reach.io smart contract audit

Security Assessment

CertiK Assessed on Mar 21st, 2025

CertiK Assessed on Mar 21st, 2025

reach.io smart contract audit

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

Executive Summary

TYPES	ECOSYSTEM	METHODS
Others	Binance Smart Chain (BSC)	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 03/21/2025	N/A

CODEBASE
https://github.com/reachme-io/contracts/
View All in Codebase Page

COMMITS
9ef2f508538a619d90fdbb5cccd49f63711630bd
View All in Codebase Page

Highlighted Centralization Risks

- Privileged role can remove users' tokens
- Fees are bounded by 20%

Vulnerability Summary

4	3	0	0	1	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.
1	Major	1 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.
0	Medium		Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
2	Minor	2 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.
1	Informational	1 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 | REACH.IO SMART CONTRACT AUDIT

CODEBASE | REACH.IO SMART CONTRACT AUDIT

Repository

https://github.com/reachme-io/contracts/

Commit

9ef2f508538a619d90fdbb5cccd49f63711630bd

AUDIT SCOPE | REACH.IO SMART CONTRACT AUDIT

2 files audited ● 2 files with Acknowledged findings

ID	Repo	File	SHA256 Checksum
● AUT	reachme-io/contracts	contracts/Authority.sol	a152a78eee8910abc7cbbbfda673035caaa3eca72dd07bab2ede118fe42ea23e
● REA	reachme-io/contracts	contracts/Reach.sol	eeb945da4ae450aa2cfd1957147b390e8a654375640b6c3f2f70c87c8e56673c

APPROACH & METHODS | REACH.IO SMART CONTRACT AUDIT

This report has been prepared for reach.io to discover issues and vulnerabilities in the source code of the reach.io smart contract audit 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, Formal Verification, 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 | REACH.IO SMART CONTRACT AUDIT

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

This report has been prepared to discover issues and vulnerabilities for reach.io smart contract audit. Through this audit, we have uncovered 4 issues ranging from different severity levels. Utilizing the techniques of Static Analysis, Formal Verification & Manual Review to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
REA-02	**Centralization Related Risks**	**Centralization**	**Major**	**Acknowledged**
REA-03	`forceRefund()` Can Be Called By Any Account Instead Of Only The Depositor	Logical Issue	Minor	Resolved
REA-04	Lack Of Duplication Check On The Deposit Identifiers	Logical Issue	Minor	Resolved
CON-01	Inconsistent Solidity Versions	Language Version	Informational	Resolved

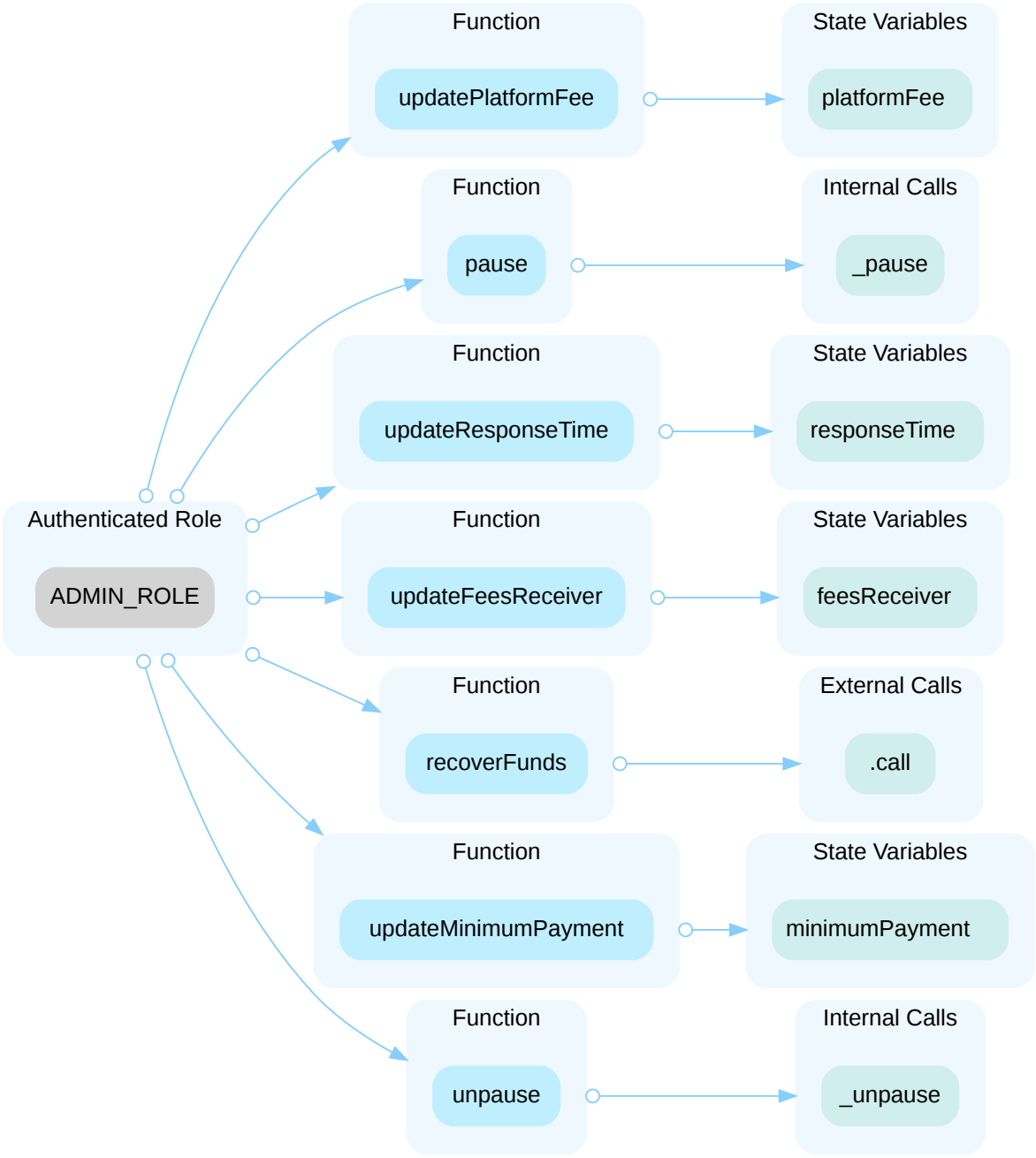
REA-02 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization	**Major**	**contracts/Authority.sol: 289; contracts/Reach.sol: 95, 144, 171, 196, 205, 213, 222, 254, 258, 289**	**Acknowledged**

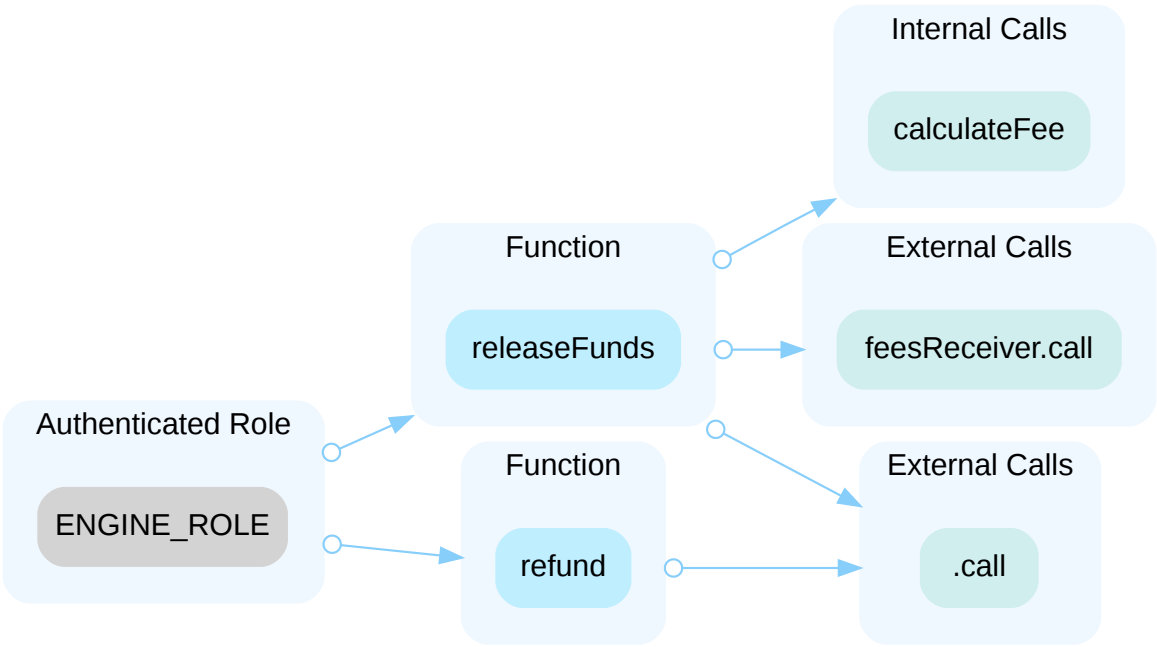
Description

In the contract `Reach`, the role `ADMIN_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `ADMIN_ROLE` account may allow the hacker to take advantage of this authority and update the platform fee, pause contract execution, update the response time, update the fees receiver address, update the minimum payment amount, and **withdraw arbitrary amounts of escrowed funds from the contract.**

The `ADMIN_ROLE` can withdraw the contract's total balance through the function `recoverFunds`, at any time, with no further checks on whether those funds are associated with an active escrow deposit.



In the contract `Reach` , the role `ENGINE_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `ENGINE_ROLE` account may allow the hacker to take advantage of this authority and release (or not release) deposit funds or process a refund (or not process the refund) for a deposit.



In the contract `Authority`, the role `DEFAULT_ADMIN_ROLE` has authority to set the `ADMIN_ROLE` and `ENGINE_ROLE` used in the `Reach` contract. Any compromise to the `DEFAULT_ADMIN_ROLE` may compromise any privileged functions controlled by the `ADMIN_ROLE` and `ENGINE_ROLE`.

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

[reach.io Team, 03/21/2025]: We acknowledged this issue. The admin address is intended to be a multi sign wallet.

[CertiK, 03/21/2025]: It is suggested to implement the aforementioned methods to avoid centralized failure. Also, it strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

REA-03 `forceRefund()` CAN BE CALLED BY ANY ACCOUNT INSTEAD OF ONLY THE DEPOSITOR

Category	Severity	Location	Status
Logical Issue	Minor	contracts/Reach.sol: 266~287	Resolved

Description

The `forceRefund()` function is currently callable by anyone after the (MAX_RESPONSE_TIME + 4 hours) window has passed. Although it is designed to allow depositors to recover their escrowed funds if the engine role does not act, there is no restriction enforcing that only the original requester can trigger this forced refund.

```
266     function forceRefund(uint256 _depositId) external nonReentrant {
267         Deposit storage _deposit = deposits[_depositId];
268
269         if (_deposit.released || _deposit.refunded) revert AlreadyProcessed();
270         if (block.timestamp < _deposit.timestamp + MAX_RESPONSE_TIME + 14400)
271             // 4 hours after max response time
272             revert TimeWindowNotElapsed();
273
274         _deposit.refunded = true;
275
276         (bool success, ) = _deposit.requester.call{
277             value: _deposit.escrowAmount
278         }("");
279         require(success, "Refund transfer failed");
```

Allowing any account to trigger `forceRefund` does not necessarily cause direct harm or loss of funds, because the refund still goes back to the correct requester address. However, this does enable arbitrary third parties to spam or trigger forced refunds on a depositor's behalf.

Recommendation

It's recommended to modify the linked function and only allows the deposit requester to submit force refund request.

Alleviation

[reach.io Team, 03/21/2025]: The team heeded the advice and resolved the issue in commit 5a87b7c9e2a15a767d698d85f6121fb27cbac1ef.

Now only the original depositor can call `forceRefund()` if the `max elapsed time + 4 hours` is reached.

REA-04 LACK OF DUPLICATION CHECK ON THE DEPOSIT IDENTIFIERS

Category	Severity	Location	Status
Logical Issue	Minor	contracts/Reach.sol	Resolved

Description

In the `deposit()` function, an `_identifier` is used as a function argument. The function does not check whether the `_identifier` has been used already, so it is possible that different `depositId` have the same `_identifier`.

Recommendation

We'd like to understand the use case for the `_identifier` and whether they should be unique identifier for each `depositId`.

Alleviation

[reach.io Team, 03/21/2025]: The team heeded the advice and resolved the issue in commit 5a87b7c9e2a15a767d698d85f6121fb27cbac1ef.

The `_identifier` is used to link a payment to specific message on the database, before a payment is sent a request is sent to create the message, however the message is not yet ‘active’ or in our terms message.status = “unconfirmed”, when a payment event is detected with the `_identifier` and amount >= required price we mark it as the message as confirmed, message.status = “confirmed” and is now visible to the KOL.

Implemented a duplication check to ensure we don’t let them pay twice for the same message, however the edge case is if a user underpays somehow via a manual deposit then the message will never activate.

CON-01 | INCONSISTENT SOLIDITY VERSIONS

Category	Severity	Location	Status
Language Version	Informational	contracts/Authority.sol: 2; contracts/Reach.sol: 2	Resolved

Description

The codebase contains multiple Solidity versions, which can lead to unexpected behavior, potential vulnerabilities, difficulties in maintaining the code, and inconsistencies in the execution of the smart contract. Using different versions may also result in increased complexity during code auditing, as different security features and bug fixes are present in different versions of the compiler.

`^0.8.20` is used in contracts/Reach.sol file.

```
2  pragma solidity ^0.8.20;
```

`^0.8.23` is used in contracts/Authority.sol file.

```
2  pragma solidity ^0.8.23;
```

Recommendation

It is recommended to standardize on a single, up-to-date Solidity version throughout the codebase to ensure consistent behavior, benefit from the latest security features, and improve maintainability.

Alleviation

[reach.io Team, 03/21/2025]: The team heeded the advice and resolved the issue in commit 5a87b7c9e2a15a767d698d85f6121fb27cbac1ef.

OPTIMIZATIONS | REACH.IO SMART CONTRACT AUDIT

ID	Title	Category	Severity	Status
REA-01	State Variables That Could Be Declared Immutable	Coding Issue	Optimization	Acknowledged

REA-01 STATE VARIABLES THAT COULD BE DECLARED IMMUTABLE

Category	Severity	Location	Status
Coding Issue	Optimization	contracts/Reach.sol: 18	Acknowledged

Description

State variables that are not updated following deployment should be declared immutable to save gas.

```
18      ReachAuthority public authority;
```

Recommendation

Add the immutable attribute to state variables that never change or are set only in the constructor.

Alleviation

[reach.io, 03/21/2025]: The team acknowledged the issue and decided to remain unchanged in the scope of the audit.

FORMAL VERIFICATION | REACH.IO SMART CONTRACT AUDIT

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

Verification of Standard Pausable Properties

We verified *partial* properties of the public interfaces of those token contracts that implement the Pausable interface. This involves:

- function `paused` that returns the if the contract is paused,
- function `pause` that pauses the contract, and
- function `unpause` that unpauses the contract.

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

Property Name	Title
pausable-paused-succeed-normal	`paused` Function Always Succeeds
pausable-unpause-correct	Function `unpause` Always Unpauses
pausable-pause-correct	Function `pause` Always Pauses

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 ReachAuthority (contracts/Authority.sol) In Commit 9ef2f508538a619d90fdbb5cccd49f63711630bd

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 `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-succeed-role-renouncing	True	
accesscontrol-renouncerole-revert-not-sender	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-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 Contract Reach (contracts/Reach.sol) In Commit 9ef2f508538a619d90fdbb5cccd49f63711630bd

Verification of Standard Pausable Properties

Detailed Results for Function `paused`

Property Name	Final Result	Remarks
pausable-paused-succeed-normal	True	

Detailed Results for Function `unpause`

Property Name	Final Result	Remarks
pausable-unpause-correct	True	

Detailed Results for Function `pause`

Property Name	Final Result	Remarks
pausable-pause-correct	True	

APPENDIX | REACH.IO SMART CONTRACT AUDIT

Finding Categories

Categories	Description
Language Version	Language Version findings indicate that the code uses certain compiler versions or language features with known security issues.
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
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 `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;
```

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;
```

Description of the Analyzed Standard Pausable Properties Properties

Properties related to function `paused`

pausable-paused-succeed-normal

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

Specification:

```
reverts_only_when false;
```

Properties related to function `unpause`

pausable-unpause-correct

All non-reverting invocations of `unpause()` must unpause the contract.

Specification:

```
ensures !this.paused();
```

Properties related to function `pause`

pausable-pause-correct

All non-reverting invocations of `pause()` must pause the contract.

Specification:

```
ensures this.paused();
```

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.