



Moonshot Magax

Security Assessment

CertiK Assessed on Aug 12th, 2025





CertiK Assessed on Aug 12th, 2025

Moonshot Magax

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

Executive Summary

TYPES

Vesting

ECOSYSTEM

Polygon (MATIC)

METHODS

Formal Verification, Manual Review, Static Analysis

LANGUAGE

Solidity

TIMELINE

Delivered on 08/12/2025

KEY COMPONENTS

N/A

CODEBASE

[base](#)[update_20250812](#)[View All in Codebase Page](#)

COMMITTS

[0x777fd819dc63418c648c1b9437d0f8d8211b3c08](#)[a147929194c6c93b93fdeee82e39b8925787ba93](#)[View All in Codebase Page](#)

Vulnerability Summary



5

Total Findings

4

Resolved

1

Timelock

0

Partially Resolved

0

Acknowledged

0

Declined



1 Centralization

1 Timelock



Centralization findings highlight privileged roles & functions and their capabilities, or instances where the project takes custody of users' assets.



0 Critical

Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.



0 Major

Major risks may include logical errors that, under specific circumstances, could result in fund losses or loss of project control.



0 Medium

Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.



1 Minor

1 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 | MOONSHOT MAGAX

Summary

[Executive Summary](#)

[Vulnerability Summary](#)

[Codebase](#)

[Audit Scope](#)

[Approach & Methods](#)

Findings

[MOM-04 : Centralization Risks](#)

[MOM-05 : Missing Finalisation Check in Stage Activation Function](#)

[MOM-06 : Too many digits](#)

[MOM-07 : Missing Emit Events](#)

[MOM-08 : Misleading Error Usage in `fallback\(\)` Function](#)

Optimizations

[MOM-01 : User-Defined Getters](#)

[MOM-02 : Redundant Unused Constant `DEFAULT_PROMO_BONUS_BPS`](#)

[MOM-03 : Redundant Emergency Withdrawal Function Due to Disabled Ether Reception](#)

Formal Verification

[Considered Functions And Scope](#)

[Verification Results](#)

Appendix

Disclaimer

CODEBASE | MOONSHOT MAGAX

Repository

base

update_20250812

Commit





0x777fd819dc63418c648c1b9437d0f8d8211b3c08

a147929194c6c93b93fdeee82e39b8925787ba93

AUDIT SCOPE | MOONSHOT MAGAX

4 files audited ● 1 file with Resolved findings ● 3 files without findings



ID	Repo	File	SHA256 Checksum
● PSO	amoy	 contracts/PreSaleOnChain.sol	6bd2bf6153014d9b94dab6ae1b22102f bab0d7214b85dacdaa5a4921c4c2c903
● PSC	moonShotMAGAX1/magax	 contracts/PreSaleOnChain.sol	b7eedfcd6e51cdb959c8f688caa53b2fd c9ae1626f316442f0cf816a3e75ff71
● PSS	moonShotMAGAX1/magax	 contracts/PreSaleOnChain.sol	dc4767f7fe757a5b6cf3205f3d4a91aa63 b277a02f4e1e78861bb341cbdf0f31
● PSM	moonShotMAGAX1/magax	 contracts/PreSaleOnChain.sol	8e4eb14f3cfc54a476fc043744cfc52c63 a2a83fb0550ffe238be0ae7c6a2d4f

APPROACH & METHODS | MOONSHOT MAGAX

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

FINDINGS | MOONSHOT MAGAX



5

Total Findings

0

Critical

1

Centralization

0

Major

0

Medium

1

Minor

3

Informational

This report has been prepared to discover issues and vulnerabilities for Moonshot Magax. Through this audit, we have uncovered 5 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
MOM-04	Centralization Risks	Centralization	Centralization	● 48h Timelock
MOM-05	Missing Finalisation Check In Stage Activation Function	Logical Issue	Minor	● Resolved
MOM-06	Too Many Digits	Magic Numbers	Informational	● Resolved
MOM-07	Missing Emit Events	Coding Style	Informational	● Resolved
MOM-08	Misleading Error Usage In <code>fallback()</code> Function	Logical Issue	Informational	● Resolved

MOM-04 | CENTRALIZATION RISKS

Category	Severity	Location	Status
Centralization	● Centralization	contracts/PreSaleOnChain.sol (base): 161, 277, 472, 497, 554, 558, 562, 568, 574, 586, 646	● 48h Timelock

Description

In the contract `MAGAXPresaleReceipts`, the roles `DEFAULT_ADMIN_ROLE` and `RECORDER_ROLE` have authority over the functions shown below, creating potential centralization risks:

1. **recordPurchase()**: Controlled by the `RECORDER_ROLE`, allowing potential manipulation of purchase records without additional oversight.
2. **recordPurchaseWithReferral()**: Also controlled by the `RECORDER_ROLE`, which could lead to biased referral purchases.
3. **configureStage()**: Managed by the `DEFAULT_ADMIN_ROLE`, providing the admin with unchecked control over stage configuration, including token pricing and allocation.
4. **activateStage()**: Controlled by the `DEFAULT_ADMIN_ROLE`, allowing for manipulation of token sale stages.
5. **finalise()**: The `DEFAULT_ADMIN_ROLE` has the authority to finalize the presale.
6. **setMaxPromoBps()**: Controlled by the `DEFAULT_ADMIN_ROLE`, giving the admin the ability to adjust promotional bonus limits.
7. **emergencyTokenWithdraw()**: The `DEFAULT_ADMIN_ROLE` has exclusive access to withdraw tokens from the contract, posing risks if abused.
8. **emergencyEthWithdraw()**: Controlled by the `DEFAULT_ADMIN_ROLE`, enabling the admin to withdraw ETH from the contract.
9. **recordPurchaseWithPromo()**: Managed by the `RECORDER_ROLE`, which could lead to biased bonus distributions if not properly monitored.

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 (2/3, 3/5) 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.

I Alleviation

[Moonshot Magax, 08/11/2025]: The team heeded the advice and resolved the issue by adding timelock in the commit.

MOM-05 | MISSING FINALISATION CHECK IN STAGE ACTIVATION FUNCTION

Category	Severity	Location	Status
Logical Issue	Minor	contracts/PreSaleOnChain.sol (base): 497-514	Resolved

Description

The `activateStage()` function lacks a check for the `finalised` flag, allowing an administrator to activate new presale stages even after the presale has been finalized. This contradicts the intended logic enforced elsewhere in the contract, where purchase related actions are blocked once `finalised` is set to `true`. Without this restriction, a finalized presale could appear reactivated, potentially leading to inconsistent state assumptions or misuse by interfaces or off chain systems relying on the finalization state.

Recommendation

We recommend adding a `require(!finalised, ...)` check to the `activateStage()` function to prevent stage activation after the presale has been finalized.

Alleviation

[Certik, 08/07/2025]: A check was added to revert the transaction with a `PresaleFinalised` error if the presale has already been finalized. The change has been reflected in [the commit](#).

MOM-06 | TOO MANY DIGITS

Category	Severity	Location	Status
Magic Numbers	● Informational	contracts/PreSaleOnChain.sol (base): 28-761	● Resolved

Description

Literals with many digits are difficult to read and review.

```
35      uint128 public constant MAX_TOTAL_USDT = 10000000 * 1e6;  
// 10M USDT total presale limit
```

```
34      uint128 public constant MAX_PURCHASE_USDT = 1000000 * 1e6;  
// 1M USDT max per purchase
```

Recommendation

We advise to use underscores `_` in numeric literals to improve readability.

Alleviation

[CertiK, 08/07/2025]: Numeric literals in the code were reformatted using underscore separators for better readability, such as changing `10000000` to `1_000_000`, without altering their actual values. The change has been reflected in [the commit](#).

MOM-07 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Coding Style	● Informational	contracts/PreSaleOnChain.sol (base): 568	● Resolved

Description

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

Recommendation

It is recommended to emit events in sensitive functions that are controlled by centralization roles.

Alleviation

[CertiK, 08/07/2025]: An event emission was added to log changes to the maximum promo basis points by emitting `MaxPromoBpsUpdated()`. The change has been reflected in [the commit](#).

MOM-08 | MISLEADING ERROR USAGE IN `fallback()` FUNCTION

Category	Severity	Location	Status
Logical Issue	● Informational	contracts/PreSaleOnChain.sol (base): 604~606	● Resolved

Description

Using the same custom error `EthNotAccepted()` for both the `receive()` and `fallback()` functions may lead to confusion during debugging or contract interaction, as these functions serve distinct purposes. While `receive()` handles plain Ether transfers, `fallback()` is triggered when calldata is present or no matching function is found. Reverting with `EthNotAccepted()` in `fallback()` suggests the issue is Ether related, when in fact the call might be an unsupported function invocation. A more appropriate error like `FallbackNotAllowed()` would provide clearer intent and better separation of concerns.

Recommendation

We recommend replacing the `EthNotAccepted()` error in the `fallback()` function with a more descriptive error such as `FallbackNotAllowed()` to accurately convey the reason for the revert.

Alleviation

[Certik, 08/07/2025]: The error `EthNotAccepted` was replaced with `FallbackNotAllowed`. The change has been reflected in [the commit](#).

OPTIMIZATIONS | MOONSHOT MAGAX

ID	Title	Category	Severity	Status
<u>MOM-01</u>	User-Defined Getters	Gas Optimization	Optimization	● Resolved
<u>MOM-02</u>	Redundant Unused Constant <code>DEFAULT_PROMO_BONUS_BPS</code>	Code Optimization	Optimization	● Resolved
<u>MOM-03</u>	Redundant Emergency Withdrawal Function Due To Disabled Ether Reception	Code Optimization	Optimization	● Resolved

MOM-01 | USER-DEFINED GETTERS

Category	Severity	Location	Status
Gas Optimization	● Optimization	contracts/PreSaleOnChain.sol (base): 414~416, 624~626	● Resolved

Description

The functions `getReceipts()` and `getUserReferrer()` are redundant as they simply return the values stored in the `userReceipts` and `userReferrer` mappings, respectively, without adding any additional logic. In fact, these functions are effectively the same as the automatically generated getter functions for public state variables, which already allow direct access to the mappings. This results in unnecessary gas costs for function calls when the mappings can be accessed directly, thus making these functions redundant and inefficient.

Recommendation

We recommend removing the redundant getter functions and accessing the `userReceipts` and `userReferrer` mappings directly to reduce unnecessary gas costs.

Alleviation

[CertiK, 08/07/2025]: The functions `getReceipts()` and `getUserReferrer()` were removed. The change has been reflected in [the commit](#).

MOM-02 | REDUNDANT UNUSED CONSTANT

DEFAULT_PROMO_BONUS_BPS

Category	Severity	Location	Status
Code Optimization	● Optimization	contracts/PreSaleOnChain.sol (base): 44	● Resolved

Description

The constant `DEFAULT_PROMO_BONUS_BPS` is declared but never used anywhere in the contract, making it redundant. This unused constant adds unnecessary clutter to the code and could potentially confuse developers or auditors, as it seems to be intended for a purpose that is not being utilized. Removing it would improve code clarity and reduce potential maintenance issues.

Recommendation

We recommend removing the unused constant `DEFAULT_PROMO_BONUS_BPS` to improve code clarity and reduce unnecessary clutter.

Alleviation

[CertiK, 08/07/2025]: The constant declaration `DEFAULT_PROMO_BONUS_BPS` was removed. The change has been reflected in [the commit](#).

MOM-03 | REDUNDANT EMERGENCY WITHDRAWAL FUNCTION DUE TO DISABLED ETHER RECEPTION

Category	Severity	Location	Status
Code Optimization	● Optimization	contracts/PreSaleOnChain.sol (base): 586~598	● Resolved

Description

The `emergencyEthWithdraw()` function is redundant since the contract explicitly rejects all incoming Ether transfers via the `receive()` function, which reverts any such attempts. With no other `payable` functions or mechanisms for receiving Ether, the contract's balance will always remain zero, rendering this withdrawal logic unreachable and unnecessary. Keeping unused emergency functions may increase code complexity and could mislead maintainers about potential fund handling capabilities.

Recommendation

We recommend removing the `emergencyEthWithdraw()` function to reduce code complexity and avoid confusion, as the contract does not accept Ether under any circumstances.

Alleviation

[Certik, 08/07/2025]: The function `emergencyEthWithdraw()` and its corresponding event declaration `EmergencyEthWithdraw()` were removed. The change has been reflected in [the commit](#).

FORMAL VERIFICATION | MOONSHOT MAGAX

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

Considered Functions And Scope

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

Verification of contracts derived from AccessControl v4.4

We verified properties of the public interface of contracts that provide an AccessControl-v4.4 compatible API. This involves:

- The `hasRole` function, which returns `true` if an account has been granted a specific `role`.
- The `getRoleAdmin` function, which returns the admin role that controls a specific `role`.
- The `grantRole` and `revokeRole` functions, which are used for granting a `role` to an account and revoking a `role` from an `account`, respectively.
- The `renounceRole` function, which allows the calling account to revoke a `role` from itself.

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

Property Name	Title
accesscontrol-grantrole-correct-role-granting	<code>grantRole</code> Correctly Grants Role
accesscontrol-revokerole-correct-role-revoking	<code>revokeRole</code> Correctly Revokes Role
accesscontrol-hasrole-change-state	<code>hasRole</code> Function Does Not Change State
accesscontrol-default-admin-role	AccessControl Default Admin Role Invariance
accesscontrol-renouncerole-revert-not-sender	<code>renounceRole</code> Reverts When Caller Is Not the Confirmation Address
accesscontrol-hasrole-succeed-always	<code>hasRole</code> Function Always Succeeds
accesscontrol-getroleadmin-succeed-always	<code>getRoleAdmin</code> Function Always Succeeds
accesscontrol-renouncerole-succeed-role-renouncing	<code>renounceRole</code> Successfully Renounces Role
accesscontrol-getroleadmin-change-state	<code>getRoleAdmin</code> Function Does Not Change State

Verification Results

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

Detailed Results For Contract MAGAXPresaleReceipts (contracts/PreSaleOnChain.sol) In Commit a147929194c6c93b93fdeee82e39b8925787ba93

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	● True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	● True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-change-state	● True	
accesscontrol-hasrole-succeed-always	● True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	● True	

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-revert-not-sender	● True	
accesscontrol-renouncerole-succeed-role-renouncing	● True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	● True	
accesscontrol-getroleadmin-change-state	● True	

Detailed Results For Contract MAGAXPresaleReceipts (contracts/PreSaleOnChain.sol) In Commit e11f1a4907ab28d56a97d3ec9c5069678f8647d4

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-revert-not-sender	● True	
accesscontrol-renouncerole-succeed-role-renouncing	● True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	● True	
accesscontrol-getroleadmin-change-state	● True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-succeed-always	● True	
accesscontrol-hasrole-change-state	● True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	● True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	● True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	● True	

Detailed Results For Contract MAGAXPresaleReceipts (contracts/PreSaleOnChain.sol) In Commit 6a4d6dcece7b7120b2737137c591f3a88a84f7ba**Verification of contracts derived from AccessControl v4.4**Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	● True	
accesscontrol-getroleadmin-change-state	● True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-succeed-always	● True	
accesscontrol-hasrole-change-state	● True	

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-revert-not-sender	● True	
accesscontrol-renouncerole-succeed-role-renouncing	● True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	● True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	● True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	● True	

Detailed Results For Contract MAGAXPresaleReceipts (contracts/PreSaleOnChain.sol) In Commit 0x777fd819dc63418c648c1b9437d0f8d8211b3c08

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-succeed-role-renouncing	● True	
accesscontrol-renouncerole-revert-not-sender	● True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	● True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-change-state	● True	
accesscontrol-getroleadmin-succeed-always	● True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	● True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-change-state	● True	
accesscontrol-hasrole-succeed-always	● True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	● True	

APPENDIX | MOONSHOT MAGAX

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.
Magic Numbers	Magic Number findings refer to numeric literals that are expressed in the code in their raw format, but should instead be declared as constants to improve readability and maintainability.
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.

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

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

The following assumptions and simplifications apply to our model:

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

Formalism for property specifications

All properties are expressed in a behavioral interface specification language that CertiK has developed for Solidity, which allows us to specify the behavior of each function in terms of the contract state and its parameters and return values, as well as contract properties that are maintained by every observable state transition. Observable state transitions occur when the contract's external interface is invoked and the invocation does not revert, and when the contract's Ether balance is changed

by the EVM due to another contract's "self-destruct" invocation. The specification language has the usual Boolean connectives, as well as the operator `\old` (used to denote the state of a variable before a state transition), and several types of specification clause:

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

- `requires [cond]` - the condition `cond`, which refers to a function's parameters, return values, and contract state variables, must hold when a function is invoked in order for it to exhibit a specified behavior.
- `ensures [cond]` - the condition `cond`, which refers to a function's parameters, return values, and both `\old` and current contract state variables, is guaranteed to hold when a function returns if the corresponding requires condition held when it was invoked.
- `invariant [cond]` - the condition `cond`, which refers only to contract state variables, is guaranteed to hold at every observable contract state.
- `constraint [cond]` - the condition `cond`, which refers to both `\old` and current contract state variables, is guaranteed to hold at every observable contract state except for the initial state after construction (because there is no previous state); constraints are used to restrict how contract state can change over time.

Description of the Analyzed AccessControl-v4.4 Properties

Properties related to function `grantRole`

accesscontrol-grantrole-correct-role-granting

After execution, `grantRole` must ensure the specified account has the granted role.

Specification:

```
ensures hasRole(role, account);
```

Properties related to function `revokeRole`

accesscontrol-revokeRole-correct-role-revoking

After execution, `revokeRole` must ensure the specified account no longer has the revoked role.

Specification:

```
ensures !hasRole(role, account);
```

Properties related to function `hasRole`

accesscontrol-hasrole-change-state

The `hasRole` function must not change any state variables.

Specification:

```
assignable \nothing;
```

accesscontrol-hasrole-succeed-always

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

Specification:

```
reverts_only_when false;
```

Properties related to function `DEFAULT_ADMIN_ROLE`

accesscontrol-default-admin-role

The default admin role must be invariant, ensuring consistent access control management.

Specification:

```
invariant DEFAULT_ADMIN_ROLE() == 0x00;
```

Properties related to function `renounceRole`

accesscontrol-renouncerole-revert-not-sender

The `renounceRole` function must revert if the caller is not the same as `account`.

Specification:

```
reverts_when account != msg.sender;
```

accesscontrol-renouncerole-succeed-role-renouncing

After execution, `renounceRole` must ensure the caller no longer has the renounced role.

Specification:

```
ensures !hasRole(role, account);
```

Properties related to function `getRoleAdmin`

accesscontrol-getroleadmin-change-state

The `getRoleAdmin` function must not change any state variables.

Specification:

```
assignable \nothing;
```

accesscontrol-getroleadmin-succeed-always

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

Specification:

```
reverts_only_when false;
```

DISCLAIMER | CERTIK

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

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

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

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

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

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

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

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

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

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

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

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

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

Elevating Your Entire **Web3** Journey

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.

