verichains

SECURITY AUDIT OF

AMNIS SMART CONTRACTS

Public Report

Nov 07, 2023

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
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.
Resource Account	A resource account is a developer feature used to manage resources independent of an account managed by a user, specifically publishing modules and automatically signing for transactions.
Aptos Blockchain	Aptos is an innovative public blockchain (proof of stake) developed by former Facebook employees, with a primary focus on delivering high throughput and robust security for smart contracts developed using the Move programming language.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Nov 07, 2023. We would like to thank the Amnis 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 Amnis Smart Contracts. 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 information issues in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Amnis Smart Contracts

Amnis Finance - a Pioneering Liquidity Staking on Aptos. As a foundational component of the Aptos ecosystem, Amnis Finance introduces a secure, user-friendly and innovative liquid staking protocol that empowers users to effortlessly maximize returns on their APT tokens while unlocking their liquidity.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of Amnis Smart Contracts. It was conducted on commit `59525ac8e9c16904b675c740c3df5017088ba637` from git repository link: *https://github.com/tony001242/amnis-contract*.

The latest version is available on commit `09f241418e8d7db14d44713c05306bb8b1fed804` from git repository link: *https://github.com/amnis-finance/amnis-contract*.

SHA256 Sum	File
307031d3b6bb4b59ab53b968704d0d04d27fbd0caad0b49d4c9aa144d92e3c49	Move.toml
19adf86a5d20197a5d72697ff064e2a1075e6a63f822cf9f50471fc96ac33ef9	amapt_token.move
9934c6a463eef5101903d6f484da2ecb1bd5492f80c761b1f15d7209dcb121ae	aptos_governance.move
f6067c52147045ea88841fcba360efcced368d16515c6f66e956a880ac5e6c7f	delegation_manager.move
197262c4ea47a3a471ae8d20576967702dec3fb3bf0a40bc83250721ac53953f	governance.move
cb6ddb07a4a60c906ddedb9ea5d6aba68e938f9021053cc080ccc6b635b402a7	package_manager.move
e0a4a2424d36a709c4f1ad4710b9a7bcf26ef1baf8174f3236a8ca397e27e9bf	router.move
1605cb288e68d6f448333fc29627fa84ad24829a4233b0d5c7637cef884d299c	stapt_token.move
64a6c5a39fcca53596187c02eca1dc6635b4317eee27888d539ea2b3b26aa082	treasury.move
fafef877e3d1404917090dc90700dddaa39f8a7a3f84cef3e35bd7b2a3e29b36	withdrawal.move

Table 1. Hash value of files in the latest version

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 2. Severity levels

1.4. Disclaimer

Amnis 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. Amnis understands and accepts that despite rigorous auditing, certain vulnerabilities may remain undetected. Therefore, Amnis 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.

2. AUDIT RESULT

2.1. Overview

The Amnis Smart Contracts was written in `Move` programming language. The project is built from a number of contracts whose main functionality will be summarized below:

2.1.1. amAPT token

The `amAPT token` module serves as the foundational token within the project. An `amAPT token` is specifically created to represent an `APT` that a user has staked in the project.

2.1.2. Aptos governance

This module empowers users to participate in voting for proposals, while also offering features for locking and releasing tokens.

2.1.3. Delegation Manager

This module handles the core logic associated with user voting.

2.1.4. Governance

Primarily tailored for project parties, this module supports project initiation, updates to account addresses for various roles, whitelist management, voting procedures, adjustments to token lock-up periods, and modifications of reward fee ratios, among other functions.

2.1.5. Package Manager

This module stores all project-related data, ensuring permission checks and maintaining operator data.

2.1.6. Router

The Router module implements features related to staking and withdrawing `amAPT tokens` and `stApt tokens`.

2.1.7. stApt Token

Designed for the project party's platform coin, this module allows users to acquire `stApt tokens` by staking `amAPT tokens`. It provides functionality for initializing tokens, retrieving token prices, depositing and minting stAPT tokens, as well as unstaking and burning `stAPT tokens`.

2.1.8. Treasury

The Treasury module enables the project party to adjust the reward rate and manage the collection and withdrawal of reward fees. It maintains control over the `treasury address`, where all deposited assets are stored.

2.1.9. Withdrawal

The Withdrawal module serves as a support module, facilitating the conversion of `amAPT tokens` back into native tokens when users choose to withdraw from the project in router module.

2.2. Findings

During the audit process, the audit team found some information issues in the given version of Amnis Smart Contracts.

2.2.1. Treasury.move - Stuck stAPT token in Treasury address INFORMATIVE

By the default, when `deposit_amapt` to treasury, the `Coin<AmnisApt>` is converted to `Coin<StakedApt>` and stake it at the `treasury_address`. However, it's important to note that there is currently no mechanism in place for withdrawing `Coin<StakedApt>` from the `treasury_address`, which means that any `Coin<StakedApt>` deposited will remain locked within the `treasury_address` indefinitely.

```
//treasury.move
public(friend) fun deposit_amapt(amapt: Coin<AmnisApt>) {
    aptos_account::deposit_coins(treasury_address(), stapt_token::deposit(amapt)); //<-- stake Coin<StakedApt> to treasury_address
}

//stap_token.move
public(friend) fun deposit(amapt: Coin<AmnisApt>): Coin<StakedApt> acquires StakedAptManagement {
    let stapt_price = stapt_price();
    let staked_apt_management = borrow_global_mut<StakedAptManagement>(@amnis);
    let amapt_deposited = coin::value(&amapt);
    let equivalent_stapt = math64::mul_div(amapt_deposited, precision_u64(), stapt_price);
    coin::merge(&mut staked_apt_management.amapt, amapt);
    event::emit_event(&mut staked_apt_management.mint_event, MintEvent { amount: equivalent_stapt });
    coin::mint(equivalent_stapt, &staked_apt_management.mint_cap) //<-- return Coin<StakedApt>
}
```

RECOMMENDATION

The contract should implement to interact with `Coin<StakedApt>` in `treasury_address`.

UPDATES

- *Oct 23,2023*: This issue has been acknowledged by the Amnis team and the logical interaction will be integrated during the upcoming development phase.

2.2.2. stapt_token.move - Conflict between comment and implementation in `add` function

INFORMATIVE

In the description for `add` function, it declares that this function can be invoked by either the `delegation_manager` or `router`, but with public permission, anyone can call this one. Currently, it does not affect the security of contract, but it may raise issues in the future development.

```
/// Called by delegation_manager to deposit amAPT minted to represent staking rewards
generated.
    /// Or by the router to add withdrawal fees.
    public fun add(amapt: Coin<AmnisApt>) acquires StakedAptManagement {
        coin::merge(&mut borrow_global_mut<StakedAptManagement>(@amnis).amapt, amapt);
    }
```

RECOMMENDATION

Using `public(friend)` modifier to ensure that only friend modules can use it.

UPDATES

- *Nov 07, 2023*: This issue has been acknowledged and fixed by the Amnis team in commit `09f241418e8d7db14d44713c05306bb8b1fed804`.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Oct 28, 2023*	Public Report	Verichains Lab
1.1	*Nov 07, 2023*	Public Report	Verichains Lab

Table 3. Report versions history