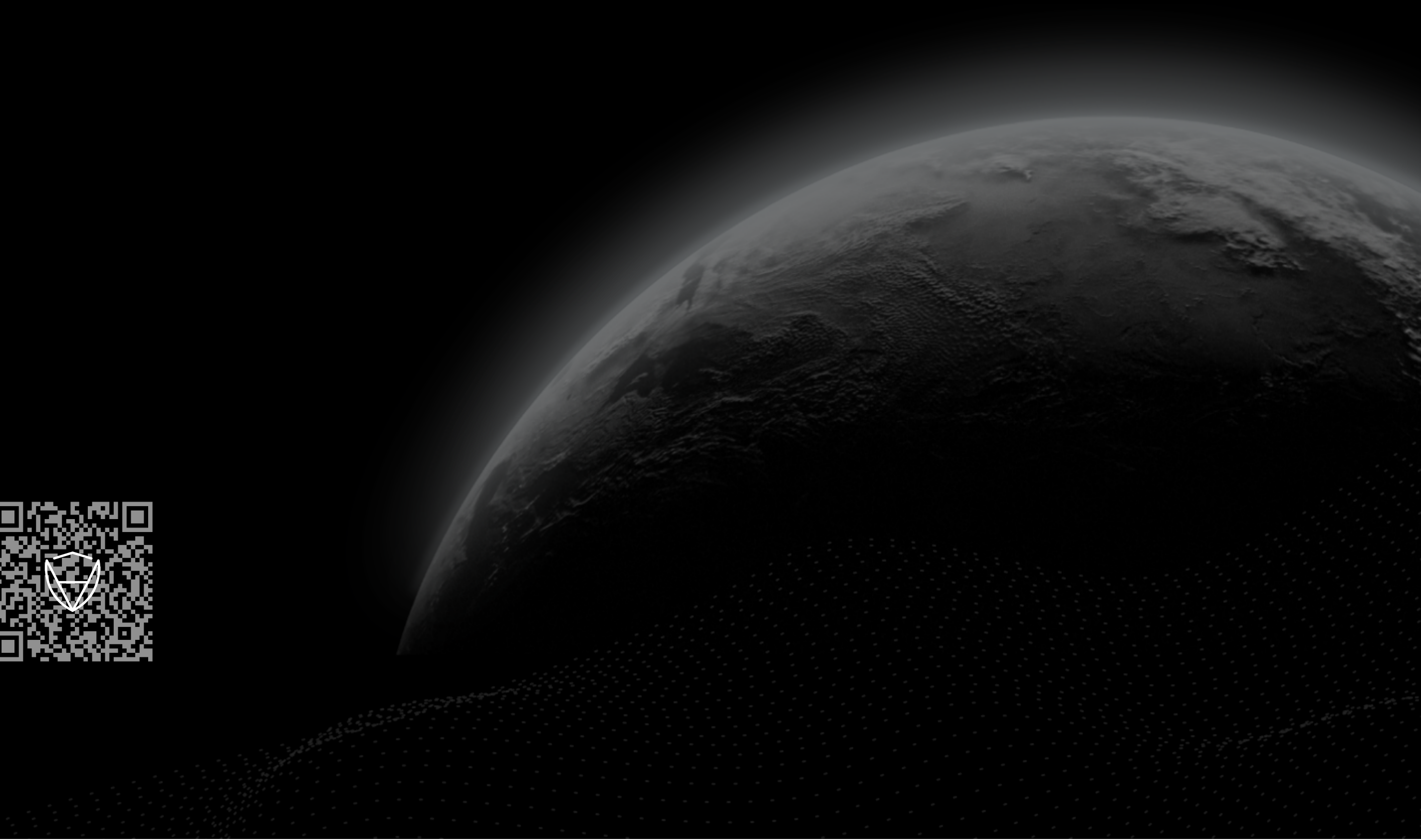
CERTIK

Security Assessment

Rubydex - Audit

CertiK Assessed on Jul 3rd, 2023



CertiK Assessed on Jul 3rd, 2023

Rubydex - Audit

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

Executive Summary

TYPES
Others

ECOSYSTEM
Ethereum (ETH);Binance Smart Chain (BSC)

METHODS
Manual Review, Static Analysis

LANGUAGE
Solidity

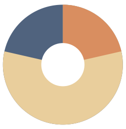
TIMELINE
Delivered on 07/03/2023

KEY COMPONENTS
N/A

CODEBASE
https://github.com/Rubydex/rubydex-smart-contracts
View All in Codebase Page

COMMITS
f7341314654540192a1685b43859f29b29348429
2cce1fc07ba1b62ca09335723bddd70708f25e70
bcb6e237e5342d625decad87c1c7e1440c8a2c35
View All in Codebase Page

Vulnerability Summary



14	6	0	0	8	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.
🟧	3	Major	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.
🟨	0	Medium		Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
🟨	8	Minor	3 Resolved, 5 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
🟦	3	Informational	3 Resolved	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | RUBYDEX - AUDIT

CODEBASE | RUBYDEX - AUDIT

Repository

https://github.com/Rubydex/rubydex-smart-contracts

Commit

f7341314654540192a1685b43859f29b29348429

2cce1fc07ba1b62ca09335723bddd70708f25e70

bcb6e237e5342d625decad87c1c7e1440c8a2c35

AUDIT SCOPE | RUBYDEX - AUDIT

19 files audited ● 7 files with Acknowledged findings ○ 12 files without findings

ID	Repo	File	SHA256 Checksum
● MSR	Rubydex/rubydex-smart-contracts	swapper/MegaSwapper.sol	eb9cb34c63fce51001680acd293e1c846c3f4dd9343f54e091b52c509954d903
● FAR	Rubydex/rubydex-smart-contracts	update/ForcedAction.sol	bec4dfac46a866627cd5fe2ee0817d16a42aa026b4d9452a642bde41ceae1d5d
● USR	Rubydex/rubydex-smart-contracts	update/UpdateState.sol	a48680bc3327fb17c906ab37a87565832ee48319150fbf870e637815f2b71464
● USI	Rubydex/rubydex-smart-contracts	update/UpdateStateImplementation.sol	bc0995c951f8d1b9f7150e4d6210b38fbcc651e425c5249557bd4a09dd863a79
● ARB	Rubydex/rubydex-smart-contracts	utils/Admin.sol	f2b5db07cb7303579d931fb1437197f5b4af36147f997b07e8f8c25ad590af78
● VRB	Rubydex/rubydex-smart-contracts	vault/Vault.sol	23506270f4f669dd3fa41bb2753eb9986727bfe44e1353652c24d02839b1a13c
● VIR	Rubydex/rubydex-smart-contracts	vault/VaultImplementation.sol	832e781e92a9d40d7a1a0c240b6527309b696916dbfe781f8b771afa74c20672
○ ISR	Rubydex/rubydex-smart-contracts	swapper/ISwapper.sol	5f406f22f683a74466353687a067327bfd4394529968bf681401c7af4ea8f5b6
○ IUS	Rubydex/rubydex-smart-contracts	update/IUpdateState.sol	fbc5e7f6806b45f9f3fcc0c69812b115125f5e4e46402f8e31952d570f4c0593
○ USS	Rubydex/rubydex-smart-contracts	update/UpdateStateStorage.sol	71dd2ba292ce4979a46ce19baa9ad48c04db2e14744d5244509e2719483e1e06
○ IAR	Rubydex/rubydex-smart-contracts	utils/IAdmin.sol	f7aca7ed2b47fb5ed75cd2c72177e0b5bfcd68fafabf5dc4bac1dc5d413878f2
○ IER	Rubydex/rubydex-smart-contracts	utils/IERC20.sol	60c2c38a5c97c4761d7c187d97dc08d8f4181e6b0290e36da51543f16c967b0a
○ INV	Rubydex/rubydex-smart-contracts	utils/INameVersion.sol	969c447b6e890e787438bd343bfd15e75e7ec1305c30cec05967bc9fe4960f49

ID	Repo	File	SHA256 Checksum
LRB	Rubydex/rubydex-smart-contracts	utils/Log.sol	4cd794cc912c065ba089078d00aaa673c10c82f0c19ebeba224619479a6d7e6f
NVR	Rubydex/rubydex-smart-contracts	utils/NameVersion.sol	eca1de529f4443cfe1952560aec6d6b5d43348b62298e213e976e0dd7e2135d4
RRP	Rubydex/rubydex-smart-contracts	utils/RevertReasonParser.sol	12489e03d010af46ac198ea1338ce3717e182224a40cdd9c182d8a6a2c5be068
SMR	Rubydex/rubydex-smart-contracts	utils/SafeMath.sol	6cf9b7df1de9a4ec5dc75d10cf17c7fc8ea04b7a35354d533c7b9ac1e82a093e
IVR	Rubydex/rubydex-smart-contracts	vault/IVault.sol	869ef0e9e128241d99c7aaa54c9fc76e27cf54adeb8b1c69a1aad8e81a17284e
VSR	Rubydex/rubydex-smart-contracts	vault/VaultStorage.sol	37d26734077d52f841be854d2b92338adc74d99f586b15f06047949d1d0690d4

APPROACH & METHODS | RUBYDEX - AUDIT

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

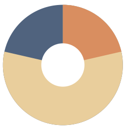
The auditing process pays special attention to the following considerations:

- 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 | RUBYDEX - AUDIT



14	0	3	0	8	3
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Rubydex - Audit. Through this audit, we have uncovered 14 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
CON-01	**Centralized Control Of Contract Upgrade**	**Centralization**	**Major**	**Acknowledged**
CON-02	Dangerous External Call	control-flow	Major	Acknowledged
GLOBAL-04	**Centralization Related Risks**	**Centralization**	**Major**	**Acknowledged**
CON-03	Missing Zero Address Validation	Volatile Code	Minor	Acknowledged
CON-04	Unchecked ERC-20 `transfer()` / `transferFrom()` Call	Volatile Code	Minor	Resolved
CON-05	Suggest Using Openzeppelin's Proxy Patterns	Logical Issue	Minor	Acknowledged
CON-07	Out Of Scope Dependency - Operator	control-flow	Minor	Acknowledged
FAR-01	Divide Before Multiply	math-operations	Minor	Acknowledged
MSR-01	Usage Of `transfer()` For Sending Ether	Volatile Code	Minor	Resolved
MSR-02	Lack Of Balance Check On `outToken`	control-flow	Minor	Acknowledged
USI-01	Flawed Require Check On The Existence Of A Symbol	Logical Issue	Minor	Resolved

ID	Title	Category	Severity	Status
FAR-02	Lack Of `_reentryLock_` Modifier In `forcedWithdraw()` Function	control-flow	Informational	● Resolved
USI-04	Ambiguous Behavior In `addSymbol()` Function	control-flow	Informational	● Resolved
VIR-03	The Mapping `validatorIndex` Cannot Distinguish Non-Signers	Logical Issue	Informational	● Resolved

CON-01 | CENTRALIZED CONTROL OF CONTRACT UPGRADE

Category	Severity	Location	Status
Centralization	● Major	update/UpdateState.sol: 7; vault/Vault.sol: 7	● Acknowledged

Description

The `UpdateState` and `Vault` contracts are upgradeable via their respective proxy contracts, which allows the owner to update the contract's implementation without the community's commitment. However, if an attacker gains access to the owner's account, they can modify the contract's implementation and drain tokens from the contract without the community's knowledge or consent.

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

Rubydex Team: Our business logic necessitates the use of an upgradeable framework for our smart contracts. This approach allows us to adapt to future changes in business requirements and maintain flexibility in our system while ensuring continued functionality and security. We will replace admin, operator, and validator accounts with MPC accounts. Once stable, we will transfer admin privileges to a timelock contract.

CON-02 | DANGEROUS EXTERNAL CALL

Category	Severity	Location	Status
control-flow	● Major	swapper/MegaSwapper.sol: 39, 46; vault/VaultImplementation.sol: 206, 210, 363, 366	● Acknowledged

Description

The `VaultImplementation` contract contains the `swapAndDeposit()` and `tryExecuteWithdraw()` functions that can trigger the `swap()` function in the `MegaSwapper` contract. The `swap()` function performs an external call to the user-input address `caller` , which can potentially be a contract with unknown and potentially malicious code. This vulnerability could be exploited by attackers to carry out various types of attacks.

Recommendation

While we understand that the external call feature in the `VaultImplementation` contract is intentional, it is still recommended that the project team implement a whitelist to restrict input addresses to only trusted and verified contracts. This will reduce the risk of attacks and ensure that all external calls made are safe and authorized. Although there is currently no evidence that the contract has been exploited, the support of external calls always poses a potential security risk for any future product upgrades or third-party integrations. Therefore, it is important to prioritize security measures such as input validation and control to mitigate this risk.

Alleviation

Rubydex Team:

Our contract includes reentrancy protection for transfer-related methods and doesn't pose a gas limit exceeding issues.

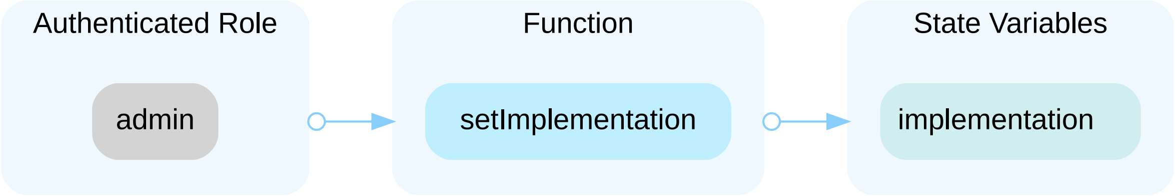
The usage of "caller" in our contract is intentional and specifically designed for interacting with APIs provided by 1inch and 0x. By obtaining the best Swap Address from 1inch/0x API, our frontend ensures that the appropriate "caller" is used for each transaction, optimizing the user experience and maximizing efficiency.

GLOBAL-04 | CENTRALIZATION RELATED RISKS

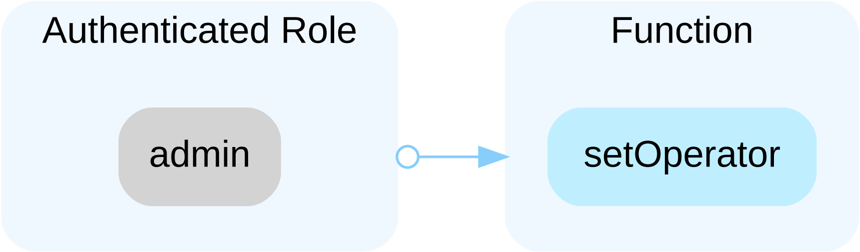
Category	Severity	Location	Status
Centralization	● Major		● Acknowledged

Description

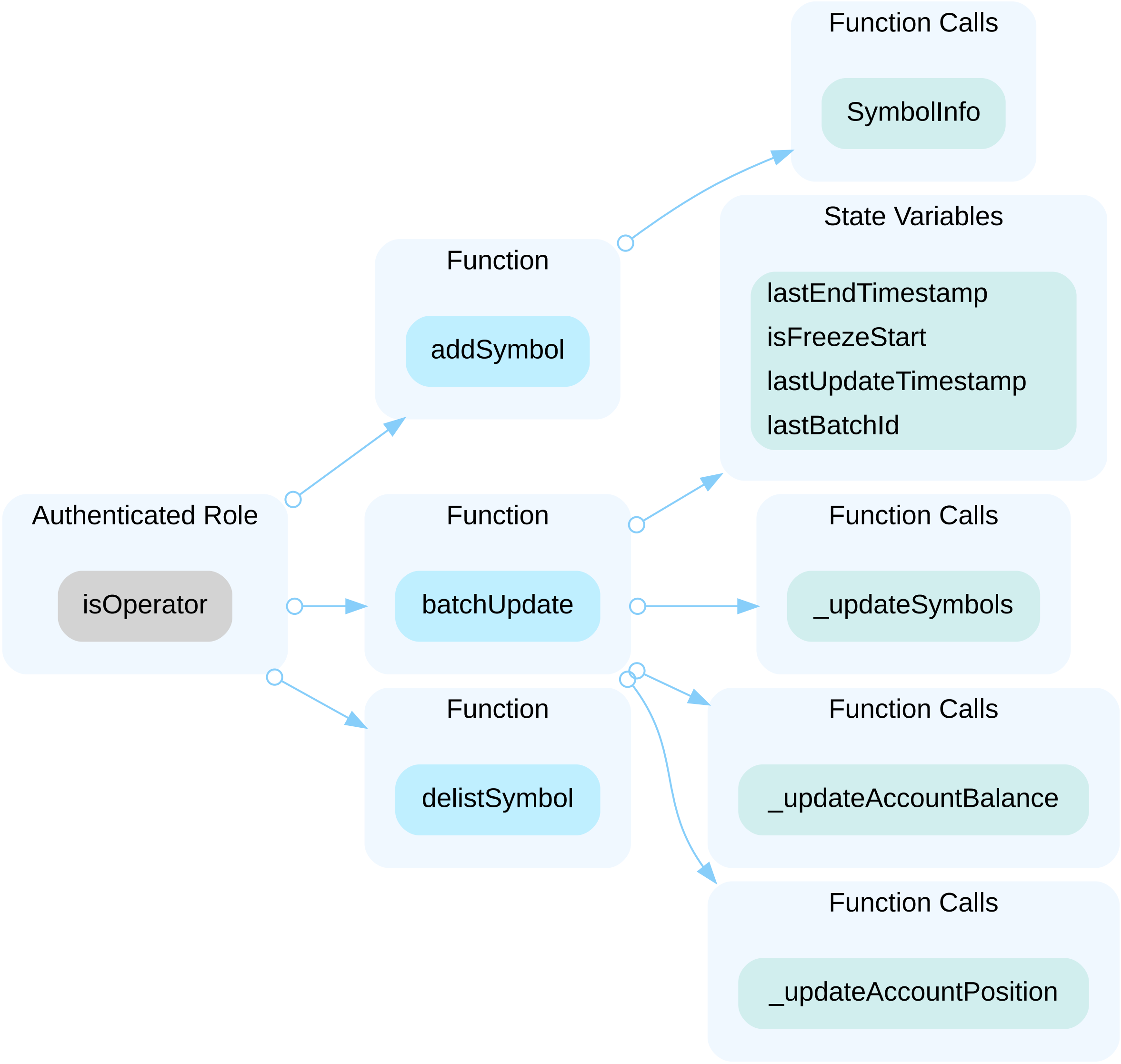
In the contract `UpdateState` the role `admin` has authority over the function shown in the diagram below. Any compromise to the `admin` account may allow the hacker to take advantage of this authority and update the implementation contract.



In the contract `UpdateStateImplementation` the role `admin` has authority over the function shown in the diagram below. Any compromise to the `admin` account may allow the hacker to take advantage of this authority and set operators.



In the contract `UpdateStateImplementation` the role `operator` has authority over the functions shown in the diagram below. Any compromise to the `operator` account may allow the hacker to take advantage of this authority and add/update/delist symbols or update users' balances and positions.

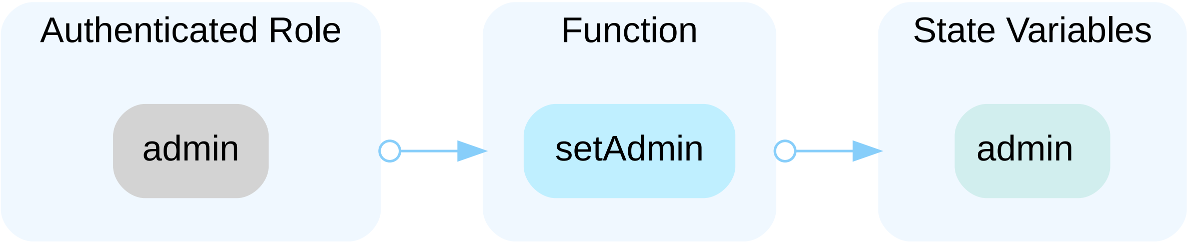


In the contract `UpdateStateImplementation` the role `vault` has authority over the following functions:

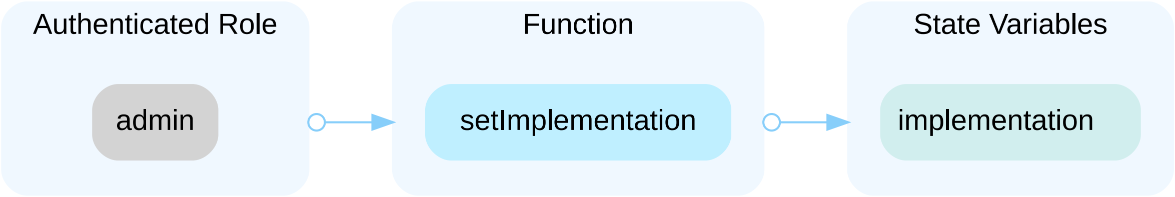
- `updateBalance()`
- `updatePosition()`
- `resetFreezeStart()`

Any compromise to the `vault` account may allow the hacker to take advantage of this authority and update the variable `isFreezeStart` and update users' balances and positions.

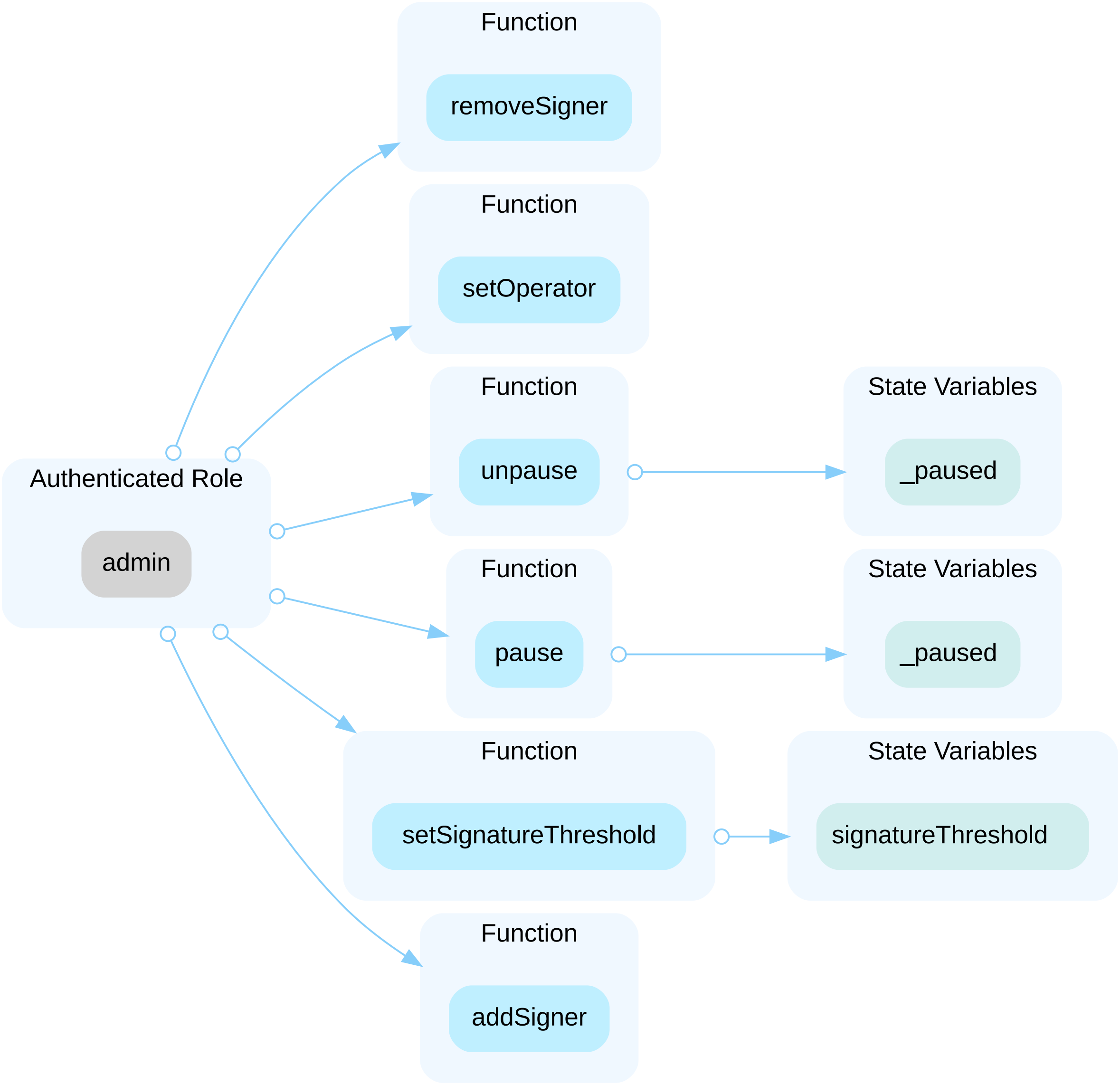
In the contract `Admin` the role `admin` has authority over the function shown in the diagram below. Any compromise to the `admin` account may allow the hacker to take advantage of this authority and transfer the admin role.



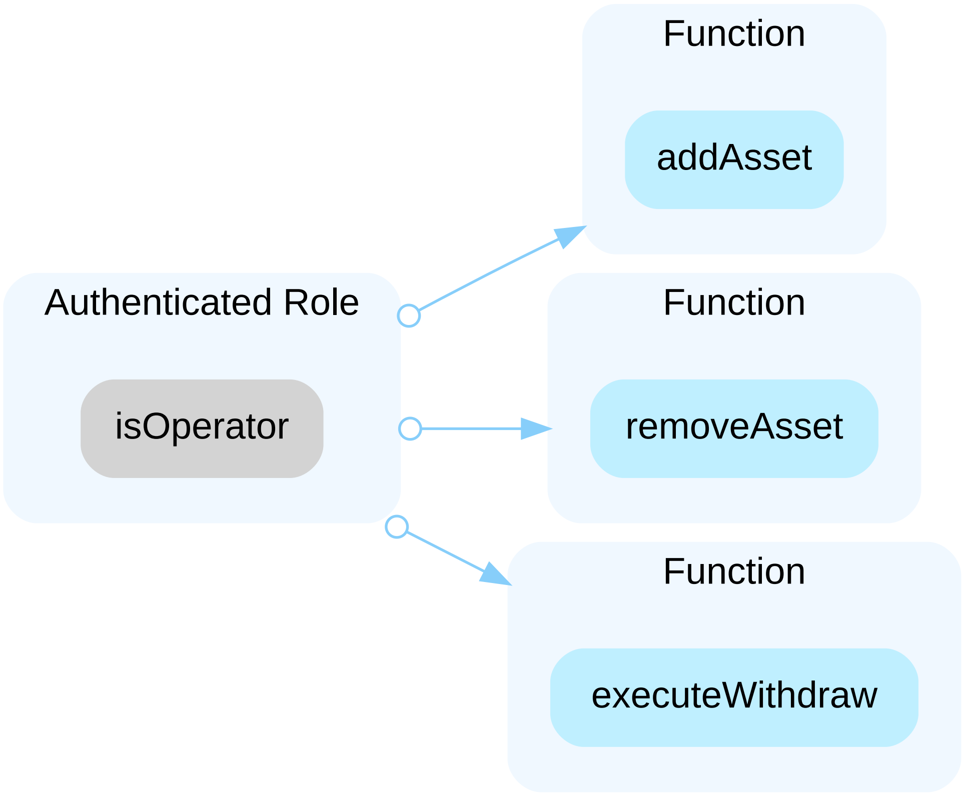
In the contract `Vault` the role `admin` has authority over the function shown in the diagram below. Any compromise to the `admin` account may allow the hacker to take advantage of this authority and update the implementation contract.



In the contract `VaultImplementation` the role `admin` has authority over the functions shown in the diagram below. Any compromise to the `admin` account may allow the hacker to take advantage of this authority and add/remove signers, set operators, pause/unpause contract, and set the minimal number of signers needed for withdrawals.



In the contract `VaultImplementation` the role `operator` has authority over the functions shown in the diagram below. Any compromise to the `operator` account may allow the hacker to take advantage of this authority and add/remove assets and execute withdrawals.



In the contract `VaultImplementation` the role `update` has authority over the following function:

- `transferOut()`

Any compromise to the `update` account may allow the hacker to take advantage of this authority and transfer tokens out.

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 recommend carefully managing 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., multi-signature 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 of privileged operations;
  AND

- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key being 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 of 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

Rubydex Team: We will replace admin, operator, and validator accounts with MPC accounts. Once stable, we will transfer admin privileges to a timelock contract.

CON-03 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	● Minor	swapper/MegaSwapper.sol: 39, 46, 63; update/UpdateState.sol: 11; utils/Admin.sol: 22; vault/Vault.sol: 11	● Acknowledged

Description

Addresses should be checked before assignment or external call to make sure they are not zero addresses.

```
39            (bool success, bytes memory result) = address(caller).call{value:msg.value}(data);
```

```
46            (bool success, bytes memory result) = address(caller).call(data);
```

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

```
63                payable(recipient).transfer(outAmount);
```

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

```
11        implementation = newImplementation;
```

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

```
22        admin = newAdmin;
```

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

```
11        implementation = newImplementation;
```

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

Recommendation

We advise adding zero-checks for the passed-in address values to prevent unexpected errors.

Alleviation

Rubydex Team: To conserve gas, we better not perform redundant checks for zero address validation.

CON-04 | UNCHECKED ERC-20 `transfer()` / `transferFrom()` CALL

Category	Severity	Location	Status
Volatile Code	● Minor	swapper/MegaSwapper.sol: 58; vault/VaultImplementation.sol: 120, 209, 232, 356, 365, 371	● Resolved

Description

The return value of the transfer()/transferFrom() call is not checked.

```
58                  IERC20(outToken).transfer(recipient, outAmount);
```

```
120             IERC20(asset).transfer(account, amount);
```

```
209             IERC20(inToken).transferFrom(account, address(swapper), inAmount);
```

```
232             IERC20(token).transferFrom(account, address(this), amount);
```

```
356                 IERC20(token).transfer(request.account, request.inAmount);
```

```
365                 IERC20(token).transfer(address(swapper), request.inAmount);
```

```
371                  IERC20(request.outToken).transfer(request.account, outAmount);
```

Recommendation

Since some ERC-20 tokens return no values and others return a `bool` value, they should be handled with care. We advise using 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

Resolved at commit 2cce1fc07ba1b62ca09335723bddd70708f25e70.

CON-05 | SUGGEST USING OPENZEPPELIN'S PROXY PATTERNS

Category	Severity	Location	Status
Logical Issue	● Minor	update/UpdateState.sol; vault/Vault.sol	● Acknowledged

Description

There are a few issues with the current proxy pattern being used in the contract, including a missing storage gap in the `ForcedAction` , `Admin` , and `NameVersion` contracts which makes it impossible to add new state variables, as well as the use of immutable variables in upgradeable contracts which can lead to potential issues.

Reference:

- https://docs.openzeppelin.com/upgrades-plugins/1.x/faq#why-cant-i-use-immutable-variables
- https://docs.openzeppelin.com/contracts/3.x/upgradeable#storage_gaps

Recommendation

To ensure the reliability and security of the proxy pattern being used, we recommend using well-established and thoroughly tested proxy patterns, such as OpenZeppelin's proxy patterns, rather than inventing custom proxy patterns. Self-invented proxy/upgradeable system is not recommended.

Alleviation

Rubydex Team: Our contract extensively uses immutable and constant for business purposes, which OpenZeppelin does not support. Also, it is not yet well-adapted to some new EVM-compatible chains, such as zkSync. We will deploy on multiple chains in the near future, so it cannot meet our needs. The current proxy-implementation code framework meets our requirements as it's already been used with some major defi projects for the last two years. The mentioned Admin and NameVersion contracts are utility contract types and do not require upgrades. The ForcedAction contract's parent is UpdateStateStorage, and its storage upgrade is solely controlled by UpdateStateStorage.

CON-07 | OUT OF SCOPE DEPENDENCY - OPERATOR

Category	Severity	Location	Status
control-flow	● Minor	update/UpdateStateImplementation.sol: 12; vault/VaultImplementation.sol: 16	● Acknowledged

Description

The system has privileged roles called `operator` who have the power to manage funds in the smart contracts. The scope of the audit treats the `operator` as black boxes and assumes their functional correctness. However, if an `operator` is compromised, the attacker can take advantage of that and take away all the funds in the contract.

Recommendation

Based on the auditor's observation, the `operator` is likely controlled through an API server backend. The team should ensure that the backend is correctly implemented and secure. If applicable, a penetration test against the backend server is recommended to ensure server safety.

Alleviation

Rubydex Team: We will strengthen the protection of our backend operator account.

FAR-01 | DIVIDE BEFORE MULTIPLY

Category	Severity	Location	Status
math-operations	● Minor	update/ForcedAction.sol: 62, 63~64	● Acknowledged

Description

Performing integer division before multiplication truncates the low bits, losing the precision of calculation.

```
62              int256 funding = -int256(symbolStats.cumulativeFundingPerVolume - pos.lastCumulativeFundingPerVolume) * ONE / pricePrecision / volumePrecision / FUNDING_PRECISION * int256(pos.volume) * ONE / volumePrecision / ONE;
```

```
63              int256 pnl = - (int256(pos.entryCost) * ONE * int256(tradeVolume).abs() / int256(pos.volume).abs() / pricePrecision / volumePrecision +
64                  int256(tradeVolume) * ONE / volumePrecision * int256(symbolStats.indexPrice) * ONE / pricePrecision / ONE);
```

Recommendation

To avoid potential loss of precision, we recommend applying multiplication before division. Additionally, it is important to be careful to avoid integer overflow when performing arithmetic operations.

Alleviation

Rubydex Team: Our current code ensures no loss of precision at each step and avoids overflow.

MSR-01 | USAGE OF `transfer()` FOR SENDING ETHER

Category	Severity	Location	Status
Volatile Code	● Minor	swapper/MegaSwapper.sol: 63	● Resolved

Description

It is not recommended to use Solidity's `transfer()` function for transferring Ether, since some contracts may not be able to receive the funds. Those functions forward only a fixed amount of gas (2300 specifically) and the receiving contracts may run out of gas before finishing the transfer. Also, EVM instructions' gas costs may increase in the future. Thus, some contracts that can receive now may stop working in the future due to the gas limitation.

```
63                  payable(recipient).transfer(outAmount);
```

- `MegaSwapper.swap` uses `transfer()`.

Recommendation

It is recommended to replace the `transfer()` function with the `call()` function for the ETH transfers in the `swap()` function.

Alleviation

Resolved at commit 2cce1fc07ba1b62ca09335723bddd70708f25e70.

MSR-02 | LACK OF BALANCE CHECK ON `outToken`

Category	Severity	Location	Status
control-flow	● Minor	swapper/MegaSwapper.sol: 28	● Acknowledged

Description

The `swap()` function does not check the balance of the `outToken` before executing the function call. This lack of validation could lead to issues if tokens are accidentally sent to the contract, as these tokens could be transferred to the next caller's balance.

Recommendation

We recommend modifying the `swap()` function to only transfer the difference in balance of the `outToken` to the caller. Additionally, we recommend adding additional withdraw functions for the owner to withdraw any locked tokens.

Alleviation

Rubydex Team: This contract serves as a helper contract and does not retain any funds by design. As a result, there is no need to perform a balance check on outToken in this particular case.

USI-01 FLAWED REQUIRE CHECK ON THE EXISTENCE OF A SYMBOL

Category	Severity	Location	Status
Logical Issue	● Minor	update/UpdateStateImplementation.sol: 102	● Resolved

Description

```
    require(!symbols[symbolId].delisted, "update: symbol not exist or delisted");
```

The require check is used to ensure that a symbol exists and has not been delisted before allowing it to pass validation. However, symbols that do not exist can pass the validation. In this scenario, if symbol A does not exist, `symbols[A].delisted` would be false, `!symbols[A].delisted` would be true, allowing non-existent symbols to pass validation.

Recommendation

We recommend adding a sanity check to ensure that a symbol exists. Additionally, we recommend modifying the error message to provide the correct information.

Alleviation

Resolved at commit 2cce1fc07ba1b62ca09335723bddd70708f25e70.

FAR-02 | LACK OF `_reentryLock_` MODIFIER IN `forcedWithdraw()` FUNCTION

Category	Severity	Location	Status
control-flow	● Informational	update/ForcedAction.sol: 39	● Resolved

Description

The `forcedWithdraw()` function does not include the `_reentryLock_` modifier to prevent reentrancy attacks.

Recommendation

We recommend adding the `_reentryLock_` modifier to the `forcedWithdraw()` function.

Alleviation

Resolved at commit 2cce1fc07ba1b62ca09335723bddd70708f25e70.

USI-04 | AMBIGUOUS BEHAVIOR IN `addSymbol()` FUNCTION

Category	Severity	Location	Status
control-flow	● Informational	update/UpdateStateImplementation.sol: 72	● Resolved

Description

The `addSymbol()` function is named as if it is intended only for adding new symbols to the `symbols` mapping, but it is actually capable of updating existing symbols as well. This ambiguity raises concerns about the intended design of the function.

Recommendation

We recommend reviewing the intended behavior of the `addSymbol()` function to determine whether updating existing symbols is part of its intended design. If it is not, we recommend renaming the function to better reflect its behavior and to avoid confusion.

Alleviation

Resolved at commit bdbab298e295b9f8d80e6556bd455ceffba60ccf.

VIR-03 | THE MAPPING `validatorIndex` CANNOT DISTINGUISH NON-SIGNERS

Category	Severity	Location	Status
Logical Issue	● Informational	vault/VaultImplementation.sol: 157	● Resolved

Description

The contract uses a mapping `validatorIndex` to show if an account is a signer and give the index in the `validSigners` array. However, one of the signers has the value of 0 in the mapping, and non-signers also have the value of 0 in the same mapping. This can create critical issues when displaying signers and non-signers, as non-signers can be easily confused with signers. Given that the contract is upgradeable, this issue could potentially bring major issues in the future's signer check process.

Recommendation

We recommend using other non-zero values, such as `index + 1`, to indicate that an account is a validator in the mapping `validatorIndex`.

Alleviation

Resolved at commit bcb6e237e5342d625decad87c1c7e1440c8a2c35.

APPENDIX | RUBYDEX - AUDIT

Finding Categories

Categories	Description
	Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.
	Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.
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.

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.

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.