CERTIK

Security Assessment

Orderly Network

CertiK Verified on Sept 16th, 2022

CertiK Verified on Sept 16th, 2022

Orderly Network

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

Executive Summary

TYPES	ECOSYSTEM	METHODS
DeFi	Near	Manual Review, Static Analysis
LANGUAGE	**TIMELINE**	**KEY COMPONENTS**
Rust	Delivered on 09/16/2022	N/A

CODEBASE

Audited Commit: 11a5ae159463086741141083ae86e30e80fad69a

Remediation Commit: 1c6206b6ffb8faeabf94d77ff49e56afe20da9f5

...View All

COMMITS

11a5ae159463086741141083ae86e30e80fad69a

...View All

Vulnerability Summary

21	19	1	0	1	0	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined	Unresolved

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.
2	Major	1 Resolved, 1 Mitigated	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
8	Medium	8 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
3	Minor	2 Resolved, 1 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
8	Informational	8 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 | ORDERLY NETWORK

CODEBASE | ORDERLY NETWORK

Repository

Audited Commit: 11a5ae159463086741141083ae86e30e80fad69a

Remediation Commit: 1c6206b6ffb8faeabf94d77ff49e56afe20da9f5

Commit

11a5ae159463086741141083ae86e30e80fad69a

AUDIT SCOPE | ORDERLY NETWORK

9 files audited ● 1 file with Acknowledged findings ● 3 files with Mitigated findings ● 2 files with Resolved findings ● 3 files without findings

ID	File	SHA256 Checksum
OPE	src/operator.rs	f6172a16caeedf2a7877788c7c0d848e55c2da45093805997634a06f0e1f2505
APP	src/approvers.rs	70ba011a72fad180a17feeaeefc8932085b7366e6cf4ac689742f7ef698bb605
CON	src/contract.rs	17af4a13251d9d07aaa1db29202d49c7957a76433587513307844e3ca2108c0a
OWN	src/owner.rs	b72d9f02507b5fe1891c159db154d743d0a6741eda9ce95772017d9db68ebfa6
COT	src/contract_utils.rs	1e52cfe0692171cc0e4cedeacd4f2238c42be20ab3cd979399546f1c5e882bf8
TYP	src/types.rs	ae39a203ba698c6033203576b9c00d8ba78eb9355444018f4968d1518b24774d
EVE	src/event.rs	31af61a7a4e6bd24acde19a90e5e518e30ae9cc8ce532b0febd603abea4e997e
LIB	src/lib.rs	9ea3d780853d4dab8c58f55c9d3dcf5bdf0d10cc35b61762d2418caf47d8cd0d
TOK	src/token_balance.rs	1a4ecefffff6bef6ed4b9a53f2d64634a58fe2413f0279a61b412dfda40185c41

APPROACH & METHODS | ORDERLY NETWORK

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

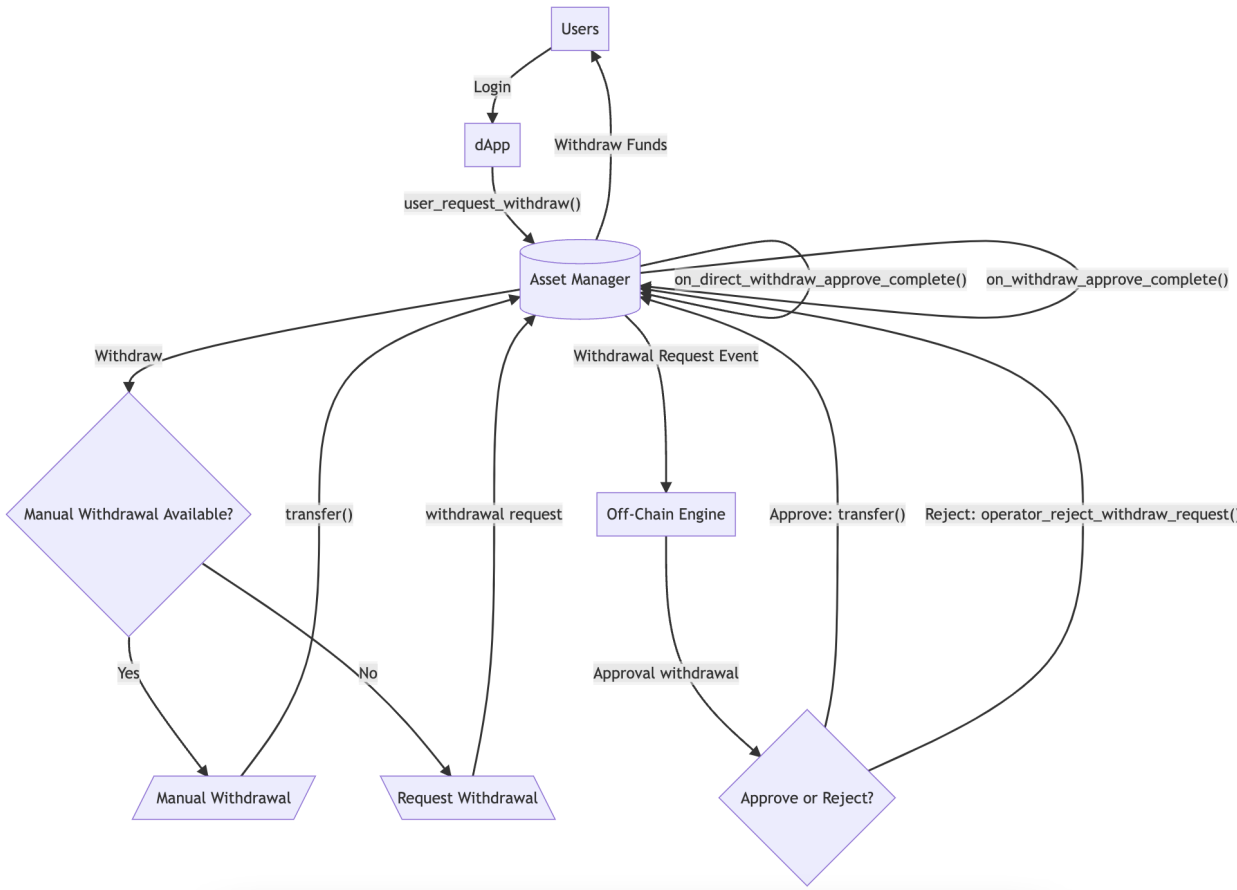
- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and 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.

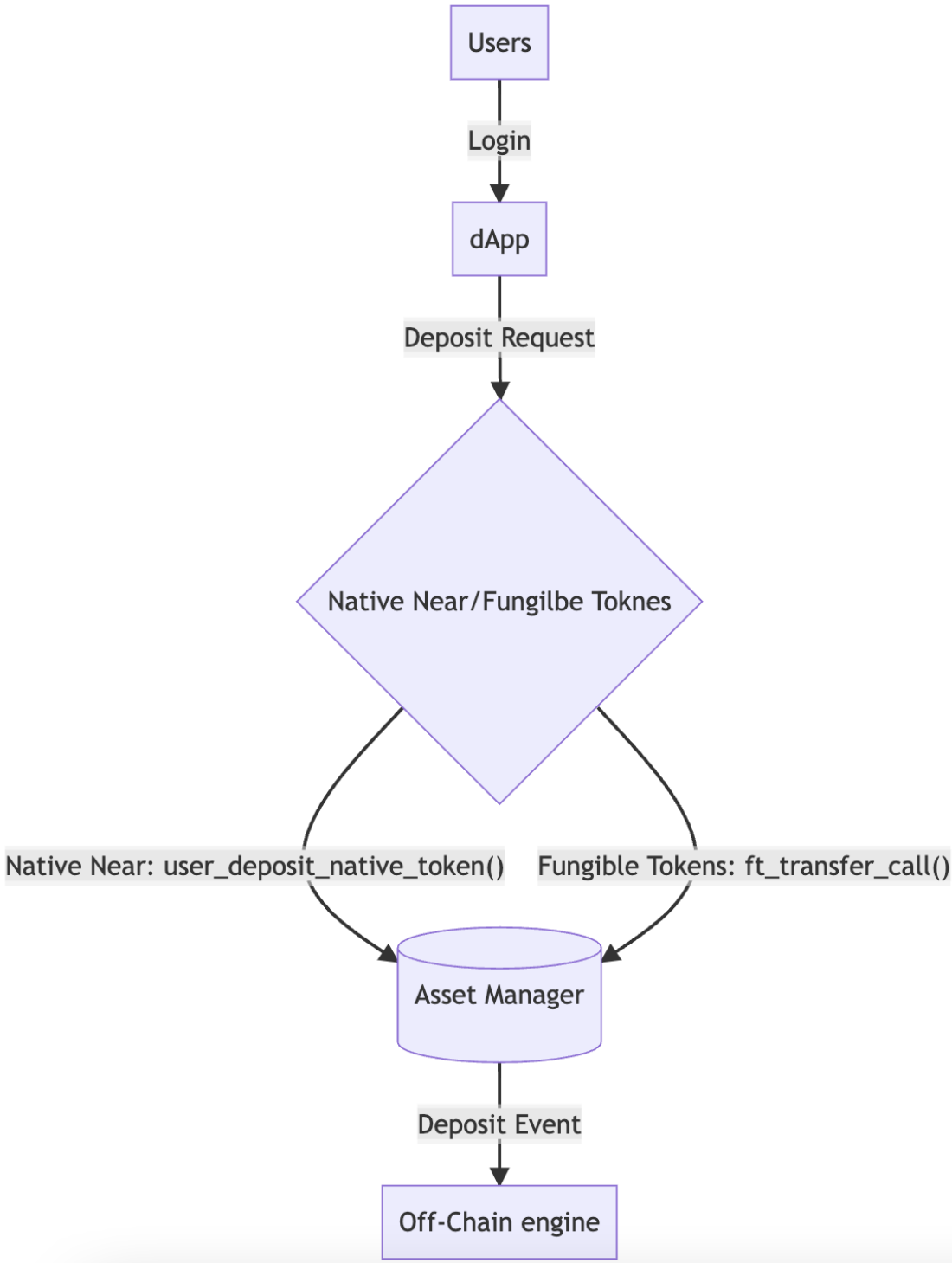
SYSTEM OVERVIEW | ORDERLY NETWORK

Orderly Network is a decentralized exchange built on the Near blockchain. It aims to provide a liquidity layer for any dApp with an on-chain orderbook. In the context of this audit, the audited codebase is an asset manager smart contract. The other component of the codebase is an out of scope off-chain orderbook which provides the matching functionality.



Review Notes

The asset manager contract holds the registered users, trading keys of users, token balances of user-token pairs, withdraw requests of user-token pairs, and whitelist of tokens. Effectively this contract is an escrow type contract that allows users to deposit or withdraw assets into the contract. The asset manager smart contract is also responsible for uploading batch trades data that comes from the off-chain engine. Once the batch trading data is verified, it will update the balances of both selling and buying tokens associated with the trade account, and accumulate the fee asset which will benefit the fee collector.



If the Operator Manager does not interact with the Asset Manager contract for a limited period of time, then the user can directly withdraw their deposit. Otherwise, the withdrawal operation will be put on hold, waiting for the approval of the Operator Manager. In the following two cases, users cannot withdraw their deposits: first is when the Operator Manager rejects the withdrawal request; and the second is when the Operator Manager frequently pings the Asset Manager contract while not approving the withdrawal request. The request-approval mechanism is used for key state updates of the Asset Manager contract, such as owner changes, adding approvers, adding authorized users, setting listing tokens/symbols, etc. Only the owner or approver if the approvers request is enabled can submit a request. The request will only be approved if 51% of approvers agree. If the request is related to a role assignment, it will only go through if 51% of the approvers and the new account for that role both approve.

Centralization and Roles

The deployer of the asset manager smart contract has full access over the asset manager contract. So the deployer has the ability to delete the asset manager contract and transfer all the `NEAR` to other contracts. There are several roles including Owner, Operator Manger, Approver as well as Fee Collector within the Asset Manager smart contract. These roles can be set initially while this contract is deployed.

- The Owner or Approver(if and only if the request approval is enabled) can update these roles, maintain authorized users, and create verifying keys of withdrawal and trading, and set tokens and symbols listed or delisted. These operations are initiated by creating the corresponding approval requests.
- The Approver could vote such requests, once the enough votes(currently more than half) acquired, requests will be processed.
- The Operator Manager can execute actions including withdrawal action and uploading batch trading data.
- The Fee Collector will collect fee asset during uploading batch trading data. The Asset Manager contract guarantees that the four roles, Operator Manager, Fee Collector, Owner and Approver are both different from each other and from the Asset Manager contract account ID.

The above roles, except for fee collector, can be trading accounts.

Off-chain Matching Engine Assumption

When users withdraw the funds out of `Asset Manger` contract, it will send a withdrawal request to the off-chain engine which will first approve or reject the request and then notify the `Operator` to call `operator_execute_action()` function.

Further, the off-chain engine also interacts with `Asset Manager` to upload batch trade data which contains the amount of transfer tokens and quantity of deposit token. The logic of off-chain engine is out scope for this audit and will be treated as blackbox. We assume it behaves correctly.

FINDINGS | ORDERLY NETWORK

21	0	2	8	3	8
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Orderly Network. Through this audit, we have uncovered 21 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
APP-01	Invalid Request Type	Logical Issue	Medium	Resolved
APP-02	`new_account_approved` Approval Logic	Logical Issue	Medium	Resolved
CON-01	Potential Million Small Deposits Attack	Logical Issue	Major	Resolved
COR-01	Function `user_request_withdraw()` Not Marked Payable	Logical Issue	Medium	Resolved
COT-01	Visibility Of `new_withdraw_id()`	Language Specific	Medium	Resolved
OPE-01	Remove Withdraw Info For Current Pair	Logical Issue	Medium	Resolved
OPE-02	Missing Check For Duplicated Trade	Logical Issue	Minor	Acknowledged
OPR-01	Missing Update User Ledger	Logical Issue	Medium	Resolved
OWN-01	The New Approver Can Be Owner	Logical Issue	Minor	Resolved
SR0-01	Inaccurate Storage Usage Calculation	Logical Issue, Mathematical Operations	Medium	Resolved

ID	Title	Category	Severity	Status
SRC-01	**Centralization Related Risks**	**Centralization / Privilege**	**Major**	**Mitigated**
SRC-02	Missing Input Validation	Logical Issue	Minor	Resolved
STO-01	Missing Cross Account Transfer Callback Function	Logical Issue	Medium	Resolved
CON-02	Useless Conversion	Language Specific	Informational	Resolved
CON-03	Incorrect Error Message	Logical Issue	Informational	Resolved
CON-04	Missing Emit Events	Coding Style	Informational	Resolved
OWN-02	Unused Code	Coding Style	Informational	Resolved
SRC-03	Needless Borrow	Language Specific	Informational	Resolved
SRC-04	Unnecessary `let` Binding	Language Specific	Informational	Resolved
SRC-05	Error Message	Logical Issue	Informational	Resolved
SRC-06	Potential Overflow	Mathematical Operations	Informational	Resolved

APP-01 | INVALID REQUEST TYPE

Category	Severity	Location	Status
Logical Issue	Medium	src/approvers.rs (11a5ae159463086741141083ae86e30e80fad69a): 89~94, 127~132, 156~161, 211~215, 240~244, 268~273, 312~317, 336~341	Resolved

Description

If the owner or an approver creates an approval request, it will cause an error if they try to process it more than once in the following functions:

1. `approve_approver_request_persmission`
2. `approve_symbol_listing`
3. `approve_symbol_delisting`
4. `approve_withdraw_verify_key`
5. `approve_trade_verify_key`
6. `approve_autorized_access`
7. `new_account_approved`
8. `token_action_approve`

The error message instructs the user to create a new request. However, if a new request is created then these functions will panic as there is already such request type in existance.

```
pub(crate) fn create_approver_request(
    &mut self,
    request_type: ApproveRequestType,
    request: ApproveRequest,
) {
    if self.approve_requests.contains_key(&request_type) {
        env::panic_str("This type of request is already in progress");
    }

    self.approve_requests.insert(&request_type, &request);
}
```

Recommendation

While invalid request will not occur frequently, we recommend removing the invalid approval request in the mentioned locations or refactoring the code so that pending requests can be directly deleted.

Alleviation

[`CertiK`]: The Orderly Network team has added logic within the code to directly delete pending requests if an error causes the request approval to fail.

APP-02 | `new_account_approved` APPROVAL LOGIC

Category	Severity	Location	Status
Logical Issue	Medium	src/approvers.rs (11a5ae159463086741141083ae86e30e80fad69a): 295~311, 378~380	Resolved

Description

The function `new_account_approved`, updates an approval request dependent on the number of approvals. Further, an approval request will only be successfully approved if the new role account and a sufficient number of approvers agree to the request.

However consider the following case where the approval request will not be completed. Assume the following:

1. There are 10 approvers
2. The number of approvals required is 6 = 10/2+1
3. The "SetOwner" request is in progress.

There is no maximum limit on the number of approvers allowed to approve a request. Therefore the `approved_by' field can be larger then the sufficient amount. If the` approved_by `is strictly larger then the sufficient amount then the` request.approved_by.is_set_enough() `function will return false. The new "owner" account will never be successfully approved. And another new` SetOwner` request cannot be created because of that stuck request.

Recommendation

We advise refactoring the codes to avoid the pending request that are unable to be approved. For example, a potential solution is refactoring the `is_set_enough()` function as below:

```
fn is_set_enough(&self, required_number: u8) -> bool {
    bits_count(*self) >= required_number as u64
}
```

Alleviation

[`CertiK`]: The Orderly Network team has changed the comparison operator from the equality operator '`==`' to the greater than or equal to operator '`>=`'. Further for the function `is_set_enough()`, it returns true the number of the approvers who agree the approval request is greater than required approvers.

CON-01 | POTENTIAL MILLION SMALL DEPOSITS ATTACK

Category	Severity	Location	Status
Logical Issue	Major	src/contract.rs (11a5ae159463086741141083ae86e30e80fad69a): 424	Resolved

Description

There does not exist a storage management system for determining if the contract holds sufficient `near` to maintain storage. Further, user's are not responsible for storage fees. As a result, the `Asset Manager` contract is responsible for paying for storage. However this leaves the contract vulnerable to the `Million Small Deposits Attack`. This can lead the contract to being cost prohibitive to call. For example, the `create_user_account()` is a public function. The more users that call this function, the larger the stake balance is needed to cover storage.

Another function call that needs proper storage fees is `operator_execute_action()`. Users are not responsible for sending trades to chain and updating the balances. An incredible amount of small volume trades, could possibly create unintended results without proper management.

There are other functions that also write to storage but are less vulnerable to such an attack. For example in owner.rs, the following functions update the state of the contract. These functions should require Near to call and should track the deposited stake:

1. The function `create_approver_request` updates the state of the LookupMap `approve_requests`.
2. The function `add_authorized_account` updates the state of the LookupSet `authorized_users`.

Recommendation

We recommend that the team implements storage management system with respect to `NEP-145`. Further require that User's pay their storage fees for account creation and maintain balances for privileged accounts that read and write to storage.

Alleviation

[`CertiK`]: The Orderly Network team added codes for asking users to pay for the storage usage generated by user registration and other data saving on Near.

COR-01 FUNCTION `user_request_withdraw()` NOT MARKED PAYABLE

Category	Severity	Location	Status
Logical Issue	Medium	src/contract.rs (03554f17f43b8f11b5acbd11cc8b7819ed9fe701): 329	Resolved

Description

The `user_request_withdraw()` function requires users to deposit 1 yoctoNear to withdraw assets, however, the function itself is not marked as `payable`.

Recommendation

We recommend adding `#[payable]` macro for this function.

Alleviation

[`CertiK`]: The team resolved this finding as recommended in commit hash ebc36db0dc1fc331b51636356de14febc2e0cec5.

COT-01 | VISIBILITY OF `new_withdraw_id()`

Category	Severity	Location	Status
Language Specific	Medium	src/contract_utils.rs (11a5ae159463086741141083ae86e30e80fad69a): 156~159	Resolved

Description

The function `new_withdraw_id()` is called when a transaction user requests a withdrawal. However, this function has public visibility. This can cause the value of `global_withdraw_id` to be modified by any NEAR account. Therefore the exact number of withdrawal operations cannot be read from the log.

Recommendation

We advise updating the visibility of the function `new_withdraw_id()` to be internal.

Alleviation

[`CertiK`]: The Orderly Network team has changed the keyword `pub` to `pub(crate)` . So the `new_withdraw_id()` function is restricted to be visible only in the crate scope.

OPE-01 | REMOVE WITHDRAW INFO FOR CURRENT PAIR

Category	Severity	Location	Status
Logical Issue	Medium	src/operator.rs (11a5ae159463086741141083ae86e30e80fad69a): 95	Resolved

Description

In the function, `operator_reject_withdraw_request()`, regardless of the result from calling the function `complete_withdraw`, the `user_withdraw_requests` are completely removed from the respective mapping at Line #95. If the user has more than one `withdraw_requests`, this can result in loss of data.

```
80          withdraw_request.complete_withdraw(withdraw_id, request.amount.0);
81          if withdraw_request.is_empty() {
82              self.user_withdraw_requests.remove(&key);
83          } else {
84              self.user_withdraw_requests.insert(&key, &withdraw_request);
85          }
86
87          Self::emit_event(Event::WithdrawApprove {
88              user: user.clone(),
89              token: token.clone(),
90              amount: request.amount,
91              withdraw_id,
92              is_withdraw_success: false,
93              event_id: Some(request.event_id),
94          });
95          self.user_withdraw_requests.remove(&key);
```

Only the current withdraw request should be removed according to the logic.

Recommendation

We recommend removing Line #95 so as not to delete valid `withdraw_requests` for a given user.

Alleviation

[`CertiK`]: The Orderly Network team has deleted the statement `self.user_withdraw_requests.remove(&key);` at L95.

OPE-02 | MISSING CHECK FOR DUPLICATED TRADE

Category	Severity	Location	Status
Logical Issue	Minor	src/operator.rs (11a5ae159463086741141083ae86e30e80fad69a): 25~28, 357~362	Acknowledged

Description

The role `operator manager` account is responsible for uploading trades generated by the off-chain matching engine program. The structure `TradeUpload` in `types.rs` is used to store the information about the uploaded trades. For the information present in the contracts, the field `trade_id` identifies a unique trade. There is no restriction in the code that prevents the same trade from being uploaded more than once. Therefore, the balance of the user in the "Asset Management" contract will be increased/decreased multiple times for the same single trade.

Recommendation

We recommend that a given transaction should be restricted to be uploaded only once on chain. This can be accomplished for example by allowing the off-chain engine programs to limit duplicate uploads.

Alleviation

[`Certik`]: As of the current design there is no feasible way to verify duplicated trades as per the Orderly Network team.

OPR-01 | MISSING UPDATE USER LEDGER

Category	Severity	Location	Status
Logical Issue	Medium	src/operator.rs (03554f17f43b8f11b5acbd11cc8b7819ed9fe701): 285 ~292	Resolved

Description

The contract state data read from `UnorderedMap` is just a copy of data on the Near chain. The changes in that copy data will not automatically synchronize with the on-chain data. So the `UnorderedMap` function `insert()` should be called again if you want to put the new data on the chain.

If the operator-manager has approved the key-removal action, the longer-used keys will be deleted from the `user_record` stored in `user_ledger`, an `UnderedMap` structure data. The `user_record` instance is just a copy of the data stored in `UnderedMap`, and after the keys' deletion, the function `insert()` of `UnderedMap` is never called. In this case, the longer-used keys wanted to delete still exist, and the user can still use the no longer-used keys to create a new order and execute it in the `asset_manager` contract.

Recommendation

We recommend refactoring the code to update the value of the `user_ledger` instance after removing the no longer-used keys.

Alleviation

[`CertiK`]: The Orderly Network team added the codes for updating `user_ledger` instance after removing the no longer-used keys.

OWN-01 | THE NEW APPROVER CAN BE OWNER

Category	Severity	Location	Status
Logical Issue	Minor	src/owner.rs (11a5ae159463086741141083ae86e30e80fad69a): 39, 43 ~48	Resolved

Description

According to the specifications of the function `set_approver`, the new `approver` cannot be the current `owner` of the `asset manager` contract. However, there is no logic in the function to enforce that the new approver account is not the `owner'.

Recommendation

We advise adding a check to verify that the new approver is not the `owner` account.

Alleviation

[`CertiK`]: The Orderly Network team has added the code logic to avoid the `owner` of `asset manager` contract being the new approver.

SR0-01 | INACCURATE STORAGE USAGE CALCULATION

Category	Severity	Location	Status
Logical Issue, Mathematical Operations	Medium	src/account_deposit.rs (03554f17f43b8f11b5acbd11cc8b7819ed9fe701): 36~42; src/storage_management.rs (03554f17f43b8f11b5acbd11cc8b7819ed9fe701): 54~55, 57	Resolved

Description

Near provides two methods either by manual byte math or using the SDK environment. The manual method is difficult and requires manual gas measurements. On the other hand, checking `env.storage_usage()` is a more accessible method.

In the codebase, a user's storage cost is calculated based on the manual byte math. We provide the user registration example to show that the method used in the codebase contains an inaccuracy.

When registering a new user in the `asset manager` contract, the types `AccountId` and `AccountV2` data will be saved on the chain. The const variable `INIT_ACCOUNT_WITH_ONE_KEY_STORAGE`, whose value is 358 bytes, denotes the initial size of the above two data types and also the minimum attached deposit for registering a new user. Actually, the storage usage denoted by `INIT_ACCOUNT_WITH_ONE_KEY_STORAGE` is inaccurate.

The following test code's output shows that the value of `INIT_ACCOUNT_WITH_ONE_KEY_STORAGE` is smaller than the real storage usage for registering a user. In this case, the contract itself needs to pay for a part of the users' storage usage.

[`Test Code`]:

```
// As needed, we change the scope of the `AccountV2`'s function `from_user` and
field `keys`  from `public(crate)` to `public`
use asset_manager::{account_deposit::*, types::*};

fn main() {
    let initial_storage = env::storage_usage() as u128;

    let alice = AccountId::try_from("alice.near".to_string()).unwrap();
    let mut user_record = AccountV2::from_user(&alice);

    let orderly_key: PublicKey =
"ed25519:6E8sCci9badyRkXb3JoRpBj5p8C6Tw41ELDZoiihKEtp".parse().unwrap();
    user_record.keys.insert(&orderly_key, &TradingKey::default());

    let mut user_ledger: UnorderedMap<AccountId, VAccount> =
UnorderedMap::new(StorageKey::CurrentUsers);
    user_ledger.insert(&alice, &VAccount::V2(user_record));

    let current_storage = env::storage_usage() as u128;
    let storage_diff = current_storage - initial_storage;

    println!("Calculating new user storage cost by calling SDK:");
    println!("initial storage: {}, current storage: {}, storage cost: {}",
initial_storage, current_storage, storage_diff);

    println!("Calculating new user storage cost by manual byte math:");
    println!("storage cost: {}", INIT_ACCOUNT_WITH_ONE_KEY_STORAGE);
}
```

[`Output`]:

Calculating new user storage cost by calling SDK:

initial storage: 307200, current storage: 307966, storage cost: 766

Calculating new user storage cost by manual byte math:

storage cost: 358

Recommendation

We recommend using the `env.storage_usage()` to calculate the storage usage.

Alleviation

[`Certik`] - The Orderly finance has implemented a function that enforces that a sufficient amount of Near has been passed during a storage write operation with the invariant defined by number of current bytes times the storage cost per byte.

SRC-01 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization / Privilege	Major	src/approvers.rs (11a5ae159463086741141083ae86e30e80fad69a): 33, 44, 55, 66, 78, 102, 140, 169, 181, 193, 223, 253; src/contract.rs (11a5ae159463086741141083ae86e30e80fad69a): 599; src/operator.rs (11a5ae159463086741141083ae86e30e80fad69a): 25; src/owner.rs (11a5ae159463086741141083ae86e30e80fad69a): 23, 40, 67, 90, 114, 135, 152, 168, 184, 202, 216, 266, 291, 313, 338	Mitigated

Description

Smart contracts on the Near blockchain are deployed in accounts backed by regular key pairs. Therefore, the overall security of a smart contract is related to the proper use and safekeeping of such key pairs.

Trading users deposit a wide range of assets used for trading into the `Asset Manager` contract. As a result, the contract locks up a significant amount of funds. Therefore, any compromise of the `Asset Manager` contract account could allow a hacker to use this power to steal funds that would otherwise remain in the custody of the contract.

Further, the following roles are also sensitive as they contain significant authority over core functionality.

In the file, `[contract.rs]`, the role `[fee_collector]` has authority over the following functions:

- `fee_collector_withdraw()`

In the file, `[owner.rs]`, the role `[owner]` has authority over the following functions:

- `set_approver_request_possible()`
- `set_approver()`
- `set_owner()`
- `set_operator_manager()`
- `set_only_authorized_access()`
- `add_authorized_account()`
- `remove_authorized_account()`
- `set_withdraw_verify_key()`
- `set_trade_verify_key()`
- `set_symbol_listed()`
- `set_token_listed()`
- `set_token_delisted()`
- `remove_listed_symbol()`

- `set_fee_collector()`

In the file, `[Operator.rs]` the role, `[operator]` has authority over the following functions:

- `operator_execute_action()` - this allows for privileged access to uploading trades or operator withdrawing.

In the file, `[Approver.rs]` the role `[approver]` has authority over the following functions:

- `approve_owner()`
- `approve_operator()`
- `approve_fee_collector()`
- `approve_new_approver()`
- `approve_approver_request_persmission()`
- `approve_symbol_listing()`
- `approve_symbol_delisting()`
- `approve_token_listing()`
- `approve_token_delisting()`
- `approve_withdraw_verify_key()`
- `approve_trade_verify_key()`
- `approve_autorized_access()`

Recommendation

We recommend that the team make efforts to restrict access to the private key of the `Asset Manager` contract's accounts. A strategy of combining a time-lock and a multi-signature wallet can be used to prevent a single point of failure due to a private key compromise. In addition, in case of contract upgrades, the team should be transparent and notify the community in advance whenever they plan to migrate to a new implementation contract.

Further, the team should be aware of which accounts have access to the priveleged roles listed above.

Alleviation

[`CertiK`]: The Orderly Network team implemented the Multi-sign mechanism for approving a request. The functions accessed by the following roles, `owner`, `operator`, and `fee collector`, are still fully controlled by a single central account. However, due to the nature of these contracts some degree of centralization will remain.

SRC-02 | MISSING INPUT VALIDATION

Category	Severity	Location	Status
Logical Issue	Minor	src/contract.rs (11a5ae159463086741141083ae86e30e80fad69a): 112~123, 176~188; src/owner.rs (11a5ae159463086741141083ae86e30e80fad69a): 43~48, 70, 93~98, 341~346, 408~432	Resolved

Description

Per the specifications in `owner.rs`, the contract itself nor existing approvers are able to be an owner, an operator manager, an approver or a fee collector. However, the two initialization functions `new()` and `migrate()` of the `asset manager` contract do not check whether the given account is the `asset manager` contract account itself.

Recommendation

We recommend adding a check within the `new()` and `migrate()` functions to prevent the `asset manager` contract or the approvers from being assigned to the four roles mentioned above.

Alleviation

[`CertiK`]: The Orderly Network team added check codes within function `new()` to prevent the `asset manager` contract from being an owner/operator manager/approver, and prevent the `approver` from being an owner/operator manager. The arguments of function `migrate()` are removed, and no check codes are needed.

STO-01 MISSING CROSS ACCOUNT TRANSFER CALLBACK FUNCTION

Category	Severity	Location	Status
Logical Issue	Medium	src/storage_management.rs (03554f17f43b8f11b5acbd11cc8b7819ed9fe701): 66, 79, 86, 170, 203	Resolved

Description

In the `Near` contract, the usage of `Promise` means that the contained operation will be executed asynchronously with respect to the called method.

When the above `Promise` at the pointed lines is created and returned, the change in the contract state took already place. However, no callback is specified for such `Promises` so the new state is not reverted in the case in which the `Promise` fails.

The pointed codes at L66 within the function `storage_deposit()` creates the `Promise` that transfers `NEAR` refunds. However, if the transfer of the `NEAR` fails, there is no code logic to send an error message to users. In this case, the `NEAR` refunds still remain in the `asset manager` contract and the users' recorded storage fee is not included the part of the refund, however, users are not aware.

A similar issue also exists within functions, `storage_withdraw()` and `storage_unregister()`.

Recommendation

We recommend configuring the callback function to handle transfer failure and sending an error message to notify users that the refund is failed. The potential solution is reverting the contract state when the `NEAR` transfer is failed in the callback function.

Alleviation

[`CertiK`]: The Orderly Network team refactored the codes to make sure only predecessor accounts receive refunds and withdrawals in NEAR.

CON-02 | USELESS CONVERSION

Category	Severity	Location	Status
Language Specific	Informational	src/contract.rs (11a5ae159463086741141083ae86e30e80fad69a): 618	Resolved

Description

It's unnecessary to convert the variable `amount` to the same type.

Recommendation

We recommend removing the type conversion operation.

Alleviation

[`CertiK`]: The Orderly Network team has deleted the conversion operation `U128::from()` .

CON-03 | INCORRECT ERROR MESSAGE

Category	Severity	Location	Status
Logical Issue	Informational	src/contract.rs (11a5ae159463086741141083ae86e30e80fad69a): 308	Resolved

Description

In the function `user_request_withdraw()`, the error message is given by the following lines of code:

```
let key = user_token_to_key(&user, &token);
  let balance = self
      .user_token_balances
      .get(&key)
      .unwrap_or_else(|| env::panic_str("Insufficient token balance"));
```

However, the error results if the key does not exist.

Recommendation

We recommend updating the error message according to reflect the fact the token balance does not exist.

Alleviation

[`CertiK`]: The Orderly Network team corrected the error message.

CON-04 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Coding Style	Informational	src/contract.rs (11a5ae159463086741141083ae86e30e80fad69a): 718	Resolved

Description

In the function `on_fee_collector_withdraw_complete`, the linked statement should emit an events to notify the transfer.

Recommendation

We recommend adding an events for transfer functions, and emitting them after the linked statement.

Alleviation

[`CertiK`]: The Orderly Network team added the codes for emitting event if the transfer succeeds.

OWN-02 | UNUSED CODE

Category	Severity	Location	Status
Coding Style	Informational	src/owner.rs (11a5ae159463086741141083ae86e30e80fad69a): 400	Resolved

Description

The function `assert_self` is defined but is unused.

Recommendation

We recommend removing the unused code.

Alleviation

[`CertiK`]: The Orderly Network team deleted the unused code.

SRC-03 | NEEDLESS BORROW

Category	Severity	Location	Status
Language Specific	Informational	src/approvers.rs (11a5ae159463086741141083ae86e30e80fad69a): 358; src/contract.rs (11a5ae159463086741141083ae86e30e80fad69a): 492, 532; src/contract_utils.rs (11a5ae1594630867411410 83ae86e30e80fad69a): 176, 181, 183; src/operator.rs (11a5ae159 463086741141083ae86e30e80fad69a): 344, 430, 434	Resolved

Description

The expressions highlighted borrow reference with operator `&`, which will be dereferenced by the compiler immediately. For more information, we invite the team to check this reference.

Recommendation

We recommend fixing the needless borrow issue in the current codebase to conform to the Rust coding practices.

Alleviation

[`CertiK`]: The Orderly Network has deleted the excessive borrow.

SRC-04 | UNNECESSARY `let` BINDING

Category	Severity	Location	Status
Language Specific	Informational	src/contract_utils.rs (11a5ae159463086741141083ae86e30e80fad69a): 64~69; src/operator.rs (11a5ae159463086741141083ae86e30e80fad69a): 129~133	Resolved

Description

Variable assignment is extraneous when returning the result of a `let` binding from a block. Removing the `let` binding would make the code follow the rust coding conventions.

For example:

```
64        let public_key = hex::decode(public_key_hex)
65            .ok()
66            .filter(|key| key.len() == 64)
67            .unwrap_or_else(|| env::panic_str("invalid public key"));
68
69        public_key
```

can be

```
64    hex::decode(public_key_hex)
65            .ok()
66            .filter(|key| key.len() == 64)
67            .unwrap_or_else(|| env::panic_str("invalid public key"))
```

Recommendation

Consider returning the expression directly.

Alleviation

[`CertiK`]: The Orderly Network team has deprecated the unnecessary `let` binding.

SRC-05 | ERROR MESSAGE

Category	Severity	Location	Status
Logical Issue	Informational	src/contract.rs (11a5ae159463086741141083ae86e30e80fad69a): 601~603; src/operator.rs (11a5ae159463086741141083ae86e30e80fad69a): 250	Resolved

Description

File: contract.rs Line #602

In the function `fee_collector_withdraw`, the error message states the following:

```
601         if signer_account != self.fee_collector {
602             env::panic_str("caller should be owner or fee collector");
603         }
```

However, the function only requires that the signer is the fee collector.

Recommendation

We recommend updating the error message to reflect the condition accordingly.

SRC-06 | POTENTIAL OVERFLOW

Category	Severity	Location	Status
Mathematical Operations	Informational	src/approvers.rs (11a5ae159463086741141083ae86e30e80fad69a): 68~70, 298~300, 366, 375; src/types.rs (11a5ae159463086741141083ae86e30e80fad69a): 222~225	Resolved

Description

The value type of `approver_id` is `u8` and its maximum value is `255`. This means that the maximum number of approvers is `255`.

In Rust, integer literals whose type is unconstrained will default to `i32`. So the type of the constant `1` used at L366 and L375 is `i32` and the maximum left shift is `31` bits. An overflow can occur if the left shift exceeds `31` bits.

The value of `approver_id` indicates the left shift bits number. Its max value is `255` which is greater than `31`. So an overflow may occur.

Recommendation

We advise the client to limit the number of approvers to avoid the overflow.

Alleviation

[`CertiK`]: The Orderly Network team deleted the codes that caused the problem.

OPTIMIZATIONS | ORDERLY NETWORK

ID	Title	Category	Severity	Status
APP-03	Unnecessary Memory Copy Operations	Volatile Code	Optimization	Resolved
CON-05	Duplicate Functions	Logical Issue	Optimization	Resolved
OWN-03	Test Function Only	Coding Style	Optimization	Resolved
OWN-04	Redundant Check	Volatile Code	Optimization	Resolved
PRO-01	Typos	Coding Style	Optimization	Resolved
SRC-07	Redundant Clone	Gas Optimization	Optimization	Resolved
SRC-08	Duplicate Variable Assignment	Gas Optimization	Optimization	Resolved

APP-03 | UNNECESSARY MEMORY COPY OPERATIONS

Category	Severity	Location	Status
Volatile Code	Optimization	src/approvers.rs (11a5ae159463086741141083ae86e30e80fad69a): 118~122	Resolved

Description

The values of `symbol` and `account_ids` are `String` type which indicates that their values actually stored in heap not stack. The calling of the `clone()` function causes a deep copy of heap data which is a time consuming operation. The linked statement is just used to ensure that `passed_symbol/passed_account_ids` and `symbol/account_ids` have the same value. It is better that using the the dereference operator `*` to avoid the deep copy of heap data.

Recommendation

We advise refactoring the linked statement as below:

```
assert_eq!(
    (passed_symbol, passed_account_ids),
    (*symbol, *account_ids,
    "Please check symbol and particular account ids which you approve"
);
```

Alleviation

[`CertiK`]: The Orderly Network team has removed the `clone` operation on `symbol` and `account_ids`.

CON-05 | DUPLICATE FUNCTIONS

Category	Severity	Location	Status
Logical Issue	Optimization	src/contract.rs (11a5ae159463086741141083ae86e30e80fad69a) : 878~889	Resolved

Description

The linked two functions, `assert_authorized()` and `is_authorized()`, have similar functionality with different output values.

Recommendation

We recommend refactoring these two functions into one function that can handle both possible outputs to reduce the size of the contract.

Alleviation

[`CertiK`]: The Orderly Network team has deleted the function `is_authorized()`.

OWN-03 | TEST FUNCTION ONLY

Category	Severity	Location	Status
Coding Style	Optimization	src/owner.rs (11a5ae159463086741141083ae86e30e80fad69a): 202	Resolved

Description

The function `set_max_operator_downtime()` is only used for testing.

Recommendation

We recommend the usage of the `#[cfg(test)]` annotation to exclude test code from the production build and save storage in the deployment phase.

OWN-04 | REDUNDANT CHECK

Category	Severity	Location	Status
Volatile Code	Optimization	src/owner.rs (11a5ae159463086741141083ae86e30e80fad69a): 43~48, 50~52	Resolved

Description

The statements L50-L52 are used to restrict existing approvers from becoming added as a new approvers. However, the `assert_privilages()` function, called before L50, reverts if given approver is an existing approver.

Recommendation

We advise removing the redundant check.

Alleviation

[`CertiK`]: The Orderly Network team has deleted the redundant code for checking duplicated approver.

PRO-01 | TYPOS

Category	Severity	Location	Status
Coding Style	Optimization	src/owner.rs (11a5ae159463086741141083ae86e30e80fad69a): 408; src/contract_utils.rs (6b2bfc460405ee1c1797788b2ec09c092e67c1bc): 60	Resolved

Description

The following lines of code have a typos:

- In Owner.rs

```
pub(crate) fn assert_privilages
```

- In Approver.rs

```
pub fn approve_approver_request_persmission(&mut self)
```

Recommendation

We recommend updating the typos to the following:

- In Owner.rs

```
pub(crate) fn assert_privileges
```

- In Approver.rs

```
pub fn approve_approver_request_permission(&mut self)
```

Alleviation

[`CertiK`]: The Orderly Network team has corrected the typos.

SRC-07 | REDUNDANT CLONE

Category	Severity	Location	Status
Gas Optimization	Optimization	src/contract.rs (11a5ae159463086741141083ae86e30e80fad69a): 765, 766, 783, 784, 826, 827, 835, 836; src/operator.rs (11a5ae159463086741141083ae86e30e80fad69a): 88, 89	Resolved

Description

It is unnecessary to use `clone` as this value is dropped without further use.

Recommendation

We recommend removing the unnecessary `clone()` calls.

Alleviation

[`CertiK`]: The team resolved the finding in commit hash ebc36db0dc1fc331b51636356de14febc2e0cec5. Logic regarding to mentioned lines in `contract.rs` is refactored, so the `clone()` are need now. Lines in `operator.rs` are updated to remove `clone()` .

SRC-08 | DUPLICATE VARIABLE ASSIGNMENT

Category	Severity	Location	Status
Gas Optimization	Optimization	src/approvers.rs (11a5ae159463086741141083ae86e30e80fad69a): 21; src/contract.rs (11a5ae159463086741141083ae86e30e80fad69a): 305~309, 312~316	Resolved

Description

File: contract.rs Line #305, #312

The variable `balance` is already assigned and checked in Line #305.

```
305         let balance = self
306             .user_token_balances
307             .get(&key)
308             .unwrap_or_else(|| env::panic_str("Insufficient token balance"));
309
310         if self.get_withdraw_unlock_duration_nanos() == 0 {
311             // In this case we will simply check the balance and if it is bigger or equal then amount will perform the transfer
312             let balance = self
313                 .user_token_balances
314                 .get(&user_token_to_key(&user, &token))
315                 .unwrap_or_default();
```

It's unnecessary to redefine it in Line #312.

File: approver.rs Line #21

It's unnecessary to retrieve the approver of caller again, `approver_id` could be reused in Line #21.

```
18          let approver_id = self.approvers_ids.get(&caller);
19
20          assert!(
21              self.approvers_ids.get(&caller).is_some(),
22              "Caller is not set as an approver"
23          );
```

Recommendation

We recommend refactoring this code so that the `balance` is only assigned once.

Alleviation

[`CertiK`]: The Orderly Network team removed the duplicated codes.

APPENDIX | ORDERLY NETWORK

Finding Categories

Categories	Description
Centralization / Privilege	Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.
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.
Mathematical Operations	Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.
Logical Issue	Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.
Language Specific	Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.
Coding Style	Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

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.