verichains

SECURITY AUDIT OF

DAOX SUBSCRIPTION

Public Report

Jan 17, 2025

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Sui Blockchain	Sui is an innovative, decentralized Layer 1 blockchain that redefines asset ownership. Sui Move feels like a paradigm change in web3 development. Treating objects as 1st class citizens brings composability to a whole new level. Polymedia. We are thrilled to be building on Sui.
Sui Object	The basic unit of storage in Sui is object. In contrast to many other blockchains where storage is centered around accounts and each account contains a key-value store, Sui's storage is centered around objects.
Move	Move is a new programming language that implements all the transactions on the Aptos/Sui blockchain.
Move Module	A Move module defines the rules for updating the global state of the Aptos/Sui blockchain. In the Aptos/Sui protocol, a Move module is a smart contract.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Jan 17, 2025. We would like to thank the VirtualDaos for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the DAOX Subscription. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified some vulnerable issues in the smart contracts code. The VirtualDaos team has resolved the issues and updated the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About DAOX Subscription

VirtualDaos is a platform designed to enable individuals to create and manage their own DAOs with ease. Often compared to "Shopify for DAOs," it incorporates an intelligent approach where AI agents manage complexities such as governance and community engagement.

With VirtualDaos subscription contract, users can lock their SUI tokens in the pools and receive DAOX tokens in return based on a predefined subscription rate.

1.2. Audit Scope

This audit focused on identifying security flaws in code and the design of the smart contracts of DAOX Subscription.

It was conducted on commit `441ff97e2cb5708fa1857de3eaf0e3acd5749036` from git repository link: *https://github.com/virtualdaos/contract-sui/*.

The following files were made available in the course of the review, which the corresponding final commit was provided as `bad24443d237957a061e24851dd4094a5f525876`.

SHA256 Sum	File
`f17e202fdf74e6842dfeda7be1bce2d75d0baceb90d1708144fa4f6a3679d2c4`	`sources/coin.move`
`4b98b2d902f31ad086c718b15912c5ea6cf582150dbbb4ad8e9152a0e85142f8`	`sources/subscription.move`

1.3. Audit Methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that were considered during the audit of the smart contract:

- Numerical precision errors
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- Gas Usage, Gas Limit and Loops

- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

VirtualDaos acknowledges that the security services provided by Verichains, are conducted to the best of their professional abilities but cannot guarantee 100% coverage of all security vulnerabilities. VirtualDaos understands and accepts that despite rigorous auditing, certain vulnerabilities may remain undetected. Therefore, VirtualDaos agrees that Verichains shall not be held responsible or liable, and shall not be charged for any hacking incidents that occur due to security vulnerabilities not identified during the audit process.

1.5. Acceptance Minute

This final report served by Verichains to the VirtualDaos will be considered an Acceptance Minute. Within 7 days, if no any further responses or reports is received from the VirtualDaos, the final report will be considered fully accepted by the VirtualDaos without the signature.

2. AUDIT RESULT

2.1. Overview

The DAOX Subscription was developed using the Move programming language and deployed on the Sui Blockchain.

The contract source code includes the DAOX Token and the Subscription contract.

2.1.1. DAOX Token

The DAOX token is written based on the SUI Coin standard for fungible tokens with the following information.

FIELD	VALUE
Module	"presale::coin"
Name	"DAOX"
Symbol	"DAOX"
Decimals	6

Table 2. DAOX Token Information

The `TreasuryCap<COIN>` object is held by the contract deployer and can be used to control the supply of DAOX tokens.

2.1.2. Subscription Contract

The Subscription contract allows sellers to create their DAOX token pools with configurable parameters such as `claim_time`, `end_time_refund`, `start_time`, `end_time`, `soft_cap`, `hard_cap`, `token_rate`, and `total_tokens`. Any buyer can contribute their SUI tokens to the pool and receive DAOX tokens based on the configured subscription rate (`token_rate`) when the pool ends. The pool owner can deposit or withdraw tokens (both DAOX and SUI) in the pool at any time, so the buyer must trust the pool owner in this case.

If the contributed tokens are less than the `soft_cap`, both the pool creator and buyers can claim back their funds. If the contributed tokens are greater than the `hard_cap`, the remaining tokens will be refunded to the buyers.

2.2. Findings

During the audit process, the audit team found some issues and recommendations in the given version of DAOX Subscription.

#	Issue	Severity	Status
1	Incorrect allocation amount calculation	HIGH	Fixed

2.2.1. Incorrect allocation amount calculation HIGH

Affected files:

- subscription.move

Look at the `claim_fund` function. When the `total_contributions` is greater than the `soft_cap` and less than the `hard_cap`, the total amount of `COIN` tokens to be distributed should be `total_contributions * token_rate`, and the remaining amount would be refunded to the pool's owner.

```
#[allow(lint(self_transfer))]
public fun claim_fund(pool: &mut SubscriptionPool, clock: &Clock, ctx: &mut TxContext) {
    let current_time = clock.timestamp_ms();

    assert!(ctx.sender() == pool.creator, ENotCreator);
    assert!(pool.is_active && current_time > pool.end_time, EWrongStatus);

    if (pool.total_contributions < pool.soft_cap) {
        // ...
    } else {
        let mut fund_amount = pool.hard_cap;
        let mut refund_amount = 0;
        if (pool.total_contributions < pool.hard_cap) { //NEED TO REVIEW
            fund_amount = pool.total_contributions;
            refund_amount = pool.total_tokens - fund_amount * pool.token_rate;
            let coin = coin::take(
                &mut pool.total_token_balance,
                refund_amount,
                ctx,
            );
            transfer::public_transfer(coin, ctx.sender());
        };
        // ...
    }
}
```

However, in the `claim_token` function, the allocation amount is calculated as `contribution.amount * pool.total_tokens / pool.total_contributions`, which is distributed to

contributors based on their share of the total contributions. Nevertheless, the total amount of `COIN` tokens in this case will not be sufficient to cover the total contribution amount, as shown in the formula above.

```
#[allow(lint(self_transfer))]
public fun claim_token(
    pool: &mut SubscriptionPool,
    contribution: &mut Contribution,
    clock: &Clock,
    ctx: &mut TxContext,
) {
    // ...
    if (pool.total_contributions < pool.soft_cap || !pool.is_active) {
        // ...
    } else {
        let allocation =
            (
                contribution.amount * pool.total_tokens
            ) / pool.total_contributions; //INCORRECT ALLOCATION

        let coin = coin::take(
            &mut pool.total_token_balance,
            allocation,
            ctx,
        );

        transfer::public_transfer(coin, ctx.sender());

        let mut refund = 0;

        if (pool.total_contributions > pool.hard_cap) {
            refund =
                (
                    contribution.amount * (
                        pool.total_contributions - pool.hard_cap
                    )
                ) / pool.total_contributions;
            let coin = coin::take(
                &mut pool.total_contribution_balance,
                refund,
                ctx,
            );
            transfer::public_transfer(coin, ctx.sender());
        };
        // ...
    }
}
```

RECOMMENDATION

The allocation logic should be reviewed carefully to handle all possible scenarios. Additionally, in the `create_pool` function, the values of `soft_cap`, `hard_cap`, `token_rate`, and `total_tokens` should be validated to ensure that the total amount of `COIN` tokens to be distributed is calculated accurately.

UPDATES

- **Jan 17, 2025**: This issue has been acknowledged and fixed by the VirtualDaos team. However, the configured parameters when creating pools are still not validated.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Jan 17, 2025*	Public Report	Verichains Lab
1.1	*Jan 17, 2025*	Public Report	Verichains Lab

Table 3. Report versions history