Blockchain Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.04.24, the SlowMist security team received the benfenorg team's security audit application for bfc node (Rust), developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Suggestion	There are better practices for coding or architecture.

In black box testing and gray box testing, we use methods such as fuzz testing and script testing to test the robustness of the interface or the stability of the components by feeding random data or constructing data with a specific structure, and to mine some boundaries Abnormal performance of the system under conditions such as bugs or abnormal performance. In white box testing, we use methods such as code review, combined with the relevant experience accumulated by the security team on known blockchain security vulnerabilities, to analyze the object definition and logic implementation of the code to ensure that the code has the key components of the key logic. Realize no known vulnerabilities; at the same time, enter the vulnerability mining mode for new scenarios and new technologies, and find possible 0day errors.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.

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 was considered during the audit of the smart contract:

NO.	Audit Items	Result
1	Design Logic Audit	Some Risks
2	Others	Some Risks
3	State Consistency Audit	Some Risks
4	Failure Rollback Audit	Some Risks
5	Unit Test Audit	Passed
6	Integer Overflow Audit	Some Risks
7	Parameter Verification Audit	Passed
8	Error Unhandle Audit	Some Risks

NO.	Audit Items	Result
9	Boundary Check Audit	Passed
10	SAST	Some Risks

3 Project Overview

3.1 Project Introduction

Blockchain based on Sui development.

3.2 Coverage

Target Code and Revision:

https://github.com/benfenorg/bfc

Phase 1 audit scope:

```
sui-execution/latest/sui-adapter/src/execution_engine.rs
crates/sui-core/src/authority.rs
crates/sui-types/src/governance.rs
crates/sui-config/src/genesis.rs
crates/sui-types/src/bfc_system_state/mod.rs
sui-execution/latest/sui-adapter/src/gas_charger.rs
sui-execution/latest/sui-adapter/src/execution_engine.rs
crates/sui-core/src/checkpoints/mod.rs
crates/sui-types/src/messages_checkpoint.rs
crates/sui-types/src/stable_coin/stable.rs
crates/sui-types/src/stable_coin.rs
crates/sui-core/src/authority/authority_store_types.rs
crates/sui-json-rpc-types/src/sui_object.rs
crates/sui-types/src/base_types.rs
crates/sui-types/src/gas.rs
crates/sui-types/src/dao.rs
crates/sui-types/src/proposal.rs
crates/sui-types/src/temporary_store.rs
crates/sui-types/src/lib.rs
crates/sui-indexer/src/apis/governance_api.rs
crates/sui-indexer/src/apis/read_api.rs
```

```
crates/sui-json-rpc/src/api/governance.rs
crates/sui-json-rpc/src/api/read.rs
crates/sui-json-rpc/src/governance_api.rs
crates/sui-json-rpc/src/read_api.rs
(Focus on code changes from version:
https://github.com/MystenLabs/sui/releases/tag/mainnet-v1.9.1 to version:
5e0efd16a7b98acfb8a7fdde64dfc4bfb0003aa5)
```

Phase 2 audit version:

https://github.com/benfenorg/bfc/tree/develop_v.1.1.4

Phase 3 audit version:

https://github.com/benfenorg/bfc/tree/develop_v1.21.0

Phase 4 audit version:

https://github.com/benfenorg/bfc/tree/develop_v1.22.0

Phase 5 audit version:

https://github.com/benfenorg/bfc/tree/develop_v1.23.0

3.3 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Duplicate variable definitions and overrides	Others	Medium	Fixed
N2	Missing transaction rollback	Failure Rollback Audit	High	Fixed
N3	Unsafe error handling methods	Error Unhandle Audit	Information	Acknowledged
N4	Unused variables or code blocks	Others	Low	Fixed
N5	Integer overflow risk	Integer Overflow Audit	High	Fixed
N6	There are a number of outstanding codes	Others	Information	Acknowledged

NO	Title	Category	Level	Status
N7	Unused async markers	Others	Information	Acknowledged
N8	Fixed value with unknown meaning	Others	Information	Fixed
N9	Unwrapped duplicate code	Others	Information	Fixed
N10	Log Printing Optimization	Others	Information	Fixed
N11	Centralized management risk	Design Logic Audit	Low	Ignored
N12	Spelling errors	Others	Information	Fixed
N13	Query records are not paged	Design Logic Audit	Medium	Fixed
N14	Error in defining field access rights	Others	Information	Fixed
N15	Relying on stockpiles in security breaches	SAST	Low	Fixed
N16	Unnecessary clone operations	Others	Information	Ignored
N17	Unnecessary for loops	Others	Information	Ignored
N18	The condition is either constant or not constant	Design Logic Audit	Medium	Fixed
N19	Unused variables or code blocks	Others	Low	Fixed
N20	Integer overflow risk	Integer Overflow Audit	Low	Ignored
N21	Incomplete query records	Design Logic Audit	Low	Fixed
N22	Use of rounding in calculating returns	Integer Overflow Audit	High	Fixed
N23	`get_bfc_zklogin_salt` exception code	Others	Low	Fixed
N24	Improper error handling	Error Unhandle Audit	High	Fixed

NO	Title	Category	Level	Status
N25	In business logic, operating system time is used.	State Consistency Audit	Information	Acknowledged
N26	Used operating system random number	State Consistency Audit	Information	Acknowledged
N27	hard-coded address	Others	Suggestion	Fixed
N28	Unsafe integer arithmetic methods	Integer Overflow Audit	Low	Acknowledged
N29	The key logic is marked as TODO	Others	Information	Acknowledged
N30	The logic of address transfer is not rigorous enough	Design Logic Audit	Low	Fixed
N31	Unnecessary comma	Others	Suggestion	Fixed

4 Findings

4.1 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

Refer to the audit process documentation.

4.2 Vulnerability Summary

[N1] [Medium] Duplicate variable definitions and overrides

Category: Others

Content

`charge_arg` and `system_obj` are reused several times in the code.

- sui-execution/latest/sui-adapter/src/execution_engine.rs

```
    pub fn construct_bfc_round_pt(
        param: ChangeObcRoundParams,
        reward_rate: u64,
        storage_rebate: u64
    ) -> Result<ProgrammableTransaction, ExecutionError> {
        let mut builder = ProgrammableTransactionBuilder::new();


        for (type_tag,gas_cost_summary) in param.stable_gas_summarys {
            // create rewards in stable coin


            let charge_arg = builder
                .input(CallArg::Pure(
                    bcs::to_bytes(&(

calculate_reward_rate(gas_cost_summary.gas_by_stable.computation_cost, reward_rate) +
gas_cost_summary.gas_by_stable.storage_cost)).unwrap(),
                ))
                .unwrap();
            let rewards = builder.programmable_move_call(
                SUI_FRAMEWORK_PACKAGE_ID,
                BALANCE_MODULE_NAME.to_owned(),
                BALANCE_CREATE_REWARDS_FUNCTION_NAME.to_owned(),
                vec![type_tag.clone()],
                vec![charge_arg],
            );


            //exchange stable coin to bfc
            let system_obj = builder.input(CallArg::BFC_SYSTEM_MUT).unwrap();
            let charge_arg = builder
                .input(CallArg::Pure(
                    bcs::to_bytes(&
(calculate_reward_rate(gas_cost_summary.gas_by_bfc.computation_cost, reward_rate) +
gas_cost_summary.gas_by_bfc.storage_cost)).unwrap(),
                ))
                .unwrap();
            let rewards_bfc = builder.programmable_move_call(
                BFC_SYSTEM_PACKAGE_ID,
                BFC_SYSTEM_MODULE_NAME.to_owned(),
                STABLE_COIN_TO_BFC_FUNCTION_NAME.to_owned(),
                vec![type_tag.clone()],
                vec![system_obj,rewards,charge_arg],
            );


            // Destroy the rewards.
```

```
            builder.programmable_move_call(
                SUI_FRAMEWORK_PACKAGE_ID,
                BALANCE_MODULE_NAME.to_owned(),
                BALANCE_DESTROY_REBATES_FUNCTION_NAME.to_owned(),
                vec![GAS::type_tag()],
                vec![rewards_bfc],
            );


        }
        let storage_rebate_arg = builder
            .input(CallArg::Pure(
                bcs::to_bytes(&(storage_rebate+ param.bfc_computation_charge -
  calculate_reward_rate(param.bfc_computation_charge, reward_rate))).unwrap(),
            ))
            .unwrap();
        let storage_rebate = builder.programmable_move_call(
            SUI_FRAMEWORK_PACKAGE_ID,
            BALANCE_MODULE_NAME.to_owned(),
            BALANCE_CREATE_REWARDS_FUNCTION_NAME.to_owned(),
            vec![GAS::type_tag()],
            vec![storage_rebate_arg],
        );


        let system_obj = builder.input(CallArg::BFC_SYSTEM_MUT).unwrap();
        builder.programmable_move_call(
            BFC_SYSTEM_PACKAGE_ID,
            BFC_SYSTEM_MODULE_NAME.to_owned(),
            DEPOSIT_TO_TREASURY_FUNCTION_NAME.to_owned(),
            vec![],
            vec![system_obj,storage_rebate],
        );


        Ok(builder.finish())
    }
```

`let result = programmable_transactions::execution::execute::<execution_mode::System>(...)` is reused and the second let result overwrites the first value under certain conditions. This can lead to potential logic errors because the first execution result is not handled appropriately.

- sui-execution/latest/sui-adapter/src/execution_engine.rs

```
    fn advance_epoch(
            change_epoch: ChangeEpoch,
```

```
        temporary_store: &mut TemporaryStore<'_>,
        tx_ctx: &mut TxContext,
        move_vm: &Arc<MoveVM>,
        gas_charger: &mut GasCharger,
        protocol_config: &ProtocolConfig,
        metrics: Arc<LimitsMetrics>,
    ) -> Result<(), ExecutionError> {
         //...
        let result = programmable_transactions::execution::execute::
<execution_mode::System>(           protocol_config,
            metrics.clone(),
            move_vm,
            temporary_store,
            tx_ctx,
            gas_charger,
            advance_epoch_pt,
        );
        //...
        let result = programmable_transactions::execution::execute::
<execution_mode::System>(
            protocol_config,
            metrics.clone(),
            move_vm,
            temporary_store,
            tx_ctx,
            gas_charger,
            advance_epoch_pt,
        );
         //...
        Ok(())
    }
```

Solution

Re-write the relevant code.

Status

Fixed

[N2] [High] Missing transaction rollback

Category: Failure Rollback Audit

Content

Statuses such as `temporary_store` and `gas_charger` should be rolled back in a timely manner when a run error message is captured.

- sui-execution/latest/sui-adapter/src/execution_engine.rs

```
        let result = programmable_transactions::execution::execute::
<execution_mode::System>(           protocol_config,
            metrics.clone(),
            move_vm,
            temporary_store,
            tx_ctx,
            gas_charger,
            advance_epoch_pt,
        );
        if result.is_err() {
            tracing::error!(
            "Failed to execute change round transaction. Switching to safe mode. Error:
{:?}. Input objects: {:?}. Tx data: {:?}",
            result.as_ref().err(),
            temporary_store.objects(),
            change_epoch,
            );
        }
```

Solution

Add rollback logic

```
temporary_store.drop_writes();
gas_charger.reset_storage_cost_and_rebate();
```

Status

Fixed

[N3] [Information] Unsafe error handling methods

Category: Error Unhandle Audit

Content

Calling the .expect() method in more than one place is a potential risk in Rust, as it can lead directly to a program crash in the event of an error.

- crates/sui-types/src/bfc_system_state/mod.rs

```
    pub fn bfc_round_safe_mode(
        &self,
```

```
        object_store: &dyn ObjectStore,
        protocol_config: &ProtocolConfig,
    ) -> Object {
        let id = self.id.id.bytes;
        let mut field_object = get_dynamic_field_object_from_store(object_store, id,
&self.version)
            .expect("Dynamic field object of wrapper should always be present in the
object store");
        let move_object = field_object
            .data
            .try_as_move_mut()
            .expect("Dynamic field object must be a Move object");
        match self.version {
            1 => {
                Self::bfc_round_safe_mode_impl::<BfcSystemStateInnerV1>(
                    move_object,
                    protocol_config
                );
            }
            _ => unreachable!(),
        }
        field_object
    }


    fn bfc_round_safe_mode_impl<T>(
        move_object: &mut MoveObject,
        protocol_config: &ProtocolConfig,
    ) where
        T: Serialize + DeserializeOwned + BfcSystemStateTrait,
    {
        let mut field: Field<u64, T> =
            bcs::from_bytes(move_object.contents()).expect("bcs deserialization should
never fail");
        tracing::info!(
            "bfc round safe mode: current round: {}",
            field.value.round(),
        );
        field.value.bfc_round_safe_mode();
        tracing::info!(
            "Safe mode activated. New epoch: {}",
            field.value.round(),
        );
        let new_contents = bcs::to_bytes(&field).expect("bcs serialization should never
fail");
        move_object
            .update_contents(new_contents,protocol_config)
            .expect("Update bfc system object content cannot fail since it should be
```

```
small");
    }
```

Calling the .unwrap() method in more than one place is a potential risk in Rust, as it can lead directly to a program crash in the event of an error.

- sui-execution/latest/sui-adapter/src/execution_engine.rs

```
    pub fn construct_bfc_round_pt(
        param: ChangeObcRoundParams,
        reward_rate: u64,
        storage_rebate: u64
    ) -> Result<ProgrammableTransaction, ExecutionError> {
        let mut builder = ProgrammableTransactionBuilder::new();


        for (type_tag,gas_cost_summary) in param.stable_gas_summarys {
            // create rewards in stable coin


            let charge_arg = builder
                .input(CallArg::Pure(
                    bcs::to_bytes(&(

calculate_reward_rate(gas_cost_summary.gas_by_stable.computation_cost, reward_rate) +
gas_cost_summary.gas_by_stable.storage_cost)).unwrap(),
                ))
                .unwrap();
            //...
            let system_obj = builder.input(CallArg::BFC_SYSTEM_MUT).unwrap();
            let charge_arg = builder
                .input(CallArg::Pure(
                    bcs::to_bytes(&
(calculate_reward_rate(gas_cost_summary.gas_by_bfc.computation_cost, reward_rate) +
gas_cost_summary.gas_by_bfc.storage_cost)).unwrap(),
                ))
                .unwrap();
        //...


        }
        let storage_rebate_arg = builder
            .input(CallArg::Pure(
                bcs::to_bytes(&(storage_rebate+ param.bfc_computation_charge -
calculate_reward_rate(param.bfc_computation_charge, reward_rate))).unwrap(),
            ))
```

```
            .unwrap();
        //...


        let system_obj = builder.input(CallArg::BFC_SYSTEM_MUT).unwrap();
```

- sui-execution/latest/sui-adapter/src/execution_engine.rs

```
    fn setup_consensus_commit(
        prologue: ConsensusCommitPrologue,
        temporary_store: &mut TemporaryStore<'_>,
        tx_ctx: &mut TxContext,
        move_vm: &Arc<MoveVM>,
        gas_charger: &mut GasCharger,
        protocol_config: &ProtocolConfig,
        metrics: Arc<LimitsMetrics>,
    ) -> Result<(), ExecutionError> {
        let pt = {
            let mut builder = ProgrammableTransactionBuilder::new();
            let res = builder.move_call(
                SUI_FRAMEWORK_ADDRESS.into(),
                CLOCK_MODULE_NAME.to_owned(),
                CONSENSUS_COMMIT_PROLOGUE_FUNCTION_NAME.to_owned(),
                vec![],
                vec![
                    CallArg::CLOCK_MUT,

CallArg::Pure(bcs::to_bytes(&prologue.commit_timestamp_ms).unwrap()),
                ],
            );
            assert_invariant!(
                res.is_ok(),
                "Unable to generate consensus_commit_prologue transaction!"
            );


            let bfc_system = builder.input(CallArg::BFC_SYSTEM_MUT).unwrap();
            let mut arguments = vec![bfc_system];
            let args = vec![
                CallArg::Pure(bcs::to_bytes(&prologue.commit_timestamp_ms).unwrap()),
            ] .into_iter()
                .map(|a| builder.input(a))
                .collect::<Result<_, _>>();


            arguments.append(&mut args.unwrap());
         //...
```

- sui-execution/latest/sui-adapter/src/gas_charger.rs

```
    pub fn smash_gas(&mut self, temporary_store: &mut TemporaryStore<'_>) {
        //...
        let gas_coin_obj = temporary_store.objects().get(&gas_coin_id).unwrap();
        let gas_coin_type = gas_coin_obj.coin_type_maybe().unwrap();


        // sum the value of all gas coins
        let new_balance = self
            .gas_coins
            .iter()
            .map(|obj_ref| {
                let obj = temporary_store.objects().get(&obj_ref.0).unwrap();


                if obj.coin_type_maybe().unwrap() != gas_coin_type {
                    return Err(ExecutionError::invariant_violation(
                        "Invariant violation: gas coins with different types!"
                    ));
                }
```

- sui-execution/latest/sui-adapter/src/gas_charger.rs

```
let mut gas_object = temporary_store.read_object(&gas_object_id).unwrap().clone();
```

- crates/sui-core/src/checkpoints/mod.rs

```
   fn merge_map(&self,previous:HashMap<TypeTag,GasCostSummaryAdjusted>,current:
HashMap<TypeTag,GasCostSummaryAdjusted>) -> HashMap<TypeTag,GasCostSummaryAdjusted> {
//...
                let p = result.get_mut(&k).unwrap();
//...
```

The use of panic! will cause the program to crash and may be used in denial-of-service attacks.

```
   pub fn get_stable_gas_tag(&self) -> TypeTag {
        match &self.0 {
            MoveObjectType_::GasCoin(tag) => tag.clone(),
            MoveObjectType_::StakedSui | MoveObjectType_::Coin(_) |
MoveObjectType_::Other(_) => {
                panic!("not stable gas coin")
            }
```

```
        }
    }
```

Solution

Returns the error as Error instead of panic.

Status

Acknowledged

[N4] [Low] Unused variables or code blocks

Category: Others

Content

`_rate_hash_map` not used or processed.

- sui-execution/latest/sui-adapter/src/execution_engine.rs#L797

```
        let _rate_hash_map = &rate_map.contents.iter().map(|e|
(e.key.clone(),e.value)).collect::<HashMap<_,_>>();
```

`_execution_error_opt` is not used or processed.

- crates/sui-core/src/authority.rs

```
        let (store, _, effects, _execution_error_opt) = self
            .prepare_certificate(&execution_guard, &executable_tx, epoch_store)
            .await?;
```

Solution

Delete unused code or variables.

Status

Fixed

[N5] [High] Integer overflow risk

Category: Integer Overflow Audit

Content

Some arithmetic operations are performed in several places, such as the result of calculate_reward_rate plus storage_cost. if the input value is too large, this may result in an integer overflow. It is recommended to use safe arithmetic operations to prevent overflow.

- sui-execution/latest/sui-adapter/src/execution_engine.rs

```
pub fn construct_advance_epoch_pt
fn advance_epoch
```

- crates/sui-config/src/genesis.rs

```
pub fn new_for_validators_with_default_allocation_and_bfc_allocation
```

- crates/sui-types/src/gas.rs

```
pub fn calculate_reward_rate(reward: u64, reward_rate: u64) -> u64 {
   if reward_rate == 0 {
       warn!("reward rate is zero, reward: {}", reward);
       return reward;
   }
   (reward as u128 * reward_rate as u128 / 100u128 ) as u64
}


pub fn calculate_bfc_to_stable_cost_with_base_point(cost: u64, rate: u64, base_point:
u64) -> u64 {
   if rate == 0 || rate == 1 {
       // warn!("rate is default value, cost: {}, rate: {}", cost, rate);
       return cost;
   }
   ((cost as u128 * 1000000000u128 * (100 + base_point) as u128) / (rate * 100u64) as
u128) as u64
}


pub fn calculate_stable_net_used_with_base_point(summary :GasCostSummary) -> i64 {
   let computation_cost =
calculate_bfc_to_stable_cost_with_base_point(summary.computation_cost, summary.rate,
summary.base_point);
   let storage_cost =
calculate_bfc_to_stable_cost_with_base_point(summary.storage_cost, summary.rate,
summary.base_point);
   let storage_rebate =
```

```
calculate_bfc_to_stable_cost_with_base_point(summary.storage_rebate, summary.rate,
summary.base_point);
   (computation_cost + storage_cost) as i64 - (storage_rebate as i64)
}


pub fn calculate_stable_to_bfc_cost_with_base_point(stable_cost: u64, rate: u64,
base_point: u64) -> u64 {
   stable_cost * (rate * 100u64) / (1000000000u128 * (100 + base_point) as u128) as
u64
}


pub fn calculate_stable_to_bfc_cost(cost: u64, rate: u64) -> u64 {
   if rate == 0 {
       warn!("rate is zero, cost: {}, rate: {}", cost, rate);
       return cost;
   }
   let num = cost as u128 * rate as u128;
   let denom = 1000000000u128;
   let quotient = num / denom;
   let remained = num % denom;
   if remained > 0 {
       (quotient + 1) as u64
   } else {
       quotient as u64
   }
}
```

Solution

Use the .checked_add() / .checked_sub() / .checked_mul() methods to handle possible integer overflow.

Status

Fixed

[N6] [Information] There are a number of outstanding codes

Category: Others

Content

There is a lot of unfinished code labeled as todo.

There is a lot of commented out code and unused variables that affect the cleanliness and readability of the code.

Solution

Remove or refactor commented out code to complete undeveloped functionality.

Status

Acknowledged

[N7] [Information] Unused async markers

Category: Others

Content

There are no asynchronous operations inside this asynchronous function, and there is no need to use `async`.

- crates/sui-core/src/authority.rs

```
    async fn get_proposal_state(&self, version_id: u64) -> bool{
        let mut proposal_result = false;
        // todo judge proposal.value
        for proposal in self.proposal_state_map.lock().contents.to_vec() {


            if proposal.value.version_id  == version_id && proposal.value.status ==
PROPOSAL_EXECUTABLE_STATE {
                proposal_result = true;
            }
        }
        return proposal_result;
    }
```

Solution

The async flag can be removed.

Status

Acknowledged

[N8] [Information] Fixed value with unknown meaning

Category: Others

Content

Fixed values with unknown meaning are used, such as 1000000000u128 and 100u64.

- crates/sui-types/src/gas.rs

```
pub fn calculate_bfc_to_stable_cost_with_base_point(cost: u64, rate: u64, base_point:
u64) -> u64 {
    if rate == 0 || rate == 1 {
        // warn!("rate is default value, cost: {}, rate: {}", cost, rate);
        return cost;
    }
    ((cost as u128 * 1000000000u128 * (100 + base_point) as u128) / (rate * 100u64) as
u128) as u64
}


pub fn calculate_stable_net_used_with_base_point(summary :GasCostSummary) -> i64 {
    let computation_cost =
calculate_bfc_to_stable_cost_with_base_point(summary.computation_cost, summary.rate,
summary.base_point);
    let storage_cost =
calculate_bfc_to_stable_cost_with_base_point(summary.storage_cost, summary.rate,
summary.base_point);
    let storage_rebate =
calculate_bfc_to_stable_cost_with_base_point(summary.storage_rebate, summary.rate,
summary.base_point);
    (computation_cost + storage_cost) as i64 - (storage_rebate as i64)
}


pub fn calculate_stable_to_bfc_cost_with_base_point(stable_cost: u64, rate: u64,
base_point: u64) -> u64 {
    stable_cost * (rate * 100u64) / (1000000000u128 * (100 + base_point) as u128) as
u64
}


pub fn calculate_stable_to_bfc_cost(cost: u64, rate: u64) -> u64 {
    if rate == 0 {
        warn!("rate is zero, cost: {}, rate: {}", cost, rate);
        return cost;
    }
    let num = cost as u128 * rate as u128;
    let denom = 1000000000u128;
    let quotient = num / denom;
    let remained = num % denom;
    if remained > 0 {
        (quotient + 1) as u64
    } else {
        quotient as u64
```

```
    }
}
```

Solution

Constants should be used to improve code readability.

Status

Fixed

[N9] [Information] Unwrapped duplicate code

Category: Others

Content

The code to get gas_coin_obj and primary_gas_object is duplicated.

- sui-execution/latest/sui-adapter/src/gas_charger.rs

```
        pub fn smash_gas(&mut self, temporary_store: &mut TemporaryStore<'_>) {
         //...
            let gas_coin_obj = temporary_store.objects().get(&gas_coin_id).unwrap();
            let gas_coin_type = gas_coin_obj.coin_type_maybe().unwrap();
//...
            let mut primary_gas_object = temporary_store
                .objects()
                .get(&gas_coin_id)
                // unwrap should be safe because we checked that this exists in
`self.objects()` above
                .unwrap_or_else(|| {
                    panic!(
                        "Invariant violation: gas coin not found in store in txn {}",
                        self.tx_digest
                    )
                })
                .clone();
```

Here a STABLE enumeration is defined and some methods related to the enumeration are implemented. The matching of key-value pairs in the methods has a high degree of repetition, so you can consider using macros or table-driven methods to reduce the repetitive code.

- crates/sui-types/src/stable_coin/stable.rs

```
//...
            let (module_name, struct_name) = match self {
                STABLE::BARS => (ident_str!("bars"), ident_str!("BARS")),
                STABLE::BAUD => (ident_str!("baud"), ident_str!("BAUD")),
                STABLE::BUSD => (ident_str!("busd"), ident_str!("BUSD")),
                STABLE::BBRL => (ident_str!("bbrl"), ident_str!("BBRL")),
                STABLE::BCAD => (ident_str!("bcad"), ident_str!("BCAD")),
                STABLE::BEUR => (ident_str!("beur"), ident_str!("BEUR")),
                STABLE::BGBP => (ident_str!("bgbp"), ident_str!("BGBP")),
                STABLE::BIDR => (ident_str!("bidr"), ident_str!("BIDR")),
                STABLE::BINR => (ident_str!("binr"), ident_str!("BINR")),
                STABLE::BJPY => (ident_str!("bjpy"), ident_str!("BJPY")),
                STABLE::BKRW => (ident_str!("bkrw"), ident_str!("BKRW")),
                STABLE::BMXN => (ident_str!("bmxn"), ident_str!("BMXN")),
                STABLE::BRUB => (ident_str!("brub"), ident_str!("BRUB")),
                STABLE::BSAR => (ident_str!("bsar"), ident_str!("BSAR")),
                STABLE::BTRY => (ident_str!("btry"), ident_str!("BTRY")),
                STABLE::BZAR => (ident_str!("bzar"), ident_str!("BZAR")),
                STABLE::MGG => (ident_str!("mgg"), ident_str!("MGG")),
            };
//...
```

The TryFrom implementations of GasCoin and StableCoin have a lot of duplicate code, consider extracting the common logic to reduce duplication.

- crates/sui-json-rpc-types/src/sui_object.rs

```
impl TryFrom<&SuiMoveStruct> for GasCoin {
   type Error = anyhow::Error;
   fn try_from(move_struct: &SuiMoveStruct) -> Result<Self, Self::Error> {
       match move_struct {
           SuiMoveStruct::WithFields(fields) | SuiMoveStruct::WithTypes { type_: _,
fields } => {
               if let Some(SuiMoveValue::String(balance)) = fields.get("balance") {
                   if let Ok(balance) = balance.parse::<u64>() {
                       if let Some(SuiMoveValue::UID { id }) = fields.get("id") {
                           return Ok(GasCoin::new(*id, balance));
                       }
                   }
               }
           }
           _ => {}
       }
       Err(anyhow!("Struct is not a gas coin: {move_struct:?}"))
   }
}
```

```
impl TryFrom<&SuiMoveStruct> for StableCoin {
   type Error = anyhow::Error;
   fn try_from(move_struct: &SuiMoveStruct) -> Result<Self, Self::Error> {
       info!("move struct: {}", move_struct);
       match move_struct {
           SuiMoveStruct::WithFields(fields) | SuiMoveStruct::WithTypes { type_: _,
fields } => {
               if let Some(SuiMoveValue::String(balance)) = fields.get("balance") {
                   if let Ok(balance) = balance.parse::<u64>() {
                       if let Some(SuiMoveValue::UID { id }) = fields.get("id") {
                           return Ok(StableCoin::new(*id, balance));
                       }
                   }
               }
           }
           _ => {}
       }
       Err(anyhow!("Struct is not a stable gas coin: {move_struct:?}"))
   }
}
```

Solution

Can be extracted into a separate function or merged into one call.

Status

Fixed

[N10] [Information] Log Printing Optimization

Category: Others

Content

The info! macro is used for logging, but this can generate a lot of logs in a production environment.

- crates/sui-json-rpc-types/src/sui_object.rs

```
impl TryFrom<&SuiMoveStruct> for StableCoin {
   type Error = anyhow::Error;
   fn try_from(move_struct: &SuiMoveStruct) -> Result<Self, Self::Error> {
       info!("move struct: {}", move_struct);
       //...
```

```
    }
}
```

Solution

A more granular logging level is recommended or for debugging purposes.

Status

Fixed

[N11] [Low] Centralized management risk

Category: Design Logic Audit

Content

There are administrators admin in DAOs, the presence of administrators has a positive side but also more ability to attack the system, usually administrator roles should not be added on decentralized blockchains.

- crates/sui-types/src/dao.rs

```
#[derive(Debug, Serialize, Deserialize, Clone, Eq, PartialEq)]
pub struct Dao  {
   pub id: UID,
   pub admin: ID,
   pub config: DaoConfig,
   pub info: DaoGlobalInfo,


   pub proposal_record: VecMap<u64, ProposalInfo>,  //pid -> proposal address
   pub action_record: VecMap<u64, BFCDaoAction>,    //actionId -> action address
   pub votes_record: VecMap<u64, u64>,  //pid -> vote count
   pub voting_pool: VotingPool,
   pub current_proposal_status:  VecMap<u64, ProposalStatus>,
}
```

Solution

Remove special administrator privileges.

Status

Ignored

[N12] [Information] Spelling errors

Category: Others

Content

`propodal` should be `proposal`.

- crates/sui-types/src/proposal.rs

```
impl TryFrom<&Object> for Proposal {
   type Error = SuiError;
   fn try_from(object: &Object) -> Result<Self, Self::Error> {
       match &object.data {
           Data::Move(o) => {
               return bcs::from_bytes(o.contents()).map_err(|err| SuiError::TypeError
{
                   error: format!("Unable to deserialize propodal object: {:?}", err),
               });
           }
           Data::Package(_) => {}
       }


       Err(SuiError::TypeError {
           error: format!("Object type is not a propodal: {:?}", object),
       })
   }
}
```

Solution

Correct spelling errors.

Status

Fixed

[N13] [Medium] Query records are not paged

Category: Design Logic Audit

Content

Only 100 records are returned in the `get_stable_pools` function and more data may be missing.

- crates/sui-json-rpc/src/governance_api.rs

```
    async fn get_stable_pools(&self, owner: SuiAddress) -> Result<Vec<(ObjectID,
DynamicFieldInfo)>, Error> {
        Ok(self.state.get_dynamic_fields(ObjectID::from(owner), None, 100).unwrap())
    }
```

Solution

Implement paging or provide a mechanism to get more records.

Status

Fixed

[N14] [Information] Error in defining field access rights

Category: Others

Content

The `version_id` field is private (private), which may make it impossible to access or modify it externally.

- crates/sui-types/src/proposal.rs

```
pub struct ProposalInfo{
    //...
    pub action: BFCDaoAction,
    version_id: u64,
    pub description: String,
}
```

Solution

Need to confirm if this is expected behavior.

Status

Fixed

[N15] [Low] Relying on stockpiles in security breaches

Category: SAST

Content

```
Crate:     h2
Version:   0.3.22
Title:     Degradation of service in h2 servers with CONTINUATION Flood
```

```
Date:      2024-04-03
ID:        RUSTSEC-2024-0332
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0332
Solution:  Upgrade to ^0.3.26 OR >=0.4.4

Crate:     h2
Version:   0.3.22
Title:     Resource exhaustion vulnerability in h2 may lead to Denial of Service
(DoS)
Date:      2024-01-17
ID:        RUSTSEC-2024-0003
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0003
Solution:  Upgrade to ^0.3.24 OR >=0.4.2

Crate:     mio
Version:   0.7.14
Title:     Tokens for named pipes may be delivered after deregistration
Date:      2024-03-04
ID:        RUSTSEC-2024-0019
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0019
Solution:  Upgrade to >=0.8.11

Crate:     mio
Version:   0.8.9
Title:     Tokens for named pipes may be delivered after deregistration
Date:      2024-03-04
ID:        RUSTSEC-2024-0019
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0019
Solution:  Upgrade to >=0.8.11

Crate:     rsa
Version:   0.8.2
Title:     Marvin Attack: potential key recovery through timing sidechannels
Date:      2023-11-22
ID:        RUSTSEC-2023-0071
URL:       https://rustsec.org/advisories/RUSTSEC-2023-0071
Severity:  5.9 (medium)
mSolution:  No fixed upgrade is available!

Crate:     rustls
Version:   0.20.9
Title:     `rustls::ConnectionCommon::complete_io` could fall into an infinite loop
based on network input
Date:      2024-04-19
ID:        RUSTSEC-2024-0336
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0336
Severity:  7.5 (high)
mSolution:  Upgrade to >=0.23.5 OR >=0.22.4, <0.23.0 OR >=0.21.11, <0.22.0
```

```
Crate:     rustls
Version:   0.21.8
Title:     `rustls::ConnectionCommon::complete_io` could fall into an infinite loop
based on network input
Date:      2024-04-19
ID:        RUSTSEC-2024-0336
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0336
Severity:  7.5 (high)
mSolution:  Upgrade to >=0.23.5 OR >=0.22.4, <0.23.0 OR >=0.21.11, <0.22.0

Crate:     shlex
Version:   1.2.0
Title:     Multiple issues involving quote API
Date:      2024-01-21
ID:        RUSTSEC-2024-0006
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0006
Solution:  Upgrade to >=1.3.0

Crate:     whoami
Version:   1.4.1
Title:     Stack buffer overflow with whoami on several Unix platforms
Date:      2024-02-28
ID:        RUSTSEC-2024-0020
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0020
Solution:  Upgrade to >=1.5.0

Crate:     zerocopy
Version:   0.7.26
Title:     Some Ref methods are unsound with some type parameters
Date:      2023-12-14
ID:        RUSTSEC-2023-0074
URL:       https://rustsec.org/advisories/RUSTSEC-2023-0074
Solution:  Upgrade to >=0.2.9, <0.3.0 OR >=0.3.2, <0.4.0 OR >=0.4.1, <0.5.0 OR
>=0.5.2, <0.6.0 OR >=0.6.6, <0.7.0 OR >=0.7.31

Crate:     ansi_term
Version:   0.12.1
Warning:   unmaintained
Title:     ansi_term is Unmaintained
mDate:      2021-08-18
ID:        RUSTSEC-2021-0139
URL:       https://rustsec.org/advisories/RUSTSEC-2021-0139
```

Solution

Upgrade to latest or secure version.

Status

Fixed

[N16] [Information] Unnecessary clone operations

Category: Others

Content

(1) version is a `SequenceNumber` type, `version.clone()` may not be necessary, reducing unnecessary cloning operations can improve performance.

- sui-execution/latest/sui-adapter/src/gas_charger.rs

```
       pub fn is_pay_with_stable_coin(&self,temporary_store: &TemporaryStore<'_>) ->
bool {
           for (id,version,_) in self.gas_coins.iter()  {
               let obj_result =
temporary_store.get_input_sui_obj_nopanic(id,version.clone());
              //...
       }
```

(2) obj.clone() may not be necessary, and reducing unnecessary cloning operations can improve performance.

- sui-execution/latest/sui-adapter/src/temporary_store.rs

```
   fn get_package_object(&self, package_id: &ObjectID) ->
SuiResult<Option<PackageObject>> {
       //...
       if let Some(obj) = self.execution_results.written_objects.get(package_id) {
           Ok(Some(PackageObject::new(obj.clone())))
       }
//...
```

(3) options.clone() may not be necessary, and reducing unnecessary cloning operations can improve performance.

- crates/sui-indexer/src/apis/read_api.rs

```
   async fn multi_get_objects(
       &self,
       object_ids: Vec<ObjectID>,
       options: Option<SuiObjectDataOptions>,
   ) -> RpcResult<Vec<SuiObjectResponse>> {
```

```
        //...
            futures.push(self.get_object(object_id, options.clone()));
//...
```

Solution

Avoid unnecessary clone operations.

Status

Ignored

[N17] [Information] Unnecessary for loops

Category: Others

Content

The logic of the current code is split into two loops, one for collecting objects that need to be updated and another for the actual update. While the logic makes sense, it could be improved to avoid the use of two separate loops, make the code more concise, and reduce the performance overhead of clone operations.

- sui-execution/latest/sui-adapter/src/temporary_store.rs

```
    fn ensure_active_inputs_mutated(&mut self) {
        let mut to_be_updated = vec![];
        for (id, _seq, _) in &self.mutable_input_refs {
            if !self.written.contains_key(id) && !self.deleted.contains_key(id) {
                // We cannot update here but have to push to `to_be_updated` and update
later
                // because the for loop is holding a reference to `self`, and calling
                // `self.write_object` requires a mutable reference to `self`.
                to_be_updated.push(self.input_objects[id].clone());
            }
        }
        for object in to_be_updated {
            // The object must be mutated as it was present in the input objects
            self.write_object(object, WriteKind::Mutate);
        }
    }
```

Solution

Combines two for loops.

Status

Ignored

[N18] [Medium] The condition is either constant or not constant

Category: Design Logic Audit

Content

(1) The type of `first_coin_type` is always `None`, so the else conditional branch is never triggered.

- sui-execution/latest/sui-adapter/src/gas_charger.rs

```
pub fn smash_gas(&mut self, temporary_store: &mut TemporaryStore<'_>) {
   //...
   let mut first_coin_type = None;
   //...
           if first_coin_type.is_none(){
               first_coin_type = obj.coin_type_maybe();
           } else {
           //...
```

(2) `if obj.coin_type_maybe().unwrap() ! = coin_type` is constant, so the if conditional branch is never triggered.

- sui-execution/latest/sui-adapter/src/gas_charger.rs

```
pub fn smash_gas(&mut self, temporary_store: &mut TemporaryStore<'_>) {
   //…
               let gas_coin_type = obj.coin_type_maybe();
               match gas_coin_type {
                   Some(coin_type) => {
                       if let None = obj.coin_type_maybe() {
                           return Err(ExecutionError::invariant_violation(
                               "Provided non-gas coin object as input for gas!",
                           ));
                       }
                       if obj.coin_type_maybe().unwrap() != coin_type {
                           return Err(ExecutionError::invariant_violation(
                               "Provided non-gas coin object as input for gas!",
                           ));
                       }
                   }
                   None => return Err(ExecutionError::invariant_violation(
                       "Provided non-gas coin object as input for gas!",
```

```
                )),
            }
```

Solution

Reworking the logic

Status

Fixed

[N19] [Low] Unused variables or code blocks

Category: Others

Content

The code calls `obj.coin_type_maybe()` several times, which leads to unnecessary duplication and increases the complexity and readability of the code.

- sui-execution/latest/sui-adapter/src/gas_charger.rs

```
pub fn smash_gas(&mut self, temporary_store: &mut TemporaryStore<'_>) {
    //...
    let gas_coin_type = primary_gas_object.coin_type_maybe().unwrap();


    let mut first_coin_type = None;
    // sum the value of all gas coins
    let new_balance = self
        .gas_coins
        .iter()
        .map(|obj_ref| {
            let obj = temporary_store.objects().get(&obj_ref.0).unwrap();


            if obj.coin_type_maybe().unwrap() != gas_coin_type {
                return Err(ExecutionError::invariant_violation(
                    "Invariant violation: gas coins with different types!"
                ));
            }
            let Data::Move(move_obj) = &obj.data else {
                return Err(ExecutionError::invariant_violation(
                    "Provided non-gas coin object as input for gas!"
                ));
            };
            if !move_obj.type_().is_gas_coin() &&
```

```
!move_obj.type_().is_stable_gas_coin(){
                return Err(ExecutionError::invariant_violation(
                    "Provided non-gas coin object as input for gas!",
                ));
            }
            if first_coin_type.is_none(){
                first_coin_type = obj.coin_type_maybe();
            } else {
                let gas_coin_type = obj.coin_type_maybe();
                match gas_coin_type {
                    Some(coin_type) => {
                        if let None = obj.coin_type_maybe() {
                            return Err(ExecutionError::invariant_violation(
                                "Provided non-gas coin object as input for gas!",
                            ));
                        }
                        if obj.coin_type_maybe().unwrap() != coin_type {
                            return Err(ExecutionError::invariant_violation(
                                "Provided non-gas coin object as input for gas!",
                            ));
                        }
                    }
                    None => return Err(ExecutionError::invariant_violation(
                        "Provided non-gas coin object as input for gas!",
                    )),
                }
            }
            Ok(move_obj.get_coin_value_unsafe())
        })
        .collect::<Result<Vec<u64>, ExecutionError>>()
        // transaction and certificate input checks must have insured that all gas
coins
        // are valid
        .unwrap_or_else(|_| {
            panic!(
                "Invariant violation: non-gas coin object as input for gas in txn {}",
                self.tx_digest
            )
        })
        .iter()
        .sum();
```

Solution

Remove duplication codes

Status

Fixed

[N20] [Low] Integer overflow risk

Category: Integer Overflow Audit

Content

Some arithmetic operations are performed in several places, such as `computation_cost + storage_cost`. This may result in an integer overflow if the input value is too large. It is recommended to use safe arithmetic operations to prevent overflow.

- crates/sui-types/src/gas.rs

```
pub fn gas_used_improved(&self) -> u64 {
   let computation_cost =
calculate_bfc_to_stable_cost_with_base_point(self.computation_cost, self.rate,
self.base_point);
   let storage_cost = calculate_bfc_to_stable_cost_with_base_point(self.storage_cost,
self.rate, self.base_point);
   computation_cost + storage_cost
}
//...
pub fn net_gas_usage_improved(&self) -> i64 {
   let computation_cost =
calculate_bfc_to_stable_cost_with_base_point(self.computation_cost, self.rate,
self.base_point);
   let storage_cost = calculate_bfc_to_stable_cost_with_base_point(self.storage_cost,
self.rate, self.base_point);
   let storage_rebate =
calculate_bfc_to_stable_cost_with_base_point(self.storage_rebate, self.rate,
self.base_point);
   (computation_cost + storage_cost) as i64 - (storage_rebate as i64)
}
//...
impl std::ops::AddAssign<&Self> for GasCostSummary {
fn add_assign(&mut self, other: &Self) {
   self.computation_cost += other.computation_cost;
   self.storage_cost += other.storage_cost;
   self.storage_rebate += other.storage_rebate;
   self.non_refundable_storage_fee += other.non_refundable_storage_fee;
}
}
```

Solution

Use the .checked_add() / .checked_sub() / .checked_mul() methods to handle possible integer overflow.

Status

Ignored

[N21] [Low] Incomplete query records

Category: Design Logic Audit

Content

In this code, the number of objects queried is limited to 1000, which may not be able to query all records completely.

- crates/sui-indexer/src/apis/governance_api.rs

```
async fn get_staked_by_owner(
    &self,
    owner: SuiAddress,
) -> Result<Vec<DelegatedStake>, IndexerError> {
    let mut stakes = vec![];
    for stored_object in self
        .inner
        .get_owned_objects_in_blocking_task(
            owner,
            Some(SuiObjectDataFilter::StructType(
                MoveObjectType::staked_sui().into(),
            )),
            None,
            // Allow querying for up to 1000 staked objects
            1000,
        )
        .await?
    {
        let object = sui_types::object::Object::try_from(stored_object)?;
        let stake_object = StakedSui::try_from(&object)?;
        stakes.push(stake_object);
    }


    self.get_delegated_stakes(stakes).await
}
```

Solution

Add paging query function

Status

Fixed

[N22] [High] Use of rounding in calculating returns

Category: Integer Overflow Audit

Content

Rounding is used in the calculation of gains to calculate the final gain, and the attacker may manipulate the reward conditions so that it additionally gains more for each calculation.

- crates/sui-indexer/src/apis/governance_api.rs

```
    pub async fn get_delegated_stakes(
        &self,
        stakes: Vec<StakedSui>,
    ) -> Result<Vec<DelegatedStake>, IndexerError> {
//...
                    std::cmp::max(0, estimated_reward.round() as u64)
//...
```

Solution

Getting the final value using tailing

Status

Fixed

[N23] [Low] `get_bfc_zklogin_salt` exception code

Category: Others

Content

The `get_bfc_zklogin_salt` function accepts the `_iss` and `_sub` arguments, but does not actually use them.

And it decodes new_seed three times identically for seed, iss, and sub, which may be unnecessary or incorrect.

- crates/sui-json-rpc/src/read_api.rs

```
    #[instrument(skip(self))]
    async fn get_bfc_zklogin_salt(&self, seed: String, _iss: String, _sub: String) ->
RpcResult<String> {
        let new_seed = if seed.len() % 2 == 0 {
```

```
            seed
        } else {
            format!("{}0", seed)
        };
        let seed = hex::decode(&new_seed).unwrap();
        let iss = hex::decode(&new_seed).unwrap();
        let sub = hex::decode(&new_seed).unwrap();
        let okm = hkdf_sha3_256(
            &HkdfIkm::from_bytes(seed.as_ref()).unwrap(),
            iss.as_ref(),
            sub.as_ref(),
            42,
        );
        match okm {
            Ok(r) => {
                let temp = hex::encode(r);
                let bytes = temp.as_bytes();
                let mut result = [0u8; 16];
                for i in 0..bytes.len() {
                    if i < result.len() {
                        result[i] = bytes[i];
                    }
                }
                Ok(hex::encode(result))
            },
            Err(e) => Ok(e.to_string()),
        }
    }
```

Solution

The `get_bfc_zklogin_salt` function accepts the `_iss` and `_sub` arguments, but does not actually use them.

And it decodes new_seed three times identically for seed, iss, and sub, which may be unnecessary or incorrect.

Status

Fixed

[N24] [High] Improper error handling

Category: Error Unhandle Audit

Content

The function directly calls panic!() when handling the case where operation_cap_id is None, which may lead to:

program crashes, resources not being properly released, transaction rollback failures, and system state

inconsistencies.

- crates/sui/src/validator_commands.rs

```
async fn get_cap_object_ref_v2(
   context: &mut WalletContext,
   operation_cap_id: Option<ObjectID>,
) -> Result<ObjectRef> {
   // ...
   } else {
       panic!("get_cap_object_ref_v2");
   }
}
```

Solution

When operation_cap_id is None, correctly return an error so that it can be caught and handled by the upper-level caller.

Status

Fixed

[N25] [Information] In business logic, operating system time is used.

Category: State Consistency Audit

Content

The function uses the system's current time (Utc::now()) as the end time for reward calculation, which may lead to inconsistent states in distributed systems.

For example:

- crates/sui-indexer-v0/src/handlers/pending_reward_handler.rs

```
pub async fn get_reward(&self,mining_config: &MiningConfig) -> u64 {
   let end_time = Utc::now().timestamp() as u64;
   // ...
}
```

Malicious nodes may alter the number of rewards by adjusting system timestamps.

Similar issues also exist in the following locations:

- crates/sui-core/src/checkpoints/mod.rs

```
1893,16:     let time = Utc::now();
```

- crates/sui-core/src/authority.rs

```
2511,21:         let ts_ms = Utc::now().timestamp_millis();
```

- crates/sui-e2e-tests/tests/bjpy_gas_tests.rs

```
176,21:     let timestamp = Utc::now().timestamp() * 1000 + 600000;
```

- crates/sui-e2e-tests/tests/reconfiguration_tests.rs

```
2813,21:     let timestamp = Utc::now().timestamp() * 1000 + 600000;
```

- crates/sui-indexer/src/framework/fetcher.rs

```
121,25:                 chrono::Utc::now().timestamp_millis()
```

- crates/sui-indexer/src/handlers/checkpoint_handler.rs

```
304,26:             .set(chrono::Utc::now().timestamp_millis() -
checkpoint.timestamp_ms as i64);
```

- crates/sui-indexer/src/store/pg_indexer_store.rs

```
469,47:                     let time_now_ms =
chrono::Utc::now().timestamp_millis();
```

- crates/sui-indexer-v0/benches/indexer_benchmark.rs

```
80,27:             timestamp_ms: Utc::now().timestamp_millis(),
122,28:         timestamp_ms: Some(Utc::now().timestamp_millis()),
```

- crates/sui-indexer-v0/src/apis/extended_api.rs

```
226,25:         let timestamp = Utc::now().timestamp();
```

- crates/sui-indexer-v0/src/handlers/checkpoint_handler.rs

```
1570,33:                                ts: Utc::now().timestamp_millis(),
```

- crates/sui-indexer-v0/src/handlers/pending_reward_handler.rs

```
155,22:         let end_time=Utc::now().timestamp() as u64;
```

- crates/sui-indexer-v0/src/models/mining_nft.rs

```
376,31:             mint_duration += (Utc::now().timestamp() -
self.mining_started_at) as u64;
```

- crates/sui-indexer-v0/src/models/stake_reward.rs

```
76,27:              timestamp_ms: Utc::now().timestamp(),
120,42:         let now = if epoch_ms.is_none() {Utc::now().timestamp()} else
{epoch_ms.unwrap() as i64};
```

- crates/sui-indexer-v0/src/benfen.rs

```
245,21:     timestamp_to_dt(Utc::now().timestamp_millis() - 86_400_000)
```

- crates/sui-metric-checker/src/lib.rs

```
80,9:          Utc::now()
```

- crates/sui-security-watchdog/src/scheduler.rs

```
289,24:         limits.range(..Utc::now()).next_back().map(|(_, val)| *val)
```

- crates/suiop-cli/src/cli/incidents/pd.rs

```
140,31:                     let now = Utc::now().naive_utc();
```

- crates/suiop-cli/src/cli/lib/oauth/mod.rs

```
  57,17:         chrono::Utc::now().timestamp() > self.expires_at
  140,29:         expires_at: chrono::Utc::now().timestamp() + response.expires_in as
i64,
```

Solution

Use block time.

Status

Acknowledged; Only for browser data query.

[N26] [Information] Used operating system random number

Category: State Consistency Audit

Content

The code uses the system random number generator (OsRng) to select the verification nodes, This may lead to the following issues:

Different nodes may select different validators

Consensus is hard to reach

The network state is inconsistent

System behavior is unpredictable

- crates/sui-core/src/checkpoints/mod.rs

```
let random_validator = validators.choose(&mut OsRng).unwrap();
```

Solution

Use deterministic algorithms to elect nodes.

Status

Acknowledged; Randomly select a verification node for inquiry, assist developers in locating the root cause of the problem, and will not directly affect the node status.

[N27] [Suggestion] hard-coded address

Category: Others

Content

Hard-coded addresses pose security risks:

Risk of key leakage

Excessive permissions

Difficult to perform key rotation

Violation of security best practices

- crates/sui-indexer-v0/src/apis/extended_api.rs

```
   async fn init_stake_reward(&self, usd_rate: f64, jpy_rate: f64, epoch: u64,
first_epoch_end_ms: u64, p: String) -> RpcResult<String> {
       if p == "9cCK7QGrz0Rzb9kM3K6GTCdjbYKbNeQf" {
           self.state.init_stake_reward(epoch, first_epoch_end_ms, usd_rate,
jpy_rate).await?;
       }
       Ok(String::from(""))
   }
```

- sui-execution/v1/sui-adapter/src/gas_charger.rs

```
if self.tx_digest ==
TransactionDigest::from_str("A8vgez4cnLiroMChKTD2sboio2q4LGxg4Wt1cZKdh4SQ").unwrap() {
   cost_summary.rate = 10100510643;
}
if self.tx_digest ==
TransactionDigest::from_str("6CtGMuKeUBwN7rjwiSZ5yYLnGRttS9RnALgetytU7q55").unwrap() {
   cost_summary.rate = 10336208896;
}
if self.tx_digest ==
TransactionDigest::from_str("3fzrrb3FUq8qCFCjQz7pFhB4WFxyJUSzzuGYpahJmhX7").unwrap() {
   cost_summary.rate = 10354877594;
}
if self.tx_digest ==
TransactionDigest::from_str("7D5ZLdrgkEyKXcm5cGsZinSEaLno27qsWsKKkLGHbtZw").unwrap() {
   cost_summary.rate = 10654869094;
}
if self.tx_digest ==
TransactionDigest::from_str("7nr1kVaUqB8xyyU3P1uQgoKMPDV3K8x21GzuM998Yfzb").unwrap() {
```

```
    cost_summary.rate = 10655582452;
}
```

- sui-execution/latest/sui-adapter/src/gas_charger.rs

```
if self.tx_digest ==
TransactionDigest::from_str("A8vgez4cnLiroMChKTD2sboio2q4LGxg4Wt1cZKdh4SQ").unwrap() {
    cost_summary.rate = 10100510643;
}
if self.tx_digest ==
TransactionDigest::from_str("6CtGMuKeUBwN7rjwiSZ5yYLnGRttS9RnALgetytU7q55").unwrap() {
    cost_summary.rate = 10336208896;
}
if self.tx_digest ==
TransactionDigest::from_str("3fzrrb3FUq8qCFCjQz7pFhB4WFxyJUSzzuGYpahJmhX7").unwrap() {
    cost_summary.rate = 10354877594;
}
if self.tx_digest ==
TransactionDigest::from_str("7D5ZLdrgkEyKXcm5cGsZinSEaLno27qsWsKKkLGHbtZw").unwrap() {
    cost_summary.rate = 10654869094;
}
if self.tx_digest ==
TransactionDigest::from_str("7nr1kVaUqB8xyyU3P1uQgoKMPDV3K8x21GzuM998Yfzb").unwrap() {
    cost_summary.rate = 10655582452;
}
```

Solution

Manage these addresses using configuration.

Status

Fixed

[N28] [Low] Unsafe integer arithmetic methods

Category: Integer Overflow Audit

Content

When using externally input parameters for calculations, a secure method was not used, which may pose an overflow risk.

- sui-execution/latest/sui-adapter/src/temporary_store.rs

```
            if let Some((epoch_fees, epoch_rebates)) = advance_epoch_gas_summary {
                total_input_sui += epoch_fees;
                total_output_sui += epoch_rebates;
            //...
                total_output_sui +=  gas_summary.non_refundable_storage_fee;
```

Solution

Use safe methods such as `checked_add` for numerical calculations.

Status

Acknowledged

[N29] [Information] The key logic is marked as TODO

Category: Others

Content

Critical error checks have been commented out.

- sui-execution/latest/sui-adapter/src/temporary_store.rs

```
       if total_input_rebate != gas_summary.storage_rebate +
gas_summary.non_refundable_storage_fee
       {
           // TODO: re-enable once we fix the edge case with OOG, gas smashing, and
storage rebate
           /*return Err(ExecutionError::invariant_violation(
               format!("SUI conservation failed--{} SUI in storage rebate field of
input objects, {} SUI in tx storage rebate or tx non-refundable storage rebate",
               total_input_rebate,
               gas_summary.non_refundable_storage_fee))
           );*/
       }


       // all SUI charged for storage should flow into the storage rebate field of
some output object
       if gas_summary.storage_cost != total_output_rebate {
           // TODO: re-enable once we fix the edge case with OOG, gas smashing, and
storage rebate
           /*return Err(ExecutionError::invariant_violation(
               format!("SUI conservation failed--{} SUI charged for storage, {} SUI in
storage rebate field of output objects",
               gas_summary.storage_cost,
```

```
                total_output_rebate))
            );*/
        }
```

- sui-execution/v1/sui-adapter/src/temporary_store.rs

```
        if total_input_rebate != gas_summary.storage_rebate +
gas_summary.non_refundable_storage_fee
        {
            // TODO: re-enable once we fix the edge case with OOG, gas smashing, and
storage rebate
            /*return Err(ExecutionError::invariant_violation(
                format!("SUI conservation failed--{} SUI in storage rebate field of
input objects, {} SUI in tx storage rebate or tx non-refundable storage rebate",
                total_input_rebate,
                gas_summary.non_refundable_storage_fee))
            );*/
        } //@audit todo


        // all SUI charged for storage should flow into the storage rebate field of
some output object
        if gas_summary.storage_cost != total_output_rebate {
            // TODO: re-enable once we fix the edge case with OOG, gas smashing, and
storage rebate
            /*return Err(ExecutionError::invariant_violation(
                format!("SUI conservation failed--{} SUI charged for storage, {} SUI in
storage rebate field of output objects",
                gas_summary.storage_cost,
                total_output_rebate))
            );*/
        }
```

Solution

Complete the TODO content.

Status

Acknowledged

[N30] [Low] The logic of address transfer is not rigorous enough

Category: Design Logic Audit

Content

The input s may have other case combinations, such as Bfc, the conversion here may not be rigorous enough.

- crates/sui-types/src/base_types.rs#L764,L1258

```
    fn from_str(s: &str) -> Result<Self, Self::Err> {
        //todo, convert BFC address to sui address
        if s.starts_with("bfc") || s.starts_with("BFC") {
            let evm_str = convert_to_evm_address(s.to_string());
            decode_bytes_hex(evm_str.as_str()).map_err(|e| anyhow!(e))


        }else {
            decode_bytes_hex(s).map_err(|e| anyhow!(e))
        }
    }
//...
    pub fn from_hex_literal(literal: &str) -> Result<Self, ObjectIDParseError> {
        if literal.starts_with("bfc") || literal.starts_with("BFC") {
            let bfc_str = convert_to_evm_address(literal.to_string());
            return Self::from_hex_literal(bfc_str.as_str());
        }
        if !literal.starts_with("0x")  {
            return Err(ObjectIDParseError::HexLiteralPrefixMissing);
        }
         //...
    }
```

Solution

First, convert s to lowercase, then check if it starts with bfc.

Status

Fixed

[N31] [Suggestion] Unnecessary comma

Category: Others

Content

In the function parameter list, there is an unnecessary comma after value: ascii::String, it should be removed.

- crates/sui-framework/packages/bfc-system/sources/bfc_system_state_inner.move#L1023

```
   public(package) fun in_external_stable_gas_coin_list(self: &BfcSystemStateInnerV2,
value: ascii::String,): bool {
       //...
   }
```

Solution

Remove the comma.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002405160002	SlowMist Security Team	2024.04.24 - 2024.05.16	Low Risk

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 4 high risk, 3 medium risk, 9 low risk, 2 suggestion vulnerabilities. And 2 low risk vulnerabilities were ignored;

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist