BLOCKAPEX

SMART CONTRACT SECURITY

V 1.0

DATE: 21 DEC 2023

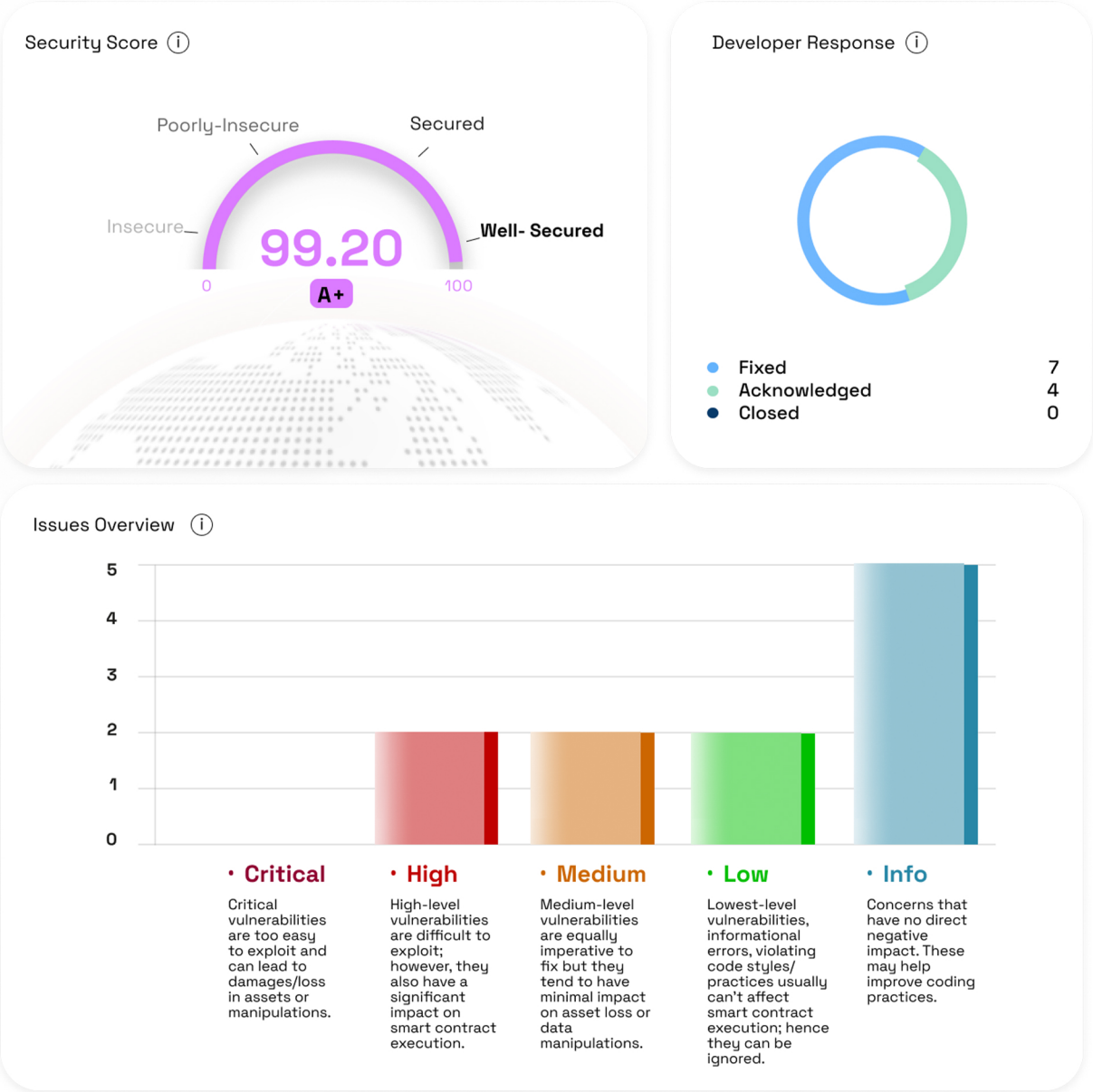
PREPARED FOR: ECLIPSEFI

Contents

1 Executive Summary

Our team performed a technique called Filtered Audit, where three individuals separately audited the EclipseFi cosmwasm vesting Contract. A thorough and rigorous manual testing process involving line by line code review for bugs was carried out. All the raised flags were manually reviewed and re-tested to identify any false positives.

1.1 Summary of Findings Identified



1.2 Scope

1.2.1 In Scope

Vesting Contract - **Contract Overview:**

The vesting contract is designed to manage the distribution of tokens over a set period. Key features include the ability to set a vesting schedule, update recipient allocations, and allow recipients to withdraw vested tokens. The contract's functionality is controlled by designated roles, namely the Owner and Worker.

Folder in scope:

1. `contracts/vesting`

Audit Commit Hash: 0c8bcb06badef9584314a343f425d333d77f7b06

Fixed Commit Hash: f8eb04fbbba82598e66dd7963917e7d6ee7e92b7

1.2.2 Out of Scope

Any features or functionalities not explicitly mentioned in the "In Scope" section are also considered outside the scope of this audit.

1.3 Methodology

The codebase was audited using a filtered audit technique. A band of three (3) auditors scanned the codebase in an iterative process for a time spanning 2 Weeks. Starting with the recon phase, a basic understanding was developed, and the auditors worked on developing presumptions for the developed codebase and the relevant documentation/whitepaper. Furthermore, the audit moved on with the manual code reviews to find logical flaws in the codebase complemented with code optimizations,software, and security design patterns, code styles and best practices.

Status Description

Acknowledged: The issue has been recognized and is under review. It indicates that the relevant team is aware of the problem and is actively considering the next steps or solutions.

Fixed: The issue has been addressed and resolved. Necessary actions or corrections have been implemented to eliminate the vulnerability or problem.

Closed: This status signifies that the issue has been thoroughly evaluated and acknowledged by the development team. While no immediate action is being taken.

S.No	Severity	Findings	Status
#1	HIGH	Potential Risk of Unintended Vesting Schedule Modification Impacting User Entitle-ments	FIXED
#2	HIGH	Improper Allowance Handling on Worker Role Update	FIXED
#3	MEDIUM	Absence of Parameter Validations in Vesting Schedule Configuration	FIXED
#4	MEDIUM	Potential Temporary Denial of Service (DoS) Due to Insufficient Funds in VestingContract	ACKNOWLEDGED
#5	LOW	Absence of Two-Step Ownership Transfer	ACKNOWLEDGED
#6	LOW	Irreversible Vesting Commencement Blocks Recipient Addition	ACKNOWLEDGED
#7	INFO	Missing Validation for Zero Amount Withdrawal	FIXED
#8	INFO	Absence of Event Emission on Contract Instantiation	FIXED
#9	INFO	Absence of Overflow Checks in cargo.toml	FIXED
#10	INFO	Indefinite CW20 Token Allowance Duration for Worker	FIXED
#11	INFO	Lack of NatSpec Documentation	ACKNOWLEDGED

2 Findings and Risk Analysis

2.1 Potential Risk of Unintended Vesting Schedule Modification Impacting User Entitlements

Severity: High

Status: Fixed

Location The vulnerability is located in the `execute_set_vesting_schedule` function within the contract. This function updates the vesting schedule parameters in the contract's state.

Description The contract allows for updating the vesting schedule parameters (`release_interval`, `release_rate`, `initial_unlock`, `lock_period`, `vesting_period`) at any time by the owner. This flexibility poses a significant risk: changes to the vesting schedule can adversely affect the withdrawable amounts of users who have already vested tokens under the original schedule. The dynamic calculation of vested amounts based on the current schedule means that any changes to the schedule parameters can reduce the previously vested amounts that users were eligible to withdraw.

Impact The impact of this vulnerability is high as it directly affects the financial interests of users participating in the vesting contract. Users who have vested tokens under one schedule may find their withdrawable amounts reduced due to a subsequent schedule change. This issue undermines the trust and reliability of the vesting process and can lead to financial losses for users.

Proof of Concept Consider the following example:

Original Schedule: `release_rate` = 10%, `vesting_period` = 100 days.

User A's Vesting: User A vests 1000 tokens. After 50 days, under the original schedule, User A is eligible to withdraw 500 tokens.

Schedule Update: The owner changes the `release_rate` to 5% and extends the `vesting_period` to 200 days.

Impact on User A: When User A checks their withdrawable amount post-update, it's recalculated based on the new parameters. Instead of 500 tokens, User A might now only be eligible to withdraw 250 tokens after 50 days.

Following are the Impacts that can be created if any of these vesting schedule paramaters are modified by the owner accidentally:

Release Interval: This parameter determines the frequency at which vested tokens are released. An accidental decrease in the release interval could lead to users receiving their vested tokens faster than intended, potentially draining the contract's token reserves prematurely. Conversely, an increase could delay access to vested funds, impacting users relying on predictable release schedules.

Release Rate: This is the percentage of total vested tokens released per interval. Increasing this rate could result in a quicker depletion of the token pool, while decreasing it could significantly slow down the vesting process, delaying users' access to their funds.

Initial Unlock: This parameter usually specifies an upfront percentage of tokens that are immediately available to users. Altering this percentage could either unexpectedly lock or release a significant amount of tokens, impacting the liquidity and trust in the vesting process.

Lock Period: The duration for which tokens are completely locked and cannot be withdrawn. Extending this period could trap user funds for longer than expected, while reducing it might lead to a sudden release of tokens, potentially impacting the token's market stability.

Vesting Period: This is the total duration over which the tokens are vested. Shortening this period could lead to a faster than expected release of all vested tokens, impacting the long-term value and utility of the tokens. Extending it could cause user dissatisfaction due to prolonged inaccessibility of their funds

```
    pub fn execute_set_vesting_schedule(
        deps: DepsMut,
        info: MessageInfo,
        release_interval: u64,
        release_rate: u64,
        initial_unlock: u64,
        lock_period: u64,
        vesting_period: u64,
    ) -&gt; Result&lt;Response, ContractError&gt; {
        OWNER.assert_admin(deps.as_ref(), &amp;info.sender)?;

        let mut state: State = STATE.load(deps.storage)?;
        state.release_interval = release_interval;
        state.release_rate = release_rate;
        state.initial_unlock = initial_unlock;
        state.lock_period = lock_period;
        state.vesting_period = vesting_period;
        STATE.save(deps.storage, &amp;state)?;

        Ok(Response::new().add_attribute(&quot;method&quot;, &quot;set_vesting_schedule&quot;)
            )
    }
```

Recommendation Implement a mechanism that preserves the vested amounts calculated under the old schedule, ensuring users can withdraw the amount they vested before any schedule change. or you can restrict owner not modify the vesting schedule during the vesting period.

2.2 Improper Allowance Handling on Worker Role Update

Severity: High

Status: Fixed

Location Function `execute_set_worker`.

Description The `execute_set_worker` function in the contract is designed to assign a new worker with maximum allowance to the reward token. However, the contract does not revoke or decrease the allowance of the previous worker. This oversight could lead to security risks if the access of the previous worker is not properly revoked.

Impact If a previous worker becomes malicious or compromised, they retain access to the maximum allowance of the reward token, posing a significant security threat. This could lead to unauthorized access or misuse of funds.

Proof of Concept :

1. A worker is set using `execute_set_worker`, granting them maximum allowance to the reward token.
2. The contract owner updates the worker to a new address using the same function.
3. The previous worker retains their maximum allowance with expiry to none, as the contract does not revoke it.

```
    pub fn execute_set_worker(
        mut deps: DepsMut,
        info: MessageInfo,
        worker: String,
    ) -&gt; Result&lt;Response, ContractError&gt; {
        OWNER.assert_admin(deps.as_ref(), &amp;info.sender)?;
        let worker = deps.api.addr_validate(&amp;worker)?;
        WORKER.set(deps.branch(), Some(worker.clone()))?;

        let state: State = STATE.load(deps.storage)?;
        let messages: Vec&lt;CosmosMsg&gt; = vec![CosmosMsg::Wasm(WasmMsg::Execute {
            contract_addr: state.reward_token.to_string(),
            msg: to_binary(&amp;Cw20ExecuteMsg::IncreaseAllowance {
                spender: worker.to_string(),
                amount: Uint128::MAX,
                expires: None,
            })?,
            funds: vec![],
        })];
        Ok(Response::new()
            .add_messages(messages)
            .add_attribute(&quot;method&quot;, &quot;set_worker&quot;))
    }
```

Recommendation Implement a mechanism to revoke or reset the allowance of the previous worker when a new worker is assigned. Setting the allowance of the previous worker to zero before assigning

the new worker can mitigate this risk. Additionally, consider setting expiration dates for allowances to enhance security.

2.3 Absence of Parameter Validations in Vesting Schedule Configuration

Severity: Medium

Status: Fixed

Location :

`InstantiateMsg` in the instantiate function for initializing vesting schedule parameters.

`execute_set_vesting_schedul`e function for updating vesting schedule parameters.

Description The vesting contract in question lacks essential validation checks for parameters set during both initialization and updates. Specifically, there are no constraints to prevent the setting of parameters like release intervals, release rates, and vesting periods to illogical or extreme values. This oversight may lead to unintended vesting schedules, potentially locking tokens for unreasonable durations or releasing them too rapidly.

Impact :

Unpredictable Vesting Behavior: Without parameter validation, the vesting schedule can be set in a way that either locks up tokens indefinitely or releases them too quickly, undermining the purpose of vesting.

Proof of Concept :

Initialization without Checks:The parameters (release interval, rate, etc.) can be set to any value, including zero or extremely high numbers, without validation.

Recommendation :

Implement Basic Validation: Enforce checks to ensure that parameters like the release interval, rate, and vesting period are within logical and practical limits. For instance, setting a minimum and maximum cap where applicable.

Non-zero Initialization Enforcement: Ensure that critical parameters cannot be initialized to zero where it's logically inappropriate.

2.4 Potential Temporary Denial of Service (DoS) Due to Insufficient Funds in Vesting Contract

Severity: Medium

Status: Acknowledged

Location :

Function: `execute_withdraw`

Description The vesting contract, as currently implemented, enables recipients to withdraw vested tokens according to a predefined schedule. However, there's a critical oversight in the contract's functionality related to the funding mechanism. The contract lacks a method to ensure sufficient funds are available for withdrawals. This gap could lead to two potential and significant risks, resulting in a temporary DoS (Denial of Service) condition:

Owner's Failure to Deposit Funds: If the contract owner neglects or forgets to deposit the necessary funds into the vesting contract, recipients, despite being eligible, may find themselves unable to withdraw their vested tokens due to the unavailability of funds.

Insufficient Funds Deposit: In cases where the owner deposits funds, but the amount is insufficient to cover all vested amounts, some recipients might successfully withdraw their share, while others may not. This situation could lead to an inequitable and partial distribution of funds, causing a temporary DoS for certain recipients.

Network Clogging: The absence of a validation mechanism for fund sufficiency can lead to a surge in failed withdrawal transactions. Recipients, seeing that they have withdrawable amounts, may repeatedly attempt to withdraw their tokens, unaware that the contract lacks sufficient funds. These repeated failed attempts could lead to network clogging

Proof of Concept

```
    pub fn execute_withdraw(
        deps: DepsMut,
        env: Env,
        info: MessageInfo,
    ) -&gt; Result&lt;Response, ContractError&gt; {
        let state: State = STATE.load(deps.storage)?;
        let mut recpinfo = RECIPIENTS.load(deps.storage, &amp;info.sender)?;
        if recpinfo.total_amount.is_zero() {
            return Ok(Response::default());
        }

        let vested = query_vested(deps.as_ref(), env.clone(), info.sender.as_ref())?;
        let withdrawable = query_withdrawable(deps.as_ref(), env, info.sender.as_ref())?;
        recpinfo.withrawn_amount = vested.amount;
        RECIPIENTS.save(deps.storage, &amp;info.sender, &amp;recpinfo)?;

        let messages: Vec&lt;CosmosMsg&gt; = vec![CosmosMsg::Wasm(WasmMsg::Execute {
            contract_addr: state.reward_token.to_string(),
```

```
            msg: to_binary(&amp;Cw20ExecuteMsg::Transfer {
                recipient: info.sender.to_string(),
                amount: withdrawable.amount,
            })?,
            funds: vec![],
        })];
        Ok(Response::new()
            .add_messages(messages)
            .add_attribute(&quot;method&quot;, &quot;withdraw&quot;))
    }
```

Recommendation :

Modify the contract to incorporate a validation mechanism ensuring sufficient funds are available in the vesting contract before the vesting process starts. This could be implemented by:

Adding a Cw20 Revive Hook: by implementing a cw20 recive whenever this function is called the funds are transfered to the vesting contract and it then execute the set timer to start the vesting which makes sure that there are funds in the contract before commenceing the vesting schedule.

2.5 Absence of Two-Step Ownership Transfer

Severity: Low

Status: Acknowledged

Location :

Function: `execute_transfer_ownership` in contract.rs

Function: `execute_set_worker in` contract.rs

Description The vesting contract employs a single-step process for transferring ownership and setting a worker. This approach, while straightforward, lacks the security of a two-step verification system like cw-ownable . The current implementation allows immediate changes to the contract's ownership and worker roles without requiring the proposed new owner or worker to accept the role.

Impact :

Ownership Transfer:

- Potential for accidental or unauthorized changes in ownership.
- No opportunity for a proposed owner to decline the role, leading to management issues or contract abandonment.

Worker Role Assignment:

- Immediate assignment of worker roles without a validation step can lead to unauthorized or unintended access.

Proof of Concept :

Ownership Transfer: The `execute_transfer_ownership` function directly sets a new owner upon execution by the current owner. There is no intermediate step for the proposed new owner to accept or reject this new role.

```
    OWNER.set(deps.branch(), Some(new_owner))?;
```

Worker Role Assignment: Similarly, the `execute_set_worker` function assigns a new worker directly upon execution by the owner.

```
    WORKER.set(deps.branch(), Some(worker.clone()))?;
```

Recommendation Implement a two-step verification process for both ownership and worker role assignments, similar to the [cw-ownable] of cw-storage plus (https://https://docs.rs/cw-ownable/latest/cw_ownable/) approach.

2.6 Irreversible Vesting Commencement Blocks Recipient Addition

Severity: Low

Status: Acknowledged

Location Function `execute_set_start_time` in the vesting contract

Description The `execute_set_start_time` function in the provided vesting contract poses a operational risk. Once the vesting start time is set and surpassed, the contract logic prohibits adding new recipients due to the check `VestingAlreadyStarted`. If the owner unintentionally triggers the start time without updating any recipients, or if a miner maliciously/unintentially prioritizes such a transaction, the contract becomes unusable for adding new vesting recipients.

Impact :

Operational Risk: The contract is rendered non-functional for adding new recipients, leading to an operational halt.It leads to re-deploying the contract which have cost.

Proof of Concept :

- The vesting start time is initially set to 0 (inactive).
- `execute_set_start_time` function can be called by the owner to set the start time.
- Once set and the current time surpasses this start time, the `VestingAlreadyStarted` error prevents adding new recipients via `execute_update_recipient`.
- If the owner mistakenly starts the vesting schedule (or a miner arranges transactions to trigger the start time before recipient updates), no new recipients can be added, rendering the contract ineffective for its intended purpose.

```
    pub fn execute_set_start_time(
        deps: DepsMut,
        env: Env,
        info: MessageInfo,
        new_start_time: u64,
    ) -&gt; Result&lt;Response, ContractError&gt; {
        OWNER.assert_admin(deps.as_ref(), &amp;info.sender)?;

        let mut state: State = STATE.load(deps.storage)?;

        if state.start_time != 0 &amp;&amp; state.start_time &lt; env.block.time.seconds() {
            return Err(ContractError::VestingAlreadyStarted {});
        }
        if new_start_time &lt; env.block.time.seconds() {
            return Err(ContractError::InvalidStartTime {});
        }

        state.start_time = new_start_time;
        STATE.save(deps.storage, &amp;state)?;

        Ok(Response::new().add_attribute(&quot;method&quot;, &quot;set_start_time&quot;))
    }
```

Recommendation :

Pre-Start Check: Implement a check in `execute_set_start_time` to ensure there are existing recipients (usercount > 0) before allowing the start time to be set.

Emergency Reset Function: Provide a secure way for the owner to reset the start time if no recipients are present.

2.7 Missing Validation for Zero Amount Withdrawal

Severity: Info

Status: Fixed

Location `execute_withdraw` function.

Description The contract lacks a validation check for zero-amount withdrawals in the `execute_withdraw` function. Without this validation, users can initiate a withdrawal transaction even when the amount is zero. Since Cosmos SDK prevents zero-amount native token transfers, such attempts will invariably fail, leading to unnecessary transaction rejections and potential user confusion.

Recommendation Introduce a check at the start of the `execute_withdraw` function to ensure withdrawable.amount exceeds zero. If the amount is zero, the function should return a custom error message.

2.8 Absence of Event Emission on Contract Instantiation

Severity: Info

Status: Fixed

Location Function: `instantiate`

Description The vesting contract, upon successful instantiation, does not emit any custom events or attributes. This omission limits the ability of off-chain listeners and indexers to track and record the initial configuration parameters of the contract.

Recommendation Modify the instantiate function to include the emission of a custom event or attribute that encapsulates relevant details about the contract's initialization. This could include the initial configuration parameters set during instantiation.

2.9 Absence of Overflow Checks in cargo.toml

Severity: Info

Status: Fixed

Location Configuration File: `Cargo.toml`

Description The vesting contract does not explicitly enable overflow checks by setting overflow-checks = true in its configuration. While Rust provides safety against integer overflow in debug mode by default, in release builds, these checks are omitted unless explicitly enabled. This can potentially lead to unchecked overflows in arithmetic operations, affecting the contract's integrity and security.

Recommendation To mitigate this issue and align with best practices, it is recommended to enable overflow checks in the contract's Cargo.toml file. This can be achieved by adding the following line under the [profile.release] section:

```
    [profile.release]
    overflow-checks = true
```

2.10 Indefinite CW20 Token Allowance Duration for Worker

Severity: Info

Status: Fixed

Location Found in the `execute_set_worker` function, specifically where the allowance is set:

Description The vesting contract assigns an indefinite CW20 token allowance to the worker role in the `execute_set_worker` function. This configuration, utilizing `Cw20ExecuteMsg::IncreaseAllowance` with the expires field set to None, grants the worker unrestricted access to the maximum token amount for an unlimited duration.

Recommendation Implement an expiration for the token allowance to enhance security. The expires field should be set to a reasonable duration, consistent with the worker's operational period or a predefined timeframe, after which the allowance would need renewal or reassessment.

2.11 Lack of NatSpec Documentation

Severity: Info

Status: Acknowledged

Location Examples include, but are not limited to, functions `execute_withdraw` and `execute_update_recipient` in vesting contract

Description The contract code is missing NatSpec comments, particularly in key functions like execute_withdraw and execute_update_recipient. NatSpec (Natural Language Specification) provides essential documentation directly in the source code, aiding in understanding the function's purpose, input, output, and behavior.

Recommendation Incorporate NatSpec comments throughout the contract, to clearly describe their functionalities. This practice will aid developers, auditors, and users in comprehending the intended behavior of the contract.

Disclaimer:

The smart contracts provided by the client with the purpose of security review have been thoroughly analyzed in compliance with the industrial best practices till date w.r.t. Smart Contract Weakness Classification (SWC) and Cybersecurity Vulnerabilities in smart contract code, the details of which are enclosed in this report.

This report is not an endorsement or indictment of the project or team, and they do not in any way guarantee the security of the particular object in context. This report is not considered, and should not be interpreted as an influence, on the potential economics of the token (if any), its sale, or any other aspect of the project that contributes to the protocol's public marketing.

Crypto assets/ tokens are the results of the emerging blockchain technology in the domain of decentralized finance and they carry with them high levels of technical risk and uncertainty. No report provides any warranty or representation to any third-party in any respect, including regarding the bug-free nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the reports in any way, including to make any decisions to buy or sell any token, product, service, or asset. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and is not a guarantee as to the absolute security of the project.

Smart contracts are deployed and executed on a blockchain. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. The scope of our review is limited to a review of the programmable code and only the programmable code, we note, as being within the scope of our review within this report. The smart contract programming language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer or any other areas beyond the programming language's compiler scope that could present security risks.

This security review cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While BlockApex has done their best in conducting the analysis and producing this report, it is important to note that one should not rely on this report only - we recommend proceeding with several independent code security reviews and a public bug bounty program to ensure the security of smart contracts.