Creek

Audit Report

contact@bitslab.xyz

https://twitter.com/movebit_

Mon Dec 15 2025

Creek Audit Report

1 Executive Summary

1.1 Project Information

Description	Creek is a lending protocol oriented around XAUM, consisting of two main components. Users can stake XAUM to receive gr and gy tokens. The gr tokens can then be used as collateral within the lending protocol to borrow GUSD. The lending module is forked from the Scallop protocol, with modifications limited primarily to the supported tokens. Additionally, the project team has implemented a dedicated oracle mechanism to ensure accurate price feeds for XAUM.
Type	DeFi
Auditors	jolyon, s3cunda
Timeline	Mon Oct 27 2025 - Mon Dec 15 2025
Languages	Move
Platform	Sui
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/Creek-Finance/core
Commits	31f6cf255f0a834d14d98340124514a2e2c4f5e8 56e8e6588de820ff4d3172e7aa9f428ed9be34f2 c6cf577709b357420582007033db653c5f78aad6 ffd5f0808021c65fe05adf3985d7325ac795681f

1.2 Files in Scope

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

ID	File	SHA-1 Hash
MOV1	contracts/xaum_indicator/pyth_adapter/Move.toml	e4b9f26e3b41f752df0ddf8693bed5aa281a1b1a
XIPA	contracts/xaum_indicator/pyth_adapter/sources/xaum_indicator_pyth_adapter.move	bc4ad1be71a717551939fc8f97b8cffa62d82bfb
MOV2	contracts/xaum_indicator/core/Move.toml	fc0fac87e23f83607c23b4a287ede3e83b41c623
XIC	contracts/xaum_indicator/core/sources/xaum_indicator_core.move	6d67478f7344a2bd3a0f30d5fa157cb160ac7911
MOV3	contracts/libs/coin_decimals_registry/Move.toml	17d598929ce6e49fa03389216857cca8ac485370
CDR	contracts/libs/coin_decimals_registry/sources/coin_decimals_registry.move	aa4966ba63984521f1b11e5419c4a0346611d90d
MOV4	contracts/libs/x/Move.toml	27c23010f8890ae188231638a03092002dfdd462
BBA	contracts/libs/x/sources/balance_bag.move	e3e31f129863c0073a93a344d822d9ff65103fd9
WIT	contracts/libs/x/sources/witness.move	fe281f012af38f794159edb8497667529c993c16

SBA	contracts/libs/x/sources/supply_bag.move	d1d619a250caf2223e27e7af9e1741e00854f996
ATA	contracts/libs/x/sources/ac_table.move	99c3efc2425a5b9eb2e4c736fcf55615addc84f2
OWN	contracts/libs/x/sources/ownership.move	7987f3937f38c0f3430aeb02c35703375899ef91
WTA	contracts/libs/x/sources/wit_table.move	ac55efa9bdefc6f61f50393c1aa38a5d0d328404
OTLV	contracts/libs/x/sources/one_time_lock_value.move	0309c250e5677daa973c426f092acd9267f5fcda
MOV5	contracts/libs/math/Move.toml	d4d689eab27b912f07014766227eb910a9e2a7f1
FP3	contracts/libs/math/sources/fixed_point32.move	a7b555ff063d3994f7990d863ef7cc8322a81196
U25	contracts/libs/math/sources/u256.move	92d942e8332a8d35b0739f9366b24133108d295b
U12	contracts/libs/math/sources/u128.move	883b054814c7bd6f4454dd7d8d8b53c33bf074b9
U64	contracts/libs/math/sources/u64.move	43c4546289fc0b2cc0cd9bf0d3ccc031090bfc17
MOV6	contracts/coins/coin_gr/Move.toml	f64552ddbfaa75a5fe7d8a90c3c4327d6447049d
CGR	contracts/coins/coin_gr/sources/coin_gr.move	58dbf11b28e1874a4af397a79f990ff2796b9c11

MOV7	contracts/coins/coin_gy/Move.toml	3daccec8d5d323e2d4e8f571c63d5c2b9424e7ca
CGY	contracts/coins/coin_gy/sources/coin_gy.move	3ae20bbbe79f435e6b4258ea4b1629553dc1fd2e
MOV8	contracts/coins/coin_gusd/Move.toml	a8b8b02e30be8b1a7e58cab3a5c5486c1d86fd35
CGU	contracts/coins/coin_gusd/sources/coin_gusd.move	7a2ca2fee85028d755565dde1cca2ea2cf9d2b4c
MOV9	contracts/sui_x_oracle/manual_rule/Move.toml	ca704d875c1442a9740ef02523f651b4dab0d18d
RUL	contracts/sui_x_oracle/manual_rule/sources/rule.move	c08b913e5e6874336750a6ff767a521a3af15cee
MOV10	contracts/sui_x_oracle/x_oracle/Move.toml	b1e4e20700f2407c5d4ea31de9dba180cf34956d
XOR	contracts/sui_x_oracle/x_oracle/sources/x_oracle.move	76a4e2b49432fde1cfc294bc6ae83bb0ce026cb3
PFE	contracts/sui_x_oracle/x_oracle/sources/price_feed.move	cbd511b7b60fbbb8f8e8a791bbc07a0c2e273c6c
PUP	contracts/sui_x_oracle/x_oracle/sources/price_update_policy.move	caba44d11c1ed555c3748fea1c956a23faa5b6db
MOV11	contracts/sui_x_oracle/supra_rule/Move.toml	b226529ed95261eb60d5b6ee8d3ab85eea9ef0f1
SAD	contracts/sui_x_oracle/supra_rule/sources/supra_adaptor.move	545a0c3281116a2103ffa878bbe1e0284326320b

RUL1	contracts/sui_x_oracle/supra_rule/sources/rule.move	9f281a9ecb0832a1b19cd914f7026dea7adfc535
SRE	contracts/sui_x_oracle/supra_rule/sources/supra_registry.move	ca082b86521ef16ff43bb5a40f0f7bd6b3c54fbe
MOV12	contracts/sui_x_oracle/switchboard_on_demand_rule/Move.toml	30522bdf428097b91f6da62305d7c3f35c89bac6
SRE1	contracts/sui_x_oracle/switchboard_on_demand_rule/sources/switchboard_registry.move	396cb009e7a6373ac79465a66b8196f0d6885a37
SAD1	contracts/sui_x_oracle/switchboard_on_demand_rule/sources/switchboard_adaptor.move	3e8bec0a05027b09e50d59e9444476e722d809ec
RUL2	contracts/sui_x_oracle/switchboard_on_demand_rule/sources/rule.move	ad104e70908f8ad17550e823b17f708ebf387f86
MOV13	contracts/sui_x_oracle/pyth_rule/Move.toml	6b9e6094baa33c173dc2263473a472a7debe3dc4
RUL3	contracts/sui_x_oracle/pyth_rule/sources/rule.move	69992fe1ef8d0951c2af29ff862f62a08bbf2159
PRE	contracts/sui_x_oracle/pyth_rule/sources/pyth_registry.move	6415887d0a27f3b95ec0784c15412a5cb8a5f4ac
PAD	contracts/sui_x_oracle/pyth_rule/sources/pyth_adaptor.move	d3636a651bf1f2efaefb85dadab05e5e9478ab63
MOV14	contracts/protocol/Move.toml	484887a5fdaec07f18b756d806cb5cf953a9c15d

OBL	contracts/protocol/sources/obligation/obligation.move	a5a29c157f9550dc9ed0815dc6bc6421bdf0b882
ODE	contracts/protocol/sources/obligation/obligation_debts.move	8ba8fb6a4e4d9d4e6189337fa33345762324ead2
OAC	contracts/protocol/sources/obligation/obligation_access.move	b72dff5f6b1f4b2e068d5f44d332e6e0ba843601
OCO	contracts/protocol/sources/obligation/obligation_collaterals.move	555858657b1f7be959be3cd109ec69e14fe5edeb
MDK	contracts/protocol/sources/market/market_dynamic_keys.move	214c2887987d62d8d874335c9c9ee49cb8d28779
BDY	contracts/protocol/sources/market/borrow_dynamics.move	098cc56acf7f2f4cff50746e001eae971d8b23ff
CST	contracts/protocol/sources/market/collateral_stats.move	2bea5f39f8681db6d1e4337824f3b217db690e53
RES	contracts/protocol/sources/market/reserve.move	ff3530d0fe364456c7586a06ac3a8d3af5f70425
RMO	contracts/protocol/sources/market/risk_model.move	3f239b6df702f8c4f1b7d37af83ff2d3356a983d
MAR	contracts/protocol/sources/market/market.move	1306f3fca9e7aca13494edddb37a25a3495ad912
AAS	contracts/protocol/sources/market/asset_active_state.move	7accfefffc9aeb0cb2258c132965105444ea78ee
IMO	contracts/protocol/sources/market/interest_model.move	fe77fcd6813aacfac1fd46a3b3b348e45d098180

LIM	contracts/protocol/sources/market/limiter.move	e4438628e999d3e7a029fe8a1e5fd6e27b0e339e
SMA	contracts/protocol/sources/staking/staking_manager.move	366fec72e6de3dd1034031035c6fff3ab861287
APP	contracts/protocol/sources/app/app.move	3ee919e56b81f072261b61638c0ca27feb70fedd
PRI	contracts/protocol/sources/evaluator/price.move	08bccf769fc53642d52bbb06704fa6c6df253ab3
CVA	contracts/protocol/sources/evaluator/collateral_value.move	3ba898390b6059d1c2f931b52141d93f0d8db418
VCA	contracts/protocol/sources/evaluator/value_calculator.move	7d3903f1fed3520f0b0e492c1561c079c39461b2
BWE	contracts/protocol/sources/evaluator/borrow_withdraw_evaluator.move	fb6c78f5b054fb679976e9f4c0e9f51f2681c762
LEV	contracts/protocol/sources/evaluator/liquidation_evaluator.move	3019774c1ed641c4c94d0200f2ec21191a47ab3b
DVA	contracts/protocol/sources/evaluator/debt_value.move	148c7585de9f2b33ab1d9e32058c0be6e2f9ce6f
ERR	contracts/protocol/sources/error/error.move	73bd4ba5e9f99a428620816371cb35b8f3f10997
CVE	contracts/protocol/sources/version/current_version.move	0d370c55cc395308227c4b03d423b94740b3b3f7
VER	contracts/protocol/sources/version/version.move	6004e081a1c1d113dfcfbeaf7299dc459908a295

OOB	contracts/protocol/sources/user/open_obligation.move	590603d61e73bd853b3ffb0b33889307aa7b98b7
REP	contracts/protocol/sources/user/repay.move	f1d5e15bb20969ac3081832ba46ea84247570bd2
WCO	contracts/protocol/sources/user/withdraw_collateral.move	8ec87ef66051299d4ee933eb3f89c66a96380ea3
GUV	contracts/protocol/sources/user/gusd_usdc_vault.move	20f2d7c3b050873b0678370db60908d99461df83
LOB	contracts/protocol/sources/user/lock_obligation.move	784ae8dad7d7f7db4fb983cd002c48d68bf308b8
BOR	contracts/protocol/sources/user/borrow.move	578b7391ec6c141c38ba78c2ceaceec1279d8b65
LIQ	contracts/protocol/sources/user/liquidate.move	fade8bd95fbd7b5f845ca9e37e90531371b6ed9e
AIN	contracts/protocol/sources/user/accrue_interest.move	4d340c8d66da585207aa1d8113f1cb50c8911746
FLO	contracts/protocol/sources/user/flash_loan.move	cd928d62bd6cd050bf772d68c247833d560ec905
DCO	contracts/protocol/sources/user/deposit_collateral.move	01765cdf14b4a37a670370ec52b4868a55eeaeae
XICT	contracts/xaum_indicator/core/sources/test/xaum_indicator_core_test.move	7e844e130c937b3ad45f26f82571f0ecd9a34f1f
CSXOXOSTMASMAM	contracts/sui_x_oracle/x_oracle/sources/test/mock_adapter/switchbo	d184127e51467abe8b8c410e6b65a2d767f2fa40

	ard_mock_adapter.move	
CSXOXOSTMASMAM	contracts/sui_x_oracle/x_oracle/sources/test/mock_adapter/supra_mock_adapter.move	1a42bfc37034516ce8f95becb58f0af80db4fdce
PMA	contracts/sui_x_oracle/x_oracle/sources/test/mock_adapter/pyth_mock_adapter.move	e4c5f2ff6cc52eab9e7c3776dab8bce42a40ed94
TUT	contracts/sui_x_oracle/x_oracle/sources/test/test_utils.move	47b11c7950c684fad78c0a2368d3825069efc0ed
PUT	contracts/sui_x_oracle/x_oracle/sources/test/price_update_test.move	5a66c5c36b36ee9f5b659ba19a14dedba2b07d5c
GIT	contracts/sui_x_oracle/x_oracle/sources/test/gr_indicator_test.move	007dd04c0cdc32ae5d2a95747801604aefc8abbb
ATE	contracts/sui_x_oracle/x_oracle/sources/test/advanced_test.move	9a52492659c4361ee87e2eacf392aac34c7fc4dd
PUPT	contracts/sui_x_oracle/x_oracle/sources/test/price_update_policy_test.move	1f86552ef420a73c5abe810e14c1b112c5986604
GPT	contracts/sui_x_oracle/x_oracle/sources/test/gr_pricing_test.move	fdc848d1ae26fbbb12e1f0bd33614f7bb0f907a8
SMT	contracts/protocol/sources/staking/test/staking_manager_test.move	204e143407ec4636a758288cd6c47aa29fd97909

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	22	18	4
Centralization	1	1	0
Critical	2	2	0
Major	2	2	0
Medium	8	5	3
Minor	8	8	0
Informational	1	0	1

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 Creek Team to identify any potential issues and vulnerabilities in the source code of the Creek Protocol smart contracts, 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 22 issues of varying severity, listed below.

ID	Title	Severity	Status
APP-3	TakeRevenueEvent May Not Match The Actual Transferred Amount	Medium	Fixed
APP-4	Missing Zero-address Validation for `reward_address` Before Transferring	Minor	Fixed
OBL-2	MIssing Healthy Check When Lock Obligation	Medium	Acknowledged
REP-9	Hardcoded Token Type Mismatches with Generic Parameter May Lead to Asset Loss	Critical	Fixed
RES-1	Incorrect Balance Destruction Logic Poses Risk of Asset Loss	Major	Fixed
RES-5	Flash-loan Fee Calculation Allows Zero Fee	Medium	Fixed
RES-8	Potential DoS Due to Improper Balance Splitting in the Liquidation	Major	Fixed

	function		
SMA-6	Missing divisibility check for `gr_amount` in unstake	Minor	Fixed
SMA-7	`take_stake_fee_coin` is Public Without Access Control	Critical	Fixed
APP-11	Unimplemented Auto Pause Feature	Minor	Fixed
BOR-22	Boundary Check for Minimum Borrow Amount	Minor	Fixed
DCO-20	Missing Zero-amount Checks in Fund-handling Functions	Minor	Fixed
GUV-10	Potential DoS Caused by Unreasonable Fee Rates	Medium	Fixed
GUV-23	Missing Zero-amount Checks in `redeem_gusd`	Minor	Fixed
MAR-16	Scope of Paused Flag in Market	Minor	Fixed
REP-21	Missing Debt Check Before Repayment	Minor	Fixed
RUL-19	Missing Access Control for `manual_rule` Module	Informational	Acknowledged
SMA-18	Centraliztion Risks	Centralizatio n	Fixed
XIC-13	Authorization in xaum_indicator_core update	Medium	Fixed

	functions		
XOR-12	Potential Precision Loss in `set_gr_indicators`	Medium	Fixed
XOR-15	Insufficient Validation of Price Source Independence	Medium	Acknowledged
RUL1-24	Fixed GUSD Pricing May Lead to Protocol Asset Loss	Medium	Acknowledged

3 Participant Process

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

- extend_interest_model_change_delay - Increase interest_model_change_delay field in admin_cap struct
- extend_risk_model_change_delay - Increase risk_model_change_delay field in admin_cap struct
- extend_limiter_change_delay - Increase limiter_change_delay field in admin_cap struct
- pause_protocol - Pause the protocol
- resume_protocol - Resume the protocol
- set_auto_pause_enabled - Enable auto-pause feature
- set_auto_pause_threshold - Edits threshold of auto-pause feature
- ext - For extension of the protocol
- create_interest_model_change - Creates a otw object represent interest model changes
- add_interest_model - Adds a interest model
- update_interest_model - Update interest model
- create_risk_model_change - Creates an otw object represent risk model changes
- add_risk_model - Adds a risk model
- update_risk_model - Update risk model
- add_limiter - Adds a limiter
- create_limiter_params_change - Creates an otw object represent limiter params changes
- create_limiter_limit_change - Creates an otw object represent limiter limit changes

- apply_limiter_limit_change - Applys the changes generated from create_limiter_limit_change
- apply_limiter_params_change - Applys the changes generated from create_limiter_params_change
- set_base_asset_active_state - Manage base asset state
- set_collateral_active_state - Manage collateral asset state
- take_revenue - Transfer accumulated revenue from protocol to admin
- take_borrow_fee - Transfer accumulated borrow fee from protocol to admin
- update_staking_fee - Update XAUM staking fee
- update_unstaking_fee - Update XAUM unstaking fee
- take_staking_fee - Transfer accumulated staking fee from protocol to admin
- add_lock_key - Add lock key in obligation_access
- remove_lock_key - Remove lock key in obligation_access
- add_reward_key - Add reward key in obligation_access
- remove_reward_key - Remove reward key in obligation_access
- update_borrow_fee - Update borrow fee factor
- update_borrow_limit - Upteda upper limit of borrow amount
- set_gusd_cap - Set mint cap of GUSD
- update_reward_address - Update reward_address field in AdminCap struct
- transfer_admin_cap - Transfer AdminCap to a new address
- set_flash_loan_single_cap - Set upper limit of flash-loan
- take_stake_fee - Take staking XAUM fee from the fee pool by admin
- owner_withdraw_xaum - Withdraw XAUM from staking pool to owner wallet

- owner_deposit_xaum - Deposit XAUM from owner wallet into staking pool

User

- lock - Lock the obligation owned by user
- unlock - Unlock the obligation owned by user
- stake_xaum - Stakes XAUM in exchange for GR and GY at the fixed exchange rate
- unstake - Unstakes by burning GR and GY to redeem XAUM.
- borrow - Borrow a certain amount of GUSD from the protocol
- deposit_collateral - Deposit collateral into the given obligation
- borrow_flash_loan - Flash-loan GUSD from protocol
- mint_gusd - Accept USDC and mint same amount of GUSD
- redeem_gusd - Burn GUSD to receive USDC, deducting 0.3% fee
- liquidate - Liquidate an unhealthy obligation
- force_unlock_unhealthy - Unlock the an unhealthy obligation
- open_obligation - Create a new obligation and transfer the obligation key to the sender
- repay - Repay the debt of the obligation
- withdraw_collateral - Withdraw collateral from obligation, and return the collateral

4 Findings

APP-3 TakeRevenueEvent May Not Match The Actual Transferred Amount

Severity: Medium

Status: Fixed

Code Location:

contracts/protocol/sources/app/app.move#374

Descriptions:

In the `take_revenue` logic, there is an inconsistency between the emitted event amount and the actual amount of tokens transferred.

Specifically, in the inner function:

```
public(package) fun take_revenue<T>(
    self: &mut Reserve,
    amount: u64,
    ctx: &mut TxContext
): Coin<T> {
    let balance_sheet = wit_table::borrow_mut(BalanceSheets {}, &mut
self.balance_sheets, get<T>());
    let all_revenue = balance_sheet.revenue;
    let take_amount = math::min(amount, all_revenue);

    balance_sheet.revenue = balance_sheet.revenue - take_amount;
    let balance = balance_bag::split<T>(&mut self.underlying_balances, take_amount);
    coin::from_balance(balance, ctx)
}
```

Because of `let take_amount = math::min(amount, all_revenue);` , the actual amount withdrawn (take_amount) can be smaller than the requested amount when the available revenue is insufficient.

However, in the higher-level function:

```
event::emit(TakeRevenueEvent {
    market: object::id(market),
    amount,
    coin_type: type_name::get<T>(),
    sender: tx_context::sender(ctx),
    recipient: admin_cap.reward_address,
});
```

the event uses the original amount instead of the actual take_amount.

Suggestion:

Emit the event after determining the actual withdrawn amount.

Resolution:

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

APP-4 Missing Zero-address Validation for `reward_address` Before Transferring

Severity: Minor

Status: Fixed

Code Location:

contracts/protocol/sources/app/app.move#381,400,441

Descriptions:

In several admin withdrawal functions, including:

- take_revenue
- take_borrow_fee
- take_staking_fee

the code transfers fee/revenue tokens directly to admin_cap.reward_address without verifying that it is a valid (non-zero) address:

```
transfer::public_transfer(coin, admin_cap.reward_address);
```

However, during initialization (init_internal), reward_address defaults to @0x0. If this value is not explicitly updated, any subsequent fee/revenue withdrawal will send tokens to the zero address.

Suggestion:

Add a zero-address validation check before performing the transfer.

Resolution:

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

OBL-2 MIssing Healthy Check When Lock Obligation

Severity: Medium

Status: Acknowledged

Code Location:

contracts/protocol/sources/obligation/obligation.move

Descriptions:

Creek allows user to set lock on their own obligation via calling `lock` function:

```
public fun lock<T: drop>(
    self: &mut Obligation,
    obligation_key: &ObligationKey,
    obligation_access_store: &ObligationAccessStore,
    lock_borrow: bool,
    lock_repay: bool,
    lock_deposit_collateral: bool,
    lock_withdraw_collateral: bool,
    lock_liquidate: bool,
    key: T,
) {
    assert_key_match(self, obligation_key);

    set_lock<T>(
        self,
        obligation_access_store,
        lock_borrow,
        lock_repay,
        lock_deposit_collateral,
        lock_withdraw_collateral,
        lock_liquidate,
        key,
    );

    emit(ObligationLocked {
        obligation: object::id(self),
```

```
        witness: type_name::get<T>(),
        borrow_locked: self.borrow_locked,
        repay_locked: self.repay_locked,
        withdraw_collateral_locked: self.withdraw_collateral_locked,
        deposit_collateral_locked: self.deposit_collateral_locked,
        liquidate_locked: self.liquidate_locked,
    });
}
```

However, this function did not check the obligation is healthy or not, user can lock their liquidatable obligation at no cost. Even there were a function to forcely unlock unhealthy obligation, but still malicious user can race this process by continously lock their unhealthy obligation.

Suggestion:

It is recommended to add extra check for obligation health in `lock` function.

Resolution:

The team states that this is by design and wont affect protocol functioning.

REP-9 Hardcoded Token Type Mismatches with Generic Parameter May Lead to Asset Loss

Severity: Critical

Status: Fixed

Code Location:

contracts/protocol/sources/user/repay.move

Descriptions:

The function repay is defined as a generic function `repay<T>`, intended to handle repayments for any type of asset. However, the type of its parameter `user_coin` is hardcoded as `Coin<COIN_GUSD>`, and the call to `coin::split` within the function body also uses the hardcoded `COIN_GUSD` type. This creates a severe conflict between the declared generic capability of the function and its actual implementation. When attempting to call this function with any asset type `T` other than `COIN_GUSD`, `type_name::get<T>()` correctly retrieves the type of `T` for debt queries, but subsequently processes `Coin` objects of type `COIN_GUSD`. This will potentialy cause asset loss due to type mismatches when calling `market::handle_repay<T>`, rendering the function unusable for multiple assets as intended.

contracts/protocol/sources/user/repay.move

```
    let coin_type = type_name::get<T>();// <= coin_type as T

    // always accrued all the interest before doing any actions
    // Because all actions should based on the latest state
    market::accrue_all_interests(market, now);
    obligation::accrue_interests_and_rewards(obligation, market);

    // If the given coin is more than the debt, repay the debt only
    let (debt_amount, _) = obligation::debt(obligation, coin_type);//<= T debt
    let repay_amount = math::min(debt_amount, coin::value(&user_coin));
    let repay_coin = coin::split<COIN_GUSD>(&mut user_coin, repay_amount, ctx);//<=
repay coins as COIN_GUSD
```

```
// Put the repay asset into market
market::handle_repay<T>(market, repay_coin, ctx);

// Decrease repay amount to the outflow limiter
market::handle_inflow<T>(market, repay_amount, now);

// Decrease debt of the obligation according to repay amount
obligation::decrease_debt(obligation, coin_type, repay_amount);//<= decrease T debt
```

Suggestion:

It is recommended to verify that type `T` matches `COIN_GUSD` within the function.

Resolution:

The team adopted our advice and fixed this issue by force using type `COIN_GUSD` instead of using generic tyoe `T`, this update can be found at commit f4bbf44d3e44e77894ad6549f4abdf5d9b3fd212.

RES-1 Incorrect Balance Destruction Logic Poses Risk of Asset Loss

Severity: Major

Status: Fixed

Code Location:

contracts/protocol/sources/market/reserve.move

Descriptions:

In the else branch of the liquidation function, the code unconditionally calls `balance::destroy_zero(repay_balance)` and `balance::destroy_zero(revenue_balance)`, but these balances are not guaranteed to be zero at this point.

```
// insert interest to reserve's underlying balances
    if (interest_to_keep > 0) {
        balance_bag::join(&mut self.underlying_balances, repay_balance);
        balance_bag::join(&mut self.underlying_balances, revenue_balance);
    } else {
        balance::destroy_zero(repay_balance);
        balance::destroy_zero(revenue_balance);
    };
```

Particularly, `revenue_balance` were calculated in previous step in liquidation process, which was actually a portion of liquidator's repay amount, this portion of repay coins is not altered when incoking `reserve.handle_liquidation` function.

```
...
let actual_repay_revenue = fixed_point32::multiply_u64(actual_repay_amount,
liq_revenue_factor);
    // actual_replay_on_behalf is the amount that is repaid on behalf of the borrower,
which should be deducted from the borrower's obligation
    let actual_replay_on_behalf = actual_repay_amount - actual_repay_revenue;
...
// handle liquidation in market & reserve
```

```
    let repay_on_behalf_balance = balance::split(&mut available_repay_balance,
repay_on_behalf);
    let revenue_balance = balance::split(&mut available_repay_balance, repay_revenue);
```

This erroneous destruction logic will cause panic ehen destroying non-zero value balance, resulting in DoS of the protocol.

Suggestion:

It's recommended to add balance check assertions to ensure only zero balances are destroyed, while non-zero balances should be merged into appropriate reserve pools or protocol revenue.

Resolution:

This issue has been fixed. The client has adopted our suggestions, which can be found at commit "952c63d290b59510364c746808311209fe950870".

RES-5 Flash-loan Fee Calculation Allows Zero Fee

Severity: Medium

Status: Fixed

Code Location:

contracts/protocol/sources/market/reserve.move#250

Descriptions:

In the borrow_flash_loan function, the fee is computed using floor division:

```
public(package) fun borrow_flash_loan(_self: &mut Reserve, amount: u64):
FlashLoan<COIN_GUSD> {
    let fee = u64::mul_div(amount, FLASH_LOAN_FEE_NUM, FLASH_LOAN_FEE_DEN);
    FlashLoan<COIN_GUSD> { loan_amount: amount, fee }
}
```

Since u64::mul_div performs integer floor division, the result truncates toward zero. As a result, when amount is small enough (e.g., amount = 999), the computed fee becomes 0.

Suggestion:

Add a check to ensure flash loan pays a positive fee.

Resolution:

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

RES-8 Potential DoS Due to Improper Balance Splitting in the Liquidation function

Severity: Major

Status: Fixed

Code Location:

contracts/protocol/sources/market/reserve.move

Descriptions:

In `handle_liquidation<T>` function, the code calculates the debt amount to be burned (`debt_to_burn`) based on the sum of `repay_balance` and `revenue_balance` . It then attempts to split only the `repay_balance` to extract the `debt_to_burn` amount. If the `debt_to_burn` exceeds the value of `repay_balance` (for example, when `revenue_balance` has a non-zero value), the `balance::split` operation will fail and cause a transaction panic. This renders the liquidation mechanism unusable under many normal conditions and may freeze funds involved in the liquidation.

contracts/protocol/sources/market/reserve.move

```
    let total_amount = balance::value(&repay_balance) +
balance::value(&revenue_balance);//<= here total_amount >= repay_balance

    // calculate how much debt and interest to keep
    let debt_to_burn = if (balance_sheet.debt >= total_amount) { total_amount } else {
        balance_sheet.debt
    };//<= debt_to_burn may equals to total_amount
    let interest_to_keep = total_amount - debt_to_burn;

    // update balance sheet
    balance_sheet.debt = balance_sheet.debt - debt_to_burn;
    if (interest_to_keep > 0) {
        balance_sheet.revenue = balance_sheet.revenue + interest_to_keep;
    };
```

```
    // split the repay_balance into principal part and interest part
    let principal_balance = balance::split(&mut repay_balance, debt_to_burn);//<=
repay_balance may less than  debt_to_burn
```

Suggestion:

It is recommended to merge `revenue_balance` into `repay_balance` before performing the split operation.

Resolution:

The team adopted our advice and fixed this issue by merge `revenue_balance` into `repay_balance` before performing the split operation, which can be found at commit 287f11b203fbabd7fd3b80c0c931cb9938889b86.

SMA-6 Missing divisibility check for `gr_amount` in unstake

Severity: Minor

Status: Fixed

Code Location:

contracts/protocol/sources/staking/staking_manager.move#154

Descriptions:

In the `unstake` function:

```
let xaum_return_amount = gr_amount / EXCHANGE_RATE;
```

If `gr_amount % EXCHANGE_RATE != 0`, this division will truncate the remainder, causing:

Slight mismatch between the actual burned GR/GY and redeemed XAUM,

Potential rounding inconsistencies in accounting or event reporting.

Suggestion:

Add an assertion to ensure divisibility before performing the division.

Resolution:

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

SMA-7 `take_stake_fee_coin` is Public Without Access Control

Severity: Critical

Status: Fixed

Code Location:

contracts/protocol/sources/staking/staking_manager.move#262

Descriptions:

The function `take_stake_fee_coin` is declared as public and allows direct withdrawal of XAUM tokens from the fee pool:

```
public fun take_stake_fee_coin(
    manager: &mut StakingManager,
    amount: u64,
    ctx: &mut TxContext,
): Coin<COIN_XAUM> {
    let fee_part = balance::split(&mut manager.xaum_fee_pool, amount);
    coin::from_balance(fee_part, ctx)
}
```

According to the comment, this helper is intended to be called only by protocol::app. However, because it is marked as public, any external module can call it directly to withdraw from manager.xaum_fee_pool without authorization.

Suggestion:

Restrict access by changing visibility, alternatively, add an explicit admin check.

Resolution:

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

APP-11 Unimplemented Auto Pause Feature

Severity: Minor

Status: Fixed

Code Location:

contracts/protocol/sources/app/app.move

Descriptions:

The `Market` struct defines the fields `auto_pause_enabled` and `auto_pause_threshold`, indicating that the protocol was designed with an automatic pause mechanism to handle extreme market conditions. However, no logic exists anywhere in the module's code to check whether the conditions defined by `auto_pause_threshold` have been triggered, nor is `set_paused(self, true)` automatically called when those conditions are met. This results in the complete absence of this functionality, giving users a false sense of security that the protocol has automatic risk controls in place, when in reality, this mechanism is not operational.

Suggestion:

It is recommended to implemented auto-pause feature or remove related code.

Resolution:

The team adopted our advice and fixed this issue by implemented auto-pause feature in `check_and_pause_if_gusd_depeg` function, this update can be found at commit d1023054812aad8eaaa94ad602c23cbe9abdb753.

BOR-22 Boundary Check for Minimum Borrow Amount

Severity: Minor

Status: Fixed

Code Location:

contracts/protocol/sources/user/borrow.move#144

Descriptions:

In `borrow_internal`, the function checks:

```
assert!(
    borrow_amount > min_borrow_amount + base_borrow_fee_amount,
    error::borrow_too_small_error(),
);
```

This condition excludes the valid edge case where the borrow amount equals the minimum allowed value plus the fee.

Suggestion:

It is recommended to use

```
assert!(
    borrow_amount >= min_borrow_amount + base_borrow_fee_amount,
    error::borrow_too_small_error(),
);
```

to correctly allow the minimum valid borrow amount.

Resolution:

This issue has been fixed. The client has adopted our suggestions. The updates can be found at commit 838798922d29d2ff83e54ae220f9adc0bcf3d6e3.

DCO-20 Missing Zero-amount Checks in Fund-handling Functions

Severity: Minor

Status: Fixed

Code Location:

contracts/protocol/sources/user/deposit_collateral.move#32

Descriptions:

Several functions related to fund operations—such as `deposit_collateral`, `withdraw_collateral`, `borrow_flash_loan`, `repay_flash_loan`, `liquidate` accept an amount parameter as input but do not verify that the value is greater than zero.

Suggestion:

Add validation at the start, ensures that zero-value operations are rejected.

Resolution:

The client has adopted our advice fixed the issue. The updates can be found at commit 5d48a98198ebec4dc5380364313c0b6742ab9c1b.

GUV-10 Potential DoS Caused by Unreasonable Fee Rates

Severity: Medium

Status: Fixed

Code Location:

contracts/protocol/sources/user/gusd_usdc_vault.move

Descriptions:

In the `update_fee_rate` function, the administrator can set new redemption fee rates. However, this function lacks an upper limit check for `new_fee_rate`. If the administrator sets a `new_fee_rate` greater than or equal to 10000 (representing a 100% fee), the calculated fee in the `redeem_gusd` function will always be greater than or equal to the user's redemption amount. This will cause the assertion `assert!(fee < amount, E_INVALID_FEE)` in the `redeem_gusd` function to fail, making all redemption operations unsuccessful. This effectively blocks the core functionality of users redeeming GUSD for USDC and may result in trapped user funds.

Suggestion:

It is recommended to add a check in the `update_fee_rate` function to ensure the new fee rate remains within a reasonable range and is strictly less than the fee base value of 10000.

Resolution:

The team adopted our advice and fixed this issue by add a check in the `update_fee_rate` function to ensure the new fee rate remains within a reasonable range, which can be found at commit dae94882cd91bb25a344c4ec2c9dcae410c457fa.

GUV-23 Missing Zero-amount Checks in `redeem_gusd`

Severity: Minor

Status: Fixed

Code Location:

contracts/protocol/sources/user/gusd_usdc_vault.move#135

Descriptions:

In `redeem_gusd`, the GUSD amount is converted to USDC:

```
// convert precision: GUSD(9)    USDC(6)
let amount_usdc_before_fee = amount_gusd / 1000;
```

Currently, only `amount_gusd > 0` is checked.

For small `amount_gusd` (<1000), `amount_usdc_before_fee` becomes 0, but there is no check to prevent zero-value USDC.

Suggestion:

It is suggested to add a zero amount check after conversion.

Resolution:

The client has adopted our suggestions and fixed this issue by adding a zero amount check after conversion, which can be found at commit 29115dcc9d45104e27785bc94814c65b64074277.

MAR-16 Scope of Paused Flag in Market

Severity: Minor

Status: Fixed

Code Location:

contracts/protocol/sources/market/market.move#32

Descriptions:

In the current implementation of Market, The paused flag currently affects the following operations:

- borrow
- deposit_collateral
- borrow_flash_loan
- repay
- withdraw_collateral

it does not affect:

- mint_gusd
- redeem_gusd

Is this the intended design?

Suggestion:

- Confirm whether the current scope of paused is intentional.
- add a paused check to `repay_flash_loan`, mirroring the logic in `borrow_flash_loan`, even though the hot potato design ensures atomicity.

Resolution:

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

REP-21 Missing Debt Check Before Repayment

Severity: Minor

Status: Fixed

Code Location:

contracts/protocol/sources/user/repay.move#63

Descriptions:

The `repay` function does not verify that the user has a positive outstanding debt before proceeding.

Suggestion:

Add a check after fetching debt_amount to ensure `debt_amount > 0` before continuing.

Resolution:

This issue has been fixed. The client has adopted our suggestions. The updates can be found at commit bb6369baecd2175b60ec2c4355063ca10e9f9079.

RUL-19 Missing Access Control for `manual_rule` Module

Severity: Informational

Status: Acknowledged

Code Location:

contracts/sui_x_oracle/manual_rule/sources/rule.move

Descriptions:

In the Creek contract oracle system, there exists a manual oracle that allows for manual price setting of specific tokens. However, this oracle lacks access control, enabling any user to arbitrarily set token prices. Based on our analysis of the protocol's implementation, we suspect that this oracle may have been introduced solely for testing purposes, though we cannot confirm this with certainty.

Suggestion:

We request the project team to clarify whether this oracle is only used in test environments and whether it will be included in the protocol's oracle whitelist in the production environment.

Resolution:

The team claims that related code is only for testing purpose, they will delete related code at future.

SMA-18 Centraliztion Risks

Severity: Centralization

Status: Fixed

Code Location:

contracts/protocol/sources/staking/staking_manager.move;

contracts/protocol/sources/user/gusd_usdc_vault.move;

contracts/protocol/sources/market/interest_model.move

Descriptions:

In this protocol, several centralization risks have been identified:

1. In the `staking_manager` contract, the owner can withdraw all XAUM tokens via the `owner_withdraw_xaum` function without any restrictions.
2. The `gusd_usdc_vault` and `interest_model` contracts lack necessary validation checks when setting the `fee_rate`.
3. Functions such as `update_team_address` and `update_admin` in the `gusd_usdc_vault` contract do not verify the validity of the input addresses.

Suggestion:

It is recommended to :

1. Consider implementing a timelock, multi-signature mechanism, or withdrawal limits for the `owner_withdraw_xaum` function to mitigate single-point control risks.
2. Introduce boundary checks (e.g., setting a maximum fee rate) and access controls to ensure the `fee_rate` can only be updated securely and reasonably.
3. Consdering use transfer-accept mechenism to prevent admin address transfered to invalid address.

Resolution:

The team fixed these issues by:

1. Claims they are using multi-sig wallet to manage the protocol.
2. Introduce neccessary boundary checks on crucial state variables.
3. Implemented transfer-accept mechnism to prevent admin address transfered to invalid address.

XIC-13 Authorization in xaum_indicator_core update functions

Severity: Medium

Status: Fixed

Code Location:

contracts/xaum_indicator/core/sources/xaum_indicator_core.move#1

Descriptions:

The `xaum_indicator_core` module exposes several update functions — including `set_price_9dec`, `update_price_storage_external`, and `init_ema_values` — that can be invoked publicly without any access control.

The comment (Manual price setter (9-decimals) for local/general usage) suggests they may be designed for local testing rather than production use.

Suggestion:

It would be helpful to clarify whether these update functions are meant only for testing or will also be deployed in production.

Resolution:

This issue has been fixed. The client has adopted our suggestions. The updated code can be found at commit 35aa44e4ffeb0b0ca3c97988f62b431f1f057cbe.

XOR-12 Potential Precision Loss in `set_gr_indicators`

Severity: Medium

Status: Fixed

Code Location:

contracts/sui_x_oracle/x_oracle/sources/x_oracle.move

Descriptions:

In the `set_gr_indicators` function, the formula for calculating `s_value` — `sValue = α × EMA120 + (1-α) × [β × EMA90 + (1-β) × SpotPrice]` — is implemented using integer arithmetic with intermediate division steps. Specifically, the lines of code `let inner = inner_num / scale;` and `let prelim: u128 = s_num / scale;` perform division operations before the final calculation is complete. Each division operation discards the remainder, leading to a loss of precision. This accumulated precision error can cause inaccuracies in the final calculated oracle price, thereby compromising the correctness of protocols that rely on this price.

Suggestion:

To minimize precision loss, the calculation process should be restructured so that all multiplication operations are performed first, followed by a single division operation at the end.

Resolution:

The team adopted our advice and fixed this issue by put division operation at the end of calculation process, this update can be found at commit 79913d37ff092bfc7fb022ca152d208ef1f43113.

XOR-15 Insufficient Validation of Price Source Independence

Severity: Medium

Status: Acknowledged

Code Location:

contracts/sui_x_oracle/x_oracle/sources/x_oracle.move#350

Descriptions:

In the `determine_price function`, the code calculates the required number of secondary matches using:

```
let required_secondary_match_num = (secondary_price_feed_num + 1) / 2;
```

However, there is no check to ensure that `required_secondary_match_num > 0`. Additionally, in the upper-level function `confirm_price_update_request`, there is no validation to ensure that the primary_price_update_request and secondary_price_update_request originate from distinct sources.

Suggestion:

- Add an explicit check to ensure at least one secondary feed exists.
- validate that primary and secondary price feeds are sourced from independent providers before invoking `determine_price`.

Resolution:

The client has acknowledged the issue. Currently, XAUM on Sui relies solely on the Pyth oracle.

RUL1-24 Fixed GUSD Pricing May Lead to Protocol Asset Loss

Severity: Medium

Status: Acknowledged

Code Location:

contracts/sui_x_oracle/gusd_rule/sources/rule.move

Descriptions:

Pegging `GUSD` to a fixed price of $1 in the protocol could result in price misalignment. For example, when the external oracle shows fluctuations in the token's market price, the internal price remains constant at 1. This discrepancy could trigger the following two scenarios:

- GUSD Market Price Below $1: Arbitrageurs could purchase `GUSD` at a lower price on external markets and use it within the protocol to repay debt at the fixed $1 price. This allows them to clear debt at below face value and release collateral. Consequently, the protocol loses collateral while the repaid debt is worth less in market terms, leading to a net loss for the protocol.
- GUSD Market Price Above $1: Users could borrow `GUSD` from the protocol at $1 and sell it at a higher market price externally, profiting from the spread. This would increase the protocol's debt burden without a corresponding increase in collateral value, thereby expanding the protocol's risk exposure.

Suggestion:

It is recommended to use the stablecoin's actual market price for internal calculations to ensure that debt valuation aligns with real time market conditions.

Resolution:

The team acknowledged this issue and state that "Before launch, there was no GUSD-USDC liquidity pool, so no oracle price was available. After launch, once the oracle price becomes

operational, the GUSD price can be updated from the fixed value of 1 to the oracle-provided price by modifying the oracle rule", so no modifications will be made to the current version.

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.