not-allowed	True	
erc721pausable-approve-revert-invalid-token	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 938a1bdb7d02c98cc2afd097cb0d9981eb58499d

Verification of ERC-20 Compliance

Detailed results for function `transfer`

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

Detailed results for function `totalSupply`

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

Detailed results for function `balanceOf`

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

Detailed results for function `allowance`

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

Detailed Results For Contract MOMToken (contracts/token/MomToken.sol) In Commit 938a1bdb7d02c98cc2afd097cb0d9981eb58499d

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-self	False	
erc20-transfer-succeed-normal	False	
erc20-transfer-correct-amount-self	True	
erc20-transfer-change-state	True	
erc20-transfer-exceed-balance	True	
erc20-transfer-false	True	
erc20-transfer-never-return-false	True	
erc20-transfer-recipient-overflow	False	

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	
erc20-transferfrom-succeed-self	False	
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	

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-correct-value	True	
erc20-allowance-succeed-always	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

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