xMoney

Audit Report

MOVEBIT

contact@bitslab.xyz

https://twitter.com/movebit_

Tue Sep 30 2025

xMoney Audit Report

1 Executive Summary

1.1 Project Information

Description	The xMoney Protocol is an on-chain platform on Sui that integrates a secure cross-chain bridge with a dynamic staking system. It allows users to seamlessly transfer assets between blockchains, secured by a multi-signature relayer network, while also providing the ability to deposit tokens into distinct pools to earn competitive, APR-based rewards
Type	DeFi
Auditors	Alex,PeiQi,Light
Timeline	Sun Sep 14 2025 - Tue Sep 30 2025
Languages	Move
Platform	Others
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/multiversx/mx-bridge-sc-sui https://github.com/multiversx/mx-staking-xmn-sui
Commits	https://github.com/multiversx/mx-bridge-sc-sui: 820dad045eb892d72f236f8147fdfc77eddd9c03 27897be86c0566fd3321f48aea042786b25932af b9394670fa8674448815792cb3cbff28b45e7ede

	https://github.com/multiversx/mx-staking-xmn-sui: 7d6659135bd1fbe7e720e921824926707ef0519f 3428c7b98f44295a84f32ea85142499f6c13ae7a 5dfaf5d18494d2983f8697137287303dffa2e69e 3a7ac98eb1bbb6cadcbde610bae556e89cb68577

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
MOV	packages/apr_pool/Move.toml	6e658a2d7b022278642158335cb64640fc6a7bdb
MOV1	packages/access_control/Move.toml	31f1d356397959917fc1d1304e2f51a32d953c29
ROL	packages/access_control/sources/roles.move	ef1cabffc0fa3e1798318d4846524e385d700a98
PAU	packages/access_control/sources/pausable.move	f1d4064fbaac469cc77b0da8ccc885ab1149f6bd
PAU	sources/pausable.move	ca45bc6074b1b419734a15a3115abf5ae7d0a840
MOV	Move.toml	acf7f0b8c0abc59f808bd427110946daf0f28834
SAF	sources/safe.move	e9c010dd14ce7f3543fbf7cf4552266e4d834ef2
EVE	sources/events.move	e222f3c8842b112ddde384f0c294d181698eaa54
BMO	sources/bridge_module.move	cc8bb7cfdb5f8c9c9bb6664f32e435996136d3f1
ROL	sources/roles.move	197f8f02e7051ca30d44e52664d1ea8525051080

UTI	sources/utils.move	e63bc892511996480c0d7f571dca97a38d22fb51
POO	packages/apr_pool/sources/pool.move	621285ec04b650107228fd35704b6c1dd9719ef9
ACO	packages/access_control/sources/access_control.move	a7e4f4742b656f23ea367255bd882871a1616f0b
MOV2	packages/staking_factory/Move.toml	ed295275551a3fb9fe61e63f07148b415c482ef5
SFA	packages/staking_factory/sources/staking_factory.move	3b8130c9565666eefc5dce29861749f835d9e14d
SST	sources/shared_structs.move	3904b8d0ccb175806a38b18e604b2939727f6714

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	19	19	0
Informational	2	2	0
Minor	10	10	0
Medium	7	7	0
Major	0	0	0
Critical	0	0	0

1.4 MoveBit Audit Breakdown

MoveBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow by bit operations
- Number of rounding errors
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting
- Unchecked CALL Return Values
- The flow of capability
- Witness Type

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Formal Verification(Optional)

Perform formal verification for key functions with the Move Prover.

(4) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by Beniamin to identify any potential issues and vulnerabilities in the source code of the xMoney smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 19 issues of varying severity, listed below.

ID	Title	Severity	Status
BMO-1	`execute_transfer` Batch Data Accumulation Error	Medium	Fixed
BMO-2	`execute_transfer` Batch Data Is Inaccurate	Medium	Fixed
BMO-3	`set_batch_settle_timeout_ms` Scope Needs To Be Verified	Minor	Fixed
BMO-4	`execute_transfer` Consistency Check For Parameter Length Missing	Minor	Fixed
MOV-1	Incorrect `tier` Handling	Minor	Fixed
PAU-1	Contract Suspension Mechanism Is Invalid	Medium	Fixed
POO-1	Unclaimed Rewards Permanently Frozen in ad Contract	Medium	Fixed

POO-2	Precision Loss in `accrue()` Allows Malicious `sync()` Calls to Block Reward Accumulation	Medium	Fixed
ROL-1	Missing Entry Functions for `update_pauser()` and `update_owner()` in Roles Contract	Medium	Fixed
SAF-1	`whitelist_token` Lack of parameter verification	Minor	Fixed
SAF-2	`create_new_batch_internal` Incorrect Permission Settings	Minor	Fixed
SAF-3	Key Words Do Not Conform To The Latest Language Norms	Informational	Fixed
SFA-1	Incorrect Event Type	Minor	Fixed
SFA-2	Missing Generic Parameters in `fund_and_poke_boosted()` Function Call to `apr_pool::pre_accrue_need()`	Minor	Fixed
SFA-3	Missing Update Methods for `max_eps_normal`, `max_eps_boosted`, `base_apr_bps`, and `boost_delta_bps` in `staking_factory` Contract	Minor	Fixed
SFA-4	`request_unstake_locked` Return Field Error	Minor	Fixed
SST-1	`create_token_config` Parameters Cannot Be Modified	Medium	Fixed

SST-2	`set_token_config_whitelisted` Function Modifier Error	Minor	Fixed
SST-3	`deposit_status_none` Unused State	Informational	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the xMoney Smart Contract
:

Bridge Admin

- `set_quorum` :Sets the minimum number of relayer signatures required to approve a batch of transfers.
- `set_batch_settle_timeout_ms` :Configures the settlement timeout period for a batch of transfers.
- `add_relayer` :Adds a new relayer to the approved set and registers their public key.
- `remove_relayer` :Removes an existing relayer from the approved set.
- `set_admin` :Transfers the admin role to a new address.
- `pause_contract` Pauses core bridge functions, such as deposits and transfers.
- `unpause_contract` :Resumes normal bridge operations after a pause.
- `whitelist_token` :Approves a new token for bridging and sets its deposit limits.
- `remove_token_from_whitelist` :Discontinues support for a previously approved token.
- `set_batch_timeout_ms` :Sets the time limit for new deposits to be added to a pending batch.
- `set_batch_size` :Defines the maximum number of deposits that can be included in a single batch.
- `set_token_min_limit` :Updates the minimum deposit amount for a specific whitelisted token.
- `set_token_max_limit` :Updates the maximum deposit amount for a specific whitelisted token.
- `set_bridge_addr` :Updates the address of the main Bridge contract that is authorized to execute transfers.

- `init_supply` :Provides an initial supply of native tokens to the BridgeSafe contract to facilitate transfers.
- `set_treasury_cap / set_policy_cap` :Configures the TreasuryCap and TokenPolicyCap for mint/burn token mechanics.

Bridge Relayer

- `execute_transfer` :Submits the required signatures to validate and trigger the execution of a batch of transfers, releasing funds to the recipients on the Sui network.

Bridge User

- `deposit` :Deposits tokens into the bridge, specifying the recipient address on the destination chain to initiate a cross-chain transfer.

Staking Admin

- `launch_with_boosted` :Initializes and deploys the entire Staking Factory, including creating the normal and boosted staking pools, and setting the initial APRs, emission caps, and unbonding periods.
- `fund_reserve` :Adds funds (reward tokens) to the factory's main reserve pool.
- `set_unbonding_ts` :Updatesthe unbonding period required for the normal staking pool.

Staking User

- `stake` :Deposits normal tokens into the standard pool to create a new personal staking positionand begin earning rewards.
- `stake_locked` :Deposits locked (boosted) tokens into the boosted pool to create a new position with a higher APR.
- `stake_more` :Adds more normal tokens to an existing staking position to increase the principal amount.
- `stake_more_locked` :Adds more locked tokens to an existing boosted position.
- `request_unstake` :Initiates the unbonding process for a specified amount from a normal position. This starts a fixed cooldown timer and creates an UnbondingTicket.

- `request_unstake_locked` :Requests to unstake from a boosted position. This is only possible after the position's own lockup period has expired, and it creates a redeemable UnbondingTicket.
- `unbond` :Redeems a matured UnbondingTicket to withdraw the original principal (normal tokens) after the cooldown period has ended.
- `unbond_locked` :Redeems a matured UnbondingTicket from a boosted stake.
- `claim` :Claims accumulated rewards from a normal position without affecting the staked principal.
- `claim_locked` :Claims accumulated rewards from a boosted position.
- `destroy_position` :Destroys an empty OpenPosition object to free up on-chain storage after all principal has been withdrawn.

Staking Keeper

- `sync` :Synchronizes the state of both staking pools.
- `poke` : Triggers the reward accrual calculation for a single pool. The sync function calls this for both the normal and boosted pools.

4 Findings

BMO-1 `execute_transfer` Batch Data Accumulation Error

Severity: Medium

Status: Fixed

Code Location:

sources/bridge_module.move#283

Descriptions:

The BatchExecuted event is triggered within the if (is_batch_complete) code block. The purpose of this event is to record the execution summary of the entire batch, including the total number of transactions (transfers_count) and the number of successful transactions (successful_transfers).

However, the execute_transfer function can be called multiple times to handle different transactions in the same batch (when the successor submits the signature in batches). Each time it is called, recipients and amounts contain only the data of the current part of the transaction. total_transfers is calculated using the length of the recipients vector at the last call, rather than the total number of transactions in the entire batch. Similarly, successful_count only reflects the number of successes of the current call.

```
if (is_batch_complete) {
    let cross_status = shared_structs::create_cross_transfer_status(
        bridge.transfer_statuses,
        clock::timestamp_ms(clock),
    );
    table::add(&mut bridge.cross_transfer_statuses, batch_nonce_mvx, cross_status);

    let total_transfers = vector::length(&recipients);
    bridge.transfer_statuses = vector::empty<DepositStatus>();
    event::emit(BatchExecuted {
        batch_nonce_mvx,
        transfers_count: total_transfers,
        successful_transfers: successful_count,
    });
};
```

Suggestion:

The data should be cumulative or calculated by reading the status of all transactions from the persistent storage when the batch is completed, rather than relying solely on the result of the current call

Resolution:

This issue has been fixed. The client has adopted our suggestions.

BMO-2 `execute_transfer` Batch Data Is Inaccurate

Severity: Medium

Status: Fixed

Code Location:

sources/bridge_module.move#217

Descriptions:

In the execute_transfer function, The state variable bridge.transfer_statuses is used to store the execution result of each transfer in the current batch. If a relayer (or malicious attacker) calls this function and sets is_batch_complete = false, the function will handle the transfers normally. And append the result to bridge.transfer_statuses. However, because the if condition is not satisfied, this vector will not be cleared after the function execution ends, resulting in the retention of outdated state data. When the next batch (regardless of whether is_batch_complete is true or false) is processed, The new transfer status will be appended to the outdated data left over from the previous batch. Ultimately, when a batch is marked as completed (is_batch_complete = true), a completely incorrect list that mixes multiple batch statuses will be permanently recorded in the cross_transfer_statuses table. This seriously undermines the data consistency of the core status of the contract and the accuracy of historical records.

Suggestion:

Fix the code to ensure that the transaction corresponding to each executed batch is accurately recorded in the corresponding batch

Resolution:

This issue has been fixed. The client has adopted our suggestions.

BMO-3 set_batch_settle_timeout_ms Scope Needs To Be Verified

Severity: Minor

Status: Fixed

Code Location:

sources/bridge_module.move#128

Descriptions:

The function directly assigns new_timeout_ms to bridge.batch_settle_timeout_ms without any validation of the value of new_timeout_ms if the timeout is set to 0 or a very small value. This may cause the batch processing to time out almost immediately, thereby disrupting the normal operation process of the bridge, and may even be exploited to trigger unexpected settlement logic. So the configuration is to add a minimum value verification for the configuration, And after the configuration is completed, it should issue an event similar to other configuration functions

```
public entry fun set_batch_settle_timeout_ms(
    bridge: &mut Bridge,
    _admin_cap: &AdminCap,
    safe: &BridgeSafe,
    new_timeout_ms: u64,
    clock: &Clock,
    ctx: &mut TxContext,
) {
    let signer = tx_context::sender(ctx);
    assert_admin(bridge, signer);
    pausable::assert_paused(&bridge.pause);
    assert!(!safe::is_any_batch_in_progress(safe, clock), EPendingBatches);

    bridge.batch_settle_timeout_ms = new_timeout_ms;
}
```

Suggestion:

Add parameter range verification and issue the event

Resolution:

This issue has been fixed. The client has adopted our suggestions.

BMO-4 `execute_transfer` Consistency Check For Parameter Length Missing

Severity: Minor

Status: Fixed

Code Location:

sources/bridge_module.move#217

Descriptions:

The function lacks length consistency checks for multiple core vectors (recipients, amounts, tokens, deposit_nonces) of the input. This function takes multiple vectors as parameters, and these vectors are logically "paired". For example, recipients[i] should receive the amount of tokens[i] of amounts amounts[i], and this transaction corresponds to the deposit_nonces[i] of the original deposit. The code will subsequently traverse these vectors to handle each withdrawal. However, before processing, the code does not verify whether the lengths of these vectors are exactly equal.

Suggestion:

Add checks

```
public entry fun execute_transfer<T>(
    // ... parameters
) {
    let len = vector::length(&tokens);
    assert!(vector::length(&recipients) == len, E_INVALID_ARGUMENT);
    assert!(vector::length(&amounts) == len, E_INVALID_ARGUMENT);
    assert!(vector::length(&deposit_nonces) == len, E_INVALID_ARGUMENT);

    // ... rest of the function logic
}
```

Resolution:

This issue has been fixed. The client has adopted our suggestions.

MOV-1 Incorrect `tier` Handling

Severity: Minor

Status: Fixed

Code Location:

packages/staking_factory/Move.toml#424

Descriptions:

The `request_unstake_locked` function is used to starts the unbond period from a bossted pool for a given amount; therefore, `tier` should be set to 1.

```
/// Starts the unbond period from a bossted pool for a given amount
public fun request_unstake_locked<R, Dn, Db>(
    sf: &mut StakingFactory<R, Dn, Db>,
    op: &mut OpenPosition<Db>,
    amount: u64,
    clk: &Clock,
    ctx: &mut TxContext
) {
    let ticket = request_unstake_locked_internal(sf, op, amount, clk, ctx);

    event::emit(EUnstakeRequested<Db> {
        position: object::id(op),
        ticket: object::id(&ticket),
        amount,
        unlock_ts: ticket.unlock_ts,
        tier: 0,
    });

    transfer::transfer(ticket, tx_context::sender(ctx));
}
```

Suggestion:

Change `tire` to 1.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

PAU-1 Contract Suspension Mechanism Is Invalid

Severity: Medium

Status: Fixed

Code Location:

packages/access_control/sources/pausable.move#34

Descriptions:

The pause mechanism in the current contract is a feature that has not been fully implemented. Although the underlying logic (the Pausable structure and the pause/unpause function) already exists, due to the lack of necessary external calling interfaces, the pauser role is rendered ineffective, and the entire protocol actually does not have the ability to suspend operations in emergency situations. This is a rather serious design defect.

```
public(package) fun pause<T>(p: &mut Pausable<T>, ctx: &TxContext) {
    if (!p.paused) {
        p.paused = true;
        event::emit(Paused<T> { by: tx_context::sender(ctx) });
    }
}

public(package) fun unpause<T>(p: &mut Pausable<T>, ctx: &TxContext) {
    if (p.paused) {
        p.paused = false;
        event::emit(Unpaused<T> { by: tx_context::sender(ctx) });
    }
}
```

When the StakingFactory contract is initialized, an AccessControl object containing the 'pausable' module is created, indicating that its design intention is to enable the contract to have a pause function. However, All core public entry functions (such as stake, stake_locked, request_unstake, unbond, claim, etc.) do not call 'pausable::assert_not_paused()' for state checking before the function is executed. This leads to the situation where even if the administrator sets a pause state in the background, it is impossible to prevent users from interacting with the key functions of the contract, making the pause mechanism completely ineffective and unable to prevent the protocol from running in an emergency.

Suggestion:

Correct the incorrect modifier

Resolution:

This issue has been fixed. The client has adopted our suggestions.

POO-1 Unclaimed Rewards Permanently Frozen in ad Contract

Severity: Medium

Status: Fixed

Code Location:

packages/apr_pool/sources/pool.move#196-202

Descriptions:

The operator first calls `fund_reserve()` to deposit funds into `sf.reserve_r` in the factory contract. When updating user rewards, the required funds are gradually transferred to `p.reserve` in the pool contract. Afterwards, through the `accrue()` function, the funds are moved from `p.reserve` to the `ad` contract. Users can then claim their rewards from the `ad` contract via the `claim_all()` function.

```
/// Claim all rewards for T from a position (will pull from AD's balances).
public fun claim_all<R, D>(
    p: &mut Pool<R, D>,
    pos: &mut ad::Position
): balance::Balance<R> {
    // (accrue not strictly needed here if other ops already poke often,
    //  but it's fine to accrue on claim as well via a keeper path.)
    ad::withdraw_all_rewards<R>(&mut p.ad, pos)
}


``

However, if some user accounts are lost and the rewards are never claimed, and since
the protocol does not provide a dedicated method to withdraw unclaimed funds from
the `ad` contract, those funds will remain permanently frozen in the `ad` contract.
```

Suggestion:

It is recommended to introduce a recovery mechanism that allows the protocol to withdraw or recycle unclaimed funds from the ad contract

Resolution:

This issue has been fixed. The client has adopted our suggestions.

POO-2 Precision Loss in `accrue()` Allows Malicious `sync()` Calls to Block Reward Accumulation

Severity: Medium

Status: Fixed

Code Location:

packages/apr_pool/sources/pool.move#117

Descriptions:

In the `accrue()` function, the protocol calls `compute_needed_funds()` to calculate the required funds and then updates `p.last_ts = now_sec`. The `need` amount is calculated as:

```
target = tvl_shares * apr_bps * dt_sec / (YEAR * BPS_DEN)
```

```
public fun compute_needed_funds(
    tvl_shares: u64,
    apr_bps: u64,
    max_emission_per_sec: u64,
    dt_sec: u64
): u64 {
    if (dt_sec == 0 || tvl_shares == 0 || apr_bps == 0) return 0;

    let target =
        (tvl_shares as u128) * (apr_bps as u128) * (dt_sec as u128)
            / ((YEAR as u128) * (BPS_DEN as u128));
    let cap = (max_emission_per_sec as u128) * (dt_sec as u128);
    (if (target < cap) { target } else { cap }) as u64
}
```

According to the configuration and test cases:

- `YEAR = 31,536,000`
- `BPS_DEN = 10,000`

- `apr_bps = 1000`

To ensure `target` is non-zero, when `dt_sec = 1s`, the minimum `tvl_shares` required is:

```
tvl_shares = 31,536,000 * 10,000 / 1000 = 315,360,000
```

This means if the first user’s stake is less than `315,360,000`, a malicious attacker can repeatedly call the `sync()` function every 1 second. Due to precision loss, the calculated `need` will always be `0`, but the timestamp (`p.last_ts`) will still be updated. As a result, normal users’ rewards cannot properly accumulate.

Suggestion:

It is recommended to introduce a minimum `need` threshold or use higher-precision arithmetic in `compute_needed_funds()` to ensure that small stakes still produce non-zero rewards.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

ROL-1 Missing Entry Functions for `update_pauser()` and `update_owner()` in Roles Contract

Severity: Medium

Status: Fixed

Code Location:

packages/access_control/sources/roles.move#75-97

Descriptions:

In the roles contract, there are package functions `update_pauser()` and `update_owner()`, but there are no corresponding entry functions to call them.

```
public(package) fun update_owner<T>(roles: &mut Roles<T>, new_owner: address, ctx:
&TxContext) {
    assert!(roles.is_owner(ctx.sender()), ENotOwner);

    let old_owner = roles.update_address(OwnerKey {}, new_owner);

    event::emit(OwnerChanged<T> {
        old_owner,
        new_owner
    });
}
```

As a result, the protocol cannot update the `pauser` and `owner`.

Suggestion:

It is recommended to add corresponding entry functions that invoke update_pauser() and update_owner() so the protocol can properly update the pauser and owner roles when needed.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

SAF-1 `whitelist_token` Lack of parameter verification

Severity: Minor

Status: Fixed

Code Location:

sources/safe.move#90

Descriptions:

In the `safe::whitelist_token` function, although `min_limit` and `max_limit` are received as parameters, the logical constraint of `min_limit <= max_limit` is not verified. Similarly, there is also a lack of cross-validation of this logic in the independent `set_token_min_limit` and `set_token_max_limit` functions. This allows administrators to set an invalid deposit range (for example, the minimum deposit amount is greater than the maximum deposit amount), thereby causing the deposit function of this token to be de facto denied service. Any user's deposit attempt will fail because both conditions cannot be met simultaneously.

Suggestion:

Verify the parameter range

Resolution:

This issue has been fixed. The client has adopted our suggestions.

SAF-2 `create_new_batch_internal` Incorrect Permission Settings

Severity: Minor

Status: Fixed

Code Location:

sources/safe.move#396

Descriptions:

The create_new_batch_internal function, based on its _internal naming suffix and the overall code calling logic, is clearly designed as an auxiliary function for use within a module. However, it was wrongly declared as public fun

```
public fun create_new_batch_internal(safe: &mut BridgeSafe, clock: &Clock, _ctx: &mut
TxContext) {
    assert!(safe.batches_count < MAX_U64, EOverflow);
    let nonce = safe.batches_count + 1;
    let batch = shared_structs::create_batch(nonce, clock::timestamp_ms(clock));
    table::add(&mut safe.batches, safe.batches_count, batch);
    safe.batches_count = nonce;
}
```

Suggestion:

Modify it to an internal function

Resolution:

This issue has been fixed. The client has adopted our suggestions.

SAF-3 Key Words Do Not Conform To The Latest Language Norms

Severity: Informational

Status: Fixed

Code Location:

sources/safe.move#1

Descriptions:

The `public(friend)` visibility modifier, which has been deprecated by the Sui Move language specification, is used in many places in the code. This modifier has been completely replaced by `public(package)` to restrict functions to be called only by modules within the same package (package). Continuing to use the old keywords will make the code not conform to the latest Sui Move language standard.

Suggestion:

Modify the corresponding keywords to conform to the latest language norms

Resolution:

This issue has been fixed. The client has adopted our suggestions.

SFA-1 Incorrect Event Type

Severity: Minor

Status: Fixed

Code Location:

packages/staking_factory/sources/staking_factory.move#794

Descriptions:

In the `set_boost_delta` function, updating `boost_delta_bps` emits the wrong event `ELockedPeriodChanged`. This event is intended for locked period changes, not boost delta modifications.

Suggestion:

Change the event to `EBoostDeltaChanged`

Resolution:

This issue has been fixed. The client has adopted our suggestions.

SFA-2 Missing Generic Parameters in `fund_and_poke_boosted()` Function Call to `apr_pool::pre_accrue_need()`

Severity: Minor

Status: Fixed

Code Location:

packages/staking_factory/sources/staking_factory.move#194

Descriptions:

In the `fund_and_poke_normal()` function, the protocol calls `apr_pool::pre_accrue_need()` and passes the generic `<R, Dn>`.

```
fun fund_and_poke_normal<R, Dn, Db>(sf: &mut StakingFactory<R, Dn, Db>, clk: &Clock)
{
    let now = clock::timestamp_ms(clk) / 1000;
    let need = apr_pool::pre_accrue_need<R, Dn>(&sf.normal_pool, now);
```

However, in the `fund_and_poke_boosted()` function, the protocol calls `apr_pool::pre_accrue_need()` without passing the generic `<R, Db>`.

```
/// Pre-fund and poke the **boosted** pool.
fun fund_and_poke_boosted<R, Dn, Db>(sf: &mut StakingFactory<R, Dn, Db>, clk: &Clock)
{
    let now = clock::timestamp_ms(clk) / 1000;
    let need = apr_pool::pre_accrue_need(&sf.boosted_pool, now);
```

Suggestion:

It is recommended to update the `fund_and_poke_boosted()` function to include the generic parameters `<R, Db>` when calling `apr_pool::pre_accrue_need()`.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

SFA-3 Missing Update Methods for `max_eps_normal`, `max_eps_boosted`, `base_apr_bps`, and `boost_delta_bps` in `staking_factory` Contract

Severity: Minor

Status: Fixed

Code Location:

packages/staking_factory/sources/staking_factory.move#93

Descriptions:

In the `staking_factory` contract, there are corresponding methods to modify `unbound_sec`, but there are no methods to update `max_eps_normal`, `max_eps_boosted`, `base_apr_bps`, `locked_period_sec`, and `boost_delta_bps`.

```
public fun set_unbonding_ts<R, Dn, Db>(
    sf: &mut StakingFactory<R, Dn, Db>,
    unbound_sec: u64,
    ctx: &mut TxContext
) {
    assert_owner(sf, ctx);

    let old_ts = sf.unbound_sec;
    sf.unbound_sec = unbound_sec;

    event::emit(EUnbondPeriodChanged {
        admin: ctx.sender(),
        old: old_ts,
        new: unbound_sec,
    });
}
```

Suggestion:

It is recommended to add dedicated update functions in the `staking_factory` contract to allow modification of `max_eps_normal`, `max_eps_boosted`, `base_apr_bps`, `locked_period_sec`, and `boost_delta_bps`.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

SFA-4 `request_unstake_locked` Return Field Error

Severity: Minor

Status: Fixed

Code Location:

packages/staking_factory/sources/staking_factory.move#406

Descriptions:

When understanding staking requests for enhancement pools, request_unstake_locked. In the EUnstakeRequested event triggered by the function, the request_unstake_locked_internal function called within the function asserts from the very beginning that the tier of the position must be 1, but the tier field is wrongly hard-coded as 0. According to the contract design, the position tier of the enhancement pool should be 1. This inconsistency will cause the on-chain event log to not match the actual state of the contract.

```
event::emit(EUnstakeRequested<Db> {
    position: object::id(op),
    ticket: object::id(&ticket),
    amount,
    unlock_ts: ticket.unlock_ts,
    tier: 0,
});
```

Suggestion:

Modify the field value

Resolution:

This issue has been fixed. The client has adopted our suggestions.

SST-1 `create_token_config` Parameters Cannot Be Modified

Severity: Medium

Status: Fixed

Code Location:

sources/shared_structs.move#195

Descriptions:

In actual operation, the project party may need to adjust the attributes of the token in accordance with market changes, security incidents or the strategies of partners. Under the current design, the fields of is_native and is_locked, as well as is_mint_burn, cannot be modified after being created in TokenConfig, and the is_mint_burn field is not found to be used anywhere in the code. Administrators are unable to perform the necessary operations for these modifications. Service interruption may occur and may lead to inconsistent status.

```
public fun create_token_config(
    whitelisted: bool,
    is_native: bool,
    min_limit: u64,
    max_limit: u64,
    is_locked: bool,
): TokenConfig {
    TokenConfig {
        whitelisted,
        is_native,
        is_mint_burn: false,
        min_limit,
        max_limit,
        total_balance: 0,
        is_locked,
    }
}
```

Suggestion:

Add the corresponding modification function code and delete unnecessary parameters

Resolution:

This issue has been fixed. The client has adopted our suggestions.

SST-2 set_token_config_whitelisted Function Modifier Error

Severity: Minor

Status: Fixed

Code Location:

sources/shared_structs.move#120-142

Descriptions:

These functions only involve internal calls and should use public(package) instead of public fun

```
public fun set_token_config_whitelisted(config: &mut TokenConfig, whitelisted: bool) {
    config.whitelisted = whitelisted;
}

public fun set_token_config_min_limit(config: &mut TokenConfig, min_limit: u64) {
    config.min_limit = min_limit;
}

public fun set_token_config_max_limit(config: &mut TokenConfig, max_limit: u64) {
    config.max_limit = max_limit;
}

public fun set_token_config_is_native(config: &mut TokenConfig, is_native: bool) {
    config.is_native = is_native;
}

public fun set_token_config_is_locked(config: &mut TokenConfig, is_locked: bool) {
    config.is_locked = is_locked;
}

public fun set_token_config_is_mint_burn(config: &mut TokenConfig, is_mint_burn: bool)
{
    config.is_mint_burn = is_mint_burn;
}
```

Suggestion:

Modify the function to an internal function

Resolution:

This issue has been fixed. The client has adopted our suggestions.

SST-3 `deposit_status_none` Unused State

Severity: Informational

Status: Fixed

Code Location:

sources/shared_structs.move#88

Descriptions:

These functions have never been called by any other logic throughout the entire contract.

Therefore, the code inside it will never be executed.

```
public fun deposit_status_none(): DepositStatus {
    DepositStatus::None
}

public fun deposit_status_pending(): DepositStatus {
    DepositStatus::Pending
}

public fun deposit_status_in_progress(): DepositStatus {
    DepositStatus::InProgress
}
```

Suggestion:

Delete the useless code

Resolution:

This issue has been fixed. The client has adopted our suggestions.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.