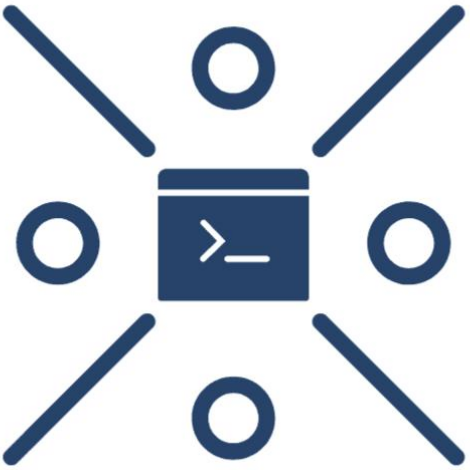

~# whoami

Report from the Security Assessment of Asset Manager

Prepared for:
Orderly Network

Version history

Version	Date	Author	Description
1.0	25/08/2023	Jakub Heba	Draft version
2.0	10/10/2023	Jakub Heba	Final version

Table of Content

1. Introduction

From August 14 to August 25, security assessment of the "Asset Manager" Smart Contract was carried out in order to detect vulnerabilities and bad practices applied in the course of contract creation.

The report is based on the vulnerabilities found in the course of the work. The document is intended for the internal needs of Orderly Network.

The report does not take into account the vulnerabilities that arose after the test completion date.

2. About auditor

Jakub Heba is a cybersecurity expert with over six years of experience in the industry. For two years associated with blockchain technology as a Smart Contract and Blockchain auditor. He has conducted over 25 audits of various protocols, mostly related to Decentralized Finances. He specializes in the security of contracts written in Rust, such as CosmWasm, NEAR, Substrate or MultiversX (Elrond), as well as has a deep technical understanding of EVM and Solidity. He participated in assessments testing low-level aspects of blockchain technology, such as finality proof verifications, serialization libraries, as well as implementations of bridges between many different ecosystems. He has experience in auditing Layer 1 Blockchains written in Rust and MOVE. Before moving to Web3, he was a Lead Security Researcher and Penetration Tester managing a team of up to 10 engineers. He also specialized in low-level binary exploitation in both UNIX and Windows environments. Holder of OSCP, OSCE and Lead ISO27001 Auditor certificates.

3. About client

Orderly Network is a decentralized orderbook protocol designed for DeFi developers and traders in the web3 domain, aiming to provide accelerated trading with low fees and robust liquidity. Combining orderbook-based trading infrastructure with a liquidity layer, Orderly enables spot and perpetual futures orderbooks without the need for a front end.

Its DEX white-label solution and plug-and-play experience promote cost-effective and flexible trading application development. The grand vision is to create an omnichain protocol, connecting traders across EVM and non-EVM chains.

The product offerings include:

- **Spot Trading -** Fast and secure buying and selling of cryptocurrencies.
- **Perpetual Futures Trading -** A derivatives trading form that allows trading on future prices without asset ownership.
- **Community Pool -** A decentralized platform for lending, designed to boost liquidity across markets.
- **Token Auction -** An infrastructure for secure token listing and market enhancement.

4. Disclaimer

This report reflects a rigorous security audit conducted on the specified codebase, utilizing industry-leading methodologies in web3 technology, including blockchain and smart contract security. While the assessment was carried out with the utmost care and proficiency, it is essential to recognize that no security audit can guarantee 100% immunity from vulnerabilities or risks.

Security is a dynamic and ever-evolving field. Even with substantial expertise, it is impossible to predict or uncover all future vulnerabilities. Regular and varied security assessments should be performed throughout the code development lifecycle, and engaging different auditors is advisable to obtain a more robust security posture.

This audit is limited to the defined scope and does not encompass parts of the system or third-party components not explicitly included. It does not provide legal assurance of compliance with regulations or standards, and the client remains responsible for implementing recommendations and continuous security practices.

The objective has been to provide cutting-edge services, but the intrinsic complexity of the technological landscape means that a guarantee of absolute safety cannot be provided. Continuous engagement with security assessments and adherence to evolving best practices are strongly encouraged to maintain a resilient security environment.

5. Scope

The scope covered by the security assessment specifies that the `asset-manager` contract will be audited, the code of which has been shared on the GitLab platform with the 6f015925aaf212430e862fdf169cba95fb15d8bc commit SHA hash.

The scope of testing includes the entire code repository. No additional internal documentation was provided - only documentation on the project website was available during the audit.

6. Vulnerabilities classification

All vulnerabilities described in the report have been thoroughly classified in terms of the risk they generate in relation to the security of the contract implementation. Depending on where they occur, their rating can be estimated on the basis of different methodologies.

In most cases, the estimation is done by summarizing the impact of the vulnerability and its likelihood of occurrence. The table below presents a simplified risk determination model for individual calculations.

		Impact		
	Severity	**High**	**Medium**	**Low**
Likelihood	**High**	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low

Vulnerabilities that do not have a direct security impact, but may affect overall code quality, as well as open doors for other potential vulnerabilities, are classified as **INFORMATIVE**.

7. Security assessment results

The table below presents all the vulnerabilities found in the course of the work, along with an appropriately estimated level of risk they entail.

Id	Vulnerability	Risk	Status
1	u64 type might be problematic while used by JavaScript	LOW	Acknowledged
2	Removal of Operator key and withdrawals require approval from owner	LOW	Acknowledged
3	Binary size could be reduced	LOW	Resolved
4	Check in `ft_on_transfer` function is redundant	INFORMATIVE	Resolved
5	Missing validation for `user_swap_balance` inside `process_validated_trade` function	INFORMATIVE	Acknowledged
6	Unusual closure in `TokenBalance::deposit`	INFORMATIVE	Resolved
7	`OperationController` type is excessive	INFORMATIVE	Acknowledged
8	Assertions in new function are redundant	INFORMATIVE	Resolved
9	TODO comments in the codebase	INFORMATIVE	Resolved
10	The order of calls in the `add_authorized_account` function can be more optimized	INFORMATIVE	Resolved

1. [Low] u64 type might be problematic while used by JavaScript

Location
`asset-manager/src/types/trade_types.rs:20-36`

Technical Description

It has been noted that for some data structures, defined parameter types can be problematic when used by another programming language, such as JavaScript. As an example, JS can handle numbers up to 2**52 with precision. Numbers beyond that will lose it, leading to start failing at some point.

Such a vulnerability has been noticed in the `TradeUpload` structure, however, it can occur globally, wherever the user can send a value of type u64 or greater.

To demonstrate, it would be necessary to run `python` and calculate a value for 2**60 as asn example. Then we need to copy those numbers. After that, node should be runned, and assignment of this number to a variable has to be made. It WILL NOT error, but if we display the variable, the number will actually be different, because precision was lost.

Vulnerable struct:

```rust
#[derive(Debug, Serialize, Deserialize, Clone, PartialEq)]
#[serde(crate = "near_sdk::serde")]
pub struct TradeUpload {
    pub trade_id: u64,
    pub match_id: String,
    pub account_id: String,
    pub symbol: String,
    pub side: PurchaseSide,
    pub quantity: String,
    pub price: String,
    pub amount: String,
    pub fee: String,
    pub fee_asset: String,
    pub timestamp: u64,
    pub order_signature: String,
    pub orderly_key: String,
    pub order: serde_json::Value,
    pub broker_fee: U128, // requires 'order.broker_id' if broker_fee > 0
}
```

Recommendation

Structures should use U128 type whenever the numerical string is used. Additionally, the u64 types should be changed to U64 for all instances where the user passes data of this type to the contract.

Status
Acknowledged

2. [Low] Removal of Operator key and withdrawals require approval from owner

Location
`asset-manager/src/operator.rs:193-209`
`asset-manager/src/operator.rs:395-408`

Technical Description

While finding the code, it was noticed that the `execute_key_removal_action` and `execute_withdraw_action` functions require the Operator to consent to the operation, which will be saved as part of the `ApprovalStatus` struct. If the status is `Approve`, the transaction will be executed. If it is `Reject`, it will be rejected.

From the perspective of the code, it is quite logical, however, taking into account the fact that it centralizes the contract quite strongly - you should pay special attention to whether it is not too restrictive an approach. Web3 is by definition oriented towards decentralization. Deviating from this may have negative consequences and affect the credibility of the solution.

Vulnerable functions:

```
    fn execute_withdraw_action(
        &mut self,
        account: AccountId,
        token: AccountId,
        withdraw_id: u64,
        status: ApprovalStatus,
        event_id: u64,
    ) -> PromiseOrValue<()> {
        match status {
            ApprovalStatus::Approve => {
                self.operator_withdraw_approve(account, token, withdraw_id,
event_id)
            }
            ApprovalStatus::Reject => {
                self.operator_reject_withdraw_request(account, token, withdraw_id,
event_id)
            }
        }
    }
    pub(crate) fn execute_key_removal_action(
        &mut self,
        account: AccountId,
        key: PublicKey,
        status: ApprovalStatus,
        remove_key_id: u64,
    ) {
        match status {
            ApprovalStatus::Approve => {
                self.operator_key_removal_approve(account, key, remove_key_id)
            }
            ApprovalStatus::Reject => self.operator_key_removal_reject(account, key,
remove_key_id),
        }
```

Recommendation

It is suggested to thoroughly analyze the pros and cons of introducing centralization in the contract. If possible, as many key functionalities as possible should be decentralized and automated.

Status

Acknowledged

3. [Low] Binary size could be reduced

Location

`asset-manager/Cargo.toml`

Technical Description

The `rlib` crate type in Rust is used to create a Rust library that is statically linked into other Rust crates. When compiling a crate with the `rlib` type, the Rust compiler generates a binary that includes both the compiled Rust code and the metadata necessary for linking and type checking.

It has been noticed that due to the use of the `rlib` library, the size of the `.wasm` file is drastically large. This is due to the presence of a large number of symbols, which makes the build result much "bigger". As this library is not directly used by the `asset-manager`, removing it can significantly reduce the size of the `.wasm` file.

Update: After analyzing the potential pros and cons, as well as presenting suggestions for changes from the developers of Orderly, one of the options that can be carried out to reduce the size of the `.wasm` file is stripping all symbols which are not required for the wasm interface. The benefit will also be that integration tests will still work properly, which would not happen if the `rlib` library was removed.

Content of Cargo.toml:

```
[package]
name = "asset-manager"
version = "0.1.0"
edition = "2021"

[lib]
crate-type = ["cdylib", "rlib"]
```

Recommendation

It is suggested to take one of the two actions described above to eliminate the relatively large `.wasm` file size.

Status

Resolved in ac08bec8caa76456ae3f1dc671b77f8dad20c612

4. [Informative] Check in `ft_on_transfer` function is redundant

Location
`asset-manager/src/contract.rs:1174-1176`

Technical Description

It was found that `ft_on_transfer` function unnecessarily checks if the `token` used equals NEAR. The "near" AccountId is reserved and is used to create new accounts. It is not a token contract. So in practice it is impossible that this check returns false anytime. It may therefore be omitted.

Vulnerable function:

```
#[near_bindgen]
impl FungibleTokenReceiver for Contract {
    /// See [`Contract::user_deposit_native_token`].
    fn ft_on_transfer(
        &mut self,
        sender_id: AccountId,
        amount: U128,
        msg: String,
    ) -> PromiseOrValue<U128> {
        self.paused_operations
            .assert_operation_not_paused(ControlledOperations::FtOnTransfer);
        self.assert_authorized(&sender_id);

        let token = env::predecessor_account_id();
        if token.as_str() == NATIVE_TOKEN {
            env::panic_str("invalid token account");
        }
        if !self.tokens_whitelist.contains(&token) {
            env::panic_str("this token is not allowed");
        }

[..]
```

Recommendation

As far as this check is redundant, it could be removed from the mentioned function.

Status
Resolved in 4f0a992bf6e0845dc2fa4c959f37d61fa4543dde

5. [Informative] Missing validation for `user_swap_balance` inside `process_validated_trade` function

Location
`asset-manager/src/operator.rs:726-733`

Technical Description

The `process_validated_trade` function processes trades, updating user balances, fees and possible broker commission. It manipulates various aspects of the user's account and writes them back into the user registry.

It was noted that during `self.user_swap_balance` execution, the function increases the balance for the `token_deposit` token AccountId without any checks. It is assumed that only valid transactions will be sent to the contract from the off-chain service, but the contract simply believes that those will always be correct.

Vulnerable function:

```
   pub(crate) fn process_validated_trade(&mut self, trade: &TradeUpload) {
        let token_pair = self.symbol_tokens_whitelist.get(&trade.symbol).unwrap();
        let trade_amount = u128::from_str(&trade.amount).unwrap();
        let trade_quantity = u128::from_str(&trade.quantity).unwrap();

        let (token_transfer, token_transfer_amount, token_deposit,
token_deposit_amount) =
            if trade.side == PurchaseSide::Buy {
                (&token_pair.1, trade_amount, &token_pair.0, trade_quantity)
            } else {
                (&token_pair.0, trade_quantity, &token_pair.1, trade_amount)
            };

        let trade_account = AccountId::from_str(&trade.account_id).unwrap();
        let mut user_record =
self.user_ledger.get_and_unwrap_handled(&trade_account);
        self.user_swap_balance(
            &mut user_record,
            &trade_account,
            token_deposit,
            token_deposit_amount,
            token_transfer,
            token_transfer_amount,
        );

[..]
```

Recommendation

Due to the fact that the audit scope did not include offchain services, it is important to make sure that the relevant checks are performed outside the smart contract. If possible, proper validation should be implemented to ensure the integrity and legitimacy of `process_validated_trade` function calls.

Status
Acknowledged

6. [Informative] Unusual closure in `TokenBalance::deposit`

Location
`asset-manager/src/token_balance.rs:28-35`

Technical Description

It was noticed that the deposit function is not a standard function, but the closure defined as part of the `TokenBalance` implementation. This is quite unusual, and while it poses no security risk in itself, using Rust's advanced concepts for such simple operations is not advisable. In addition, closures use a bit more computational power compared to regular functions due to the higher use of the memory heap.

Vulnerable function:

```
impl TokenBalance {
   pub fn deposit(&mut self, amount: u128) {
        (|| {
            self.balance = self.balance.checked_add(amount)?;
            self.balance.checked_add(self.pending_transfer)?;
            Some(())
        })()
        .unwrap_or_else(|| env::panic_str(ERR_TOKEN_AMOUNT_OVERFLOW))
   }

[..]
```

Recommendation

It is recommended to analyze the possibility of turning closure into a regular, standard function.

Status
Resolved in b7268eb9d6fda2a9aa3b0fb09edbe111b45782a2

7. [Informative] `OperationController` type is excessive

Location
`asset-manager/src/operation_control.rs:112-114`

Technical Description

The `OperationController` is using bit flags to control which operation is currently paused. However, it uses the u64 type for this. Taking into account that there are 14 operations that could be paused, the u64 seems excessive.

Vulnerable function:

```
pub struct OperationController64 {
   bit_set: u64,
}

pub fn operations() -> &'static [ControlledOperations; 14] {
       static CONTROLLED_OPERATIONS: [ControlledOperations; 14] = [
           ControlledOperations::CreateUserRecord,
           ControlledOperations::UserRequestWithdraw,
           ControlledOperations::UserAnnounceKey,
           ControlledOperations::UserRequestSetTradingKey,
           ControlledOperations::UserDepositNativeToken,
           ControlledOperations::FeeCollectorWithdraw,
           ControlledOperations::FtOnTransfer,
           ControlledOperations::OperatorRejectWithdrawRequest,
           ControlledOperations::OperatorWithdrawApprove,
           ControlledOperations::OperatorTradesUpload,
           ControlledOperations::CreateOnchainOrder,
           ControlledOperations::OperatorUpdateOnchainOrder,
           ControlledOperations::OperatorEventUpload,
           ControlledOperations::DepositAndCreateOrder,
       ];

       &CONTROLLED_OPERATIONS
   }
```

Recommendation

It is suggested to adjust the type of the `bit_set` variable to your actual needs. In this case, u32 will be sufficient.

Status
Acknowledged

8. [Informative] Assertions in new function are redundant

Location
`asset-manager/src/contract.rs:192`
`asset-manager/src/contract.rs:218`

Technical Description

It was noted that the `new` function implements the assert macro to verify that state has not already been initialized. However, this function comes with the `#[init]` macro. According to the NEAR documentation, this macro performs the same action before calling the function, so duplicating the check is unnecessary.

Ref. https://docs.near.org/bos/api/state#stateinit

In addition, when verifying that `approvers_ids` does not contain any system roles, `operator_manager` is checked twice, which is unnecessary.

Vulnerable function:

```
#[near_bindgen]
impl Contract {

    #[init]
    pub fn new(
        owner: AccountId,
        operator_manager: AccountId,
        fee_collector: AccountId,
        symbol_tokens_whitelist: Vec<(String, (AccountId, AccountId))>,
        tokens_whitelist: Vec<AccountId>,
        trade_verify_key: Base64VecU8,
        only_authorized_access: bool,
        authorized_users: Vec<AccountId>,
        approvers_ids: Vec<AccountId>,
        removal_verify_key: Base64VecU8,
        onchain_order_verify_key: Base64VecU8,
        event_upload_verify_key: Base64VecU8,
        operator_gas_fee_collector: AccountId,
        onchain_order_fees: Vec<(AccountId, U128)>,
        perp_upload_verify_key: Base64VecU8,
        market_data_verify_key: Base64VecU8,
        futures_fee_collector: AccountId,
    ) -> Self {
        assert!(!env::state_exists(), "Already initialized");
        assert!(
            owner != operator_manager
                && owner != fee_collector
                && operator_manager != fee_collector,
            "Roles should be separated"
        );

[..]

assert!(
            !approvers_ids.contains(&operator_manager)
```

```
                && !approvers_ids.contains(&owner)
                && !approvers_ids.contains(&operator_manager)
                && !approvers_ids.contains(&fee_collector),
            "Approvers should not be similar with roles"
        );
[..]
```

Recommendation

It is recommended to remove redundant checks in the new function.

Status

Resolved in c5c7928d74f6bbd9e87042a6da85c3f43ab00be8

9. [Informative] TODO comments in the codebase

Location

`asset-manager/src/contract.rs:682`
`asset-manager/src/types/trade_types.rs:17`

Technical Description

It has been noticed that there are comments marked "TODO" in several places in the codebase. While this is not a direct security threat, this type of information can be dangerous from the point of view of creating potential attack vectors by an attacker, and does not look professional if the code is in a production version.

Vulnerable code:

```
// TODO cover with integration tests
    #[payable]
    pub fn operator_gas_fee_withdraw(&mut self, token: AccountId, amount: U128) {

[..]

// TODO change numeric String values to U128
#[derive(Debug, Serialize, Deserialize, Clone, PartialEq)]
#[serde(crate = "near_sdk::serde")]
pub struct TradeUpload {
    pub trade_id: u64,

[..]
```

Recommendation

It is suggested to remove TODO comments and implement the functionalities described by them.

Status

Resolved in 484e314e6cf7f027b756af6b4a07e54c57f71aa3

10. [Informative] The order of calls in the `add_authorized_account` function can be more optimized

Location

`asset-manager/src/contract.rs:682`
`asset-manager/src/types/trade_types.rs:17`

Technical Description

The `add_authorized_account` function, as the name suggests, is used to add authorized accounts. In its content, it verifies whether it was called by the owner of the contract, as well as checks whether the indicated user is not already added as an authorized one.

When attempting to insert a user into a set, the `is_present` variable is set to true or false. Then, its verification is carried out and depending on the content - panic is triggered or not.

From an optimization point of view, a better order would be to verify if the user is already in the set and return an error based on that, or allow an insert into it.

Additionally, the name of the variable is misleading. It is receiving True value, if value is **not** present. So in fact, `is_not_present` is more suitable.

Vulnerable code:

```
   /// Add authorized account so it will be NOT limited when the authorized feature
is enabled
   ///
   /// # Transaction panics
   /// * If caller is not the `owner`.
   /// * If the state is set to the same parameter
   pub fn add_authorized_account(&mut self, account: AccountId) {
       self.assert_owner();

       let is_present = self.authorized_users.insert(&account);

       if !is_present {
           env::panic_str(
               format!("{} has been already added to the authorized list",
account).as_str(),
           );
       }
   }
```

Recommendation

It is suggested to adjust the order of verification and insert to the set in order to save gas.

Status

Resolved in eed0d73ba073333c71c9593f25f5054e85a1bdde