Bluefin Spot Smart Contract Audit

audited by Asymptotic

Summary

Asymptotic reviewed Bluefin's spot smart contract code. We only identified low-severity issues and have advisory recommendations. The low-severity issues primarily involved minor overflow handling, missing validations, duplicate code, and error handling. The advisory recommendations suggested improvements to code clarity, maintainability, and logical simplifications.

Formal verification confirmed correctness in liquidity management and swap functionalities, verifying the security and accuracy of core operations.

The development team has effectively addressed all identified issues and recommendations.

Legal Disclaimer

Scope and Limitations

This security audit report focuses specifically on reviewing the Move smart contract code of Bluefin Spot smart contract. The audit does not analyze or make any claims about other components of the system, including but not limited to:

- Frontend applications and user interfaces
- Backend services and infrastructure
- Off-chain components and integrations
- Deployment procedures and operational security
- Third-party dependencies and external services

The findings and recommendations in this report are limited to the Move code implementation and its immediate interactions within the Sui blockchain environment.

Creating proofs for specifications is on a "best effort" basis. We manually audited the code in instances where formal proofs were not technically feasible.

Audited code

Audited: 2fdbe99
Commit integrating all remediations: 7957dac

Legend

Issue severity

- **Critical** — Vulnerabilities which allow account takeovers or stealing significant funds, along with being easy to exploit.
- **High** — Vulnerabilities which either can have significant impact but are hard to exploit, or are easy to exploit but have more limited impact.
- **Medium** — Moderate risks with notable but limited impact.
- **Low** — Minor issues with minimal security implications.
- **Advisory** — Informational findings for security/code improvements.

L-1: Observation cardinality comment inconsistency

Severity: Low

Description

The grow (oracle.move) function contains contradictory logic between its comment and implementation. The comment states that ”cardinality can only be increased, not decreased”, but the code uses the min, function which could potentially decrease the value.

Three possible scenarios:

1. When observation_cardinality_next is less than new_cardinality, the current_cardinality will equal new_cardinality after the loop, making the min operation redundant
2. When values are equal, the operation is unnecessary
3. When observation_cardinality_next is larger than new_cardinality, the value would decrease, contradicting the comment

Recommendation

Ensure the comment describes the intended behavior and, if so, replace min() with max() to match the comment.

Remediation

✓ **Commit:** 7957dac

L-2: Incorrect overflow_add

Severity: Low

Description

The overflow_add function contains incorrect arithmetic logic that leads to unexpected behavior during overflow handling. The current implementation underflows during the first subtraction: num2 - constants::max_u256() - num1 - 1

The risk is low because the only use of the function is in a context where overflow is unlikely to occur. However, we recommend fixing, just in case the function will be used in different contexts in the future.

Recommendation

Modify the arithmetic logic: num2 - (constants::max_u256() - num1) - 1

Remediation

✓ **Commit:** 5ffee3e

L-3: Public Pool functions lack slippage protection

Severity: Low

Description

The public liquidity functions (add_liquidity_with_fixed_amount, add_liquidity, remove_liquidity) and flash_swap function rely on the gateway module for slippage protection checks. Since these functions are also public themselves, they can be called directly from other modules, bypassing the gateway's slippage checks, which creates a potential security risk.

Recommendation

We recommend one of the following solutions:

a. Move slippage protection checks directly into the pool module functions

b. Restrict access to pool functions by changing their visibility to public(package)

c. Document that slippage protection is intentionally omitted from pool functions and must be implemented by the calling module if needed

Remediation

✓ **Acknowledged**

L-4: flash_swap should not use the pause-pool functionality

Severity: Low

Description

The flash_swap function uses the pool's pause flag to prevent other operations until repayment is received. This creates a risk since administrators also control the pause functionality—an admin could potentially unpause the pool during an active flash swap they initiated, allowing other operations before repayment that would corrupt the pool's state.

Recommendation

Introduce a dedicated flash_swap_in_progress flag to track flash swap state instead of reusing the pause functionality. This ensures flash swap state cannot be manipulated by admin pause/unpause operations.

Remediation

✓ **Commit:** bc0814f

L-5: Unsafe vector index validation in get_observation

Severity: Low

Description

The current get_observation implementation uses an unnecessary subtraction operation (length - 1) when checking vector bounds. This approach is unsafe as subtraction may underflow in case of empty vector. In the current implementation, during creation the first observation element is inserted into manager.observations. However, it is recommended to use a safer and more optimized approach instead

if (index ¿ vector::length(&manager.observations) - 1)

Recommendation

Replace the current implementation with a simpler bounds check that directly compares the index against the vector length. The suggested implementation is:

```
if (index < vector::length(&manager.observations)) {
    *vector::borrow(&manager.observations, index)
} else {
    default_observation()
}
```

This change will:

- eliminate unnecessary arithmetic operations
- make the code safer by avoiding potential underflow
- improve code readability

Remediation

✓ **Commit:** 35fffd8

L-6: Error code used twice

Severity: **Low**

Description

Error code 1029 is used for both EZeroAmount and EVersionMax

Remediation

✓ **Commit:** 6bfef85

L-7: close_position_v2 should not take a pool param

Severity: Low

Description

In bluefin_spot::pool, close_position_v2 doesn't need the mutable pool parameter. The pool param is not necessary and there is currently no check that the position refers to the right pool id. The risk is that it will start being used in the future without also adding the check.

Remediation

✓ **Remediated**

L-8: Code duplication: swap_in_pool and calculate_swap_results

Severity: Low

Description

swap_in_pool and calculate_swap_results are both very large functions implementing almost identical functionality, with calculate_swap_results just not applying the effects. There is a rick for the function to drift apart in the future. We recommend deduplicating, either by extracting the common parts or, simpler, just passing in a flag which decides whether effects take place.

There is already a small, inconsequential drift: the first two asserts use ¡= / ¿= in calculate_swap_results and ¡ / ¿ in swap_in_pool.

Remediation

✓ **Acknowledged**

A-1: Multi-sig for admin cap

Severity: Advisory

Description

The system implements two privileged capability types, AdminCap and ProtocolFeeCap, that grant significant administrative control. The AdminCap allows functions like version updates, fee modifications, pausing pools, and reward management, while the ProtocolFeeCap enables claiming protocol and pool creation fees. While these administrative functions are necessary and well-designed, they represent a centralization of control that could pose risks if compromised.

Recommendation

Implement decentralization measures to reduce centralization risks:

- If you are not already, use `https://docs.sui.io/concepts/cryptography/transaction-auth/multisig` for the admin cap
- Consider reducing administrative functionality where feasible
- Consider adding time-locks for sensitive operations

Remediation

✓ **Remediated**

A-2: Check for non-zero collectible fees

Severity: Advisory

Description

The collect_fee function lacks a validation check to verify if there is an actual fee amount greater than zero to collect from the pool. This could lead to unnecessary gas consumption and transaction execution for zero-amount fee collections.

Recommendation

Add an explicit validation check to verify that the fee amount to be collected is greater than zero.

Remediation

✓ **Acknowledged**

A-3: Simplify boolean logic

Severity: Advisory

Description

The code contains unnecessarily verbose boolean expressions and control flow statements that can be simplified. Specific instances are:

```
swap_result.is_exceed =
    if (swap_result.amount_specified_remaining > 0) { true } else { false };
```

```
    let pool_manager = field::borrow<String, address>(&pool.id, constants::manager());
    if (*pool_manager == manager) {
        return true
    };
    return false
```

Recommendation

Implement the following optimizations:

```
swap_result.is_exceed = swap_result.amount_specified_remaining > 0;

return manager == *field::borrow<String, address>(&pool.id, constants::manager())
```

Remediation

✓ **Commit:** 1107a62

A-4: Unnecessary continue statement

Severity: Advisory

Description

pool.move contains a continue statement as the last expression in a while loop. This is redundant since the loop will naturally continue to the next iteration after reaching the end of the loop body.

Recommendation

Remove the unnecessary continue statement to improve code readability and eliminate redundant code.

Remediation

✓ **Commit:** e3c2dcb

A-5: Unused struct fields

Severity: Advisory

Description

Several struct fields in the codebase are not used, which increases code complexity:

- Position.fee_rate is unused and not publicly accessible. This field would be more logically placed in the pool struct if needed.
- TickInfo.sqrt_price is not used anywhere in the codebase.

Remediation

✓ **Acknowledged**

A-6: Unused error constants

Severity: Advisory

Description

Several error constants and corresponding functions are defined but never used in the codebase:

- insufficient_pool_balance
- tick_score_out_of_bounds
- swap_amount_exceeds

Recommendation

Review code to either use the error constants or remove them.

Remediation

✓ **Commit:** 3e1b23b

A-7: reward_managers can be a VecSet

Severity: Advisory

Description

In bluefin_spot::config, reward_managers can be a VecSet , as current functionality essentially implements it:

- in remove_reward_manager, the while can be VecSet::remove
- verify_reward_manager can be contains

We do realize this might be a more complex upgrade, so ok to postpone implementing it.

Remediation

✓ **Acknowledged**

About the Auditor

Asymptotic provides a white-glove, formal-verification-based auditing service for Sui smart contracts.

- Analyze customer codebases to create mathematical proofs of security
- Use AI to accelerate specification writing, proof construction, and adjusting specs as code changes
- Focus on real-world security properties that matter for production systems

Asymptotic Team

Andrei Stefanescu

- Led verification of AWS cryptographic algorithms using SAW at Galois
- Created first Ethereum smart contract verification tool
- Three-time International Math Olympiad silver medalist
- PhD in Computer Science from UIUC with David J. Kuck Outstanding Thesis Award
- ACM SIGPLAN Distinguished Paper Award at OOPSLA 2016
- Pioneered K Framework for programming language semantics and verification

Cosmin Radoi

- Created GritQL, a language for large-scale code migration (backed by Founders Fund, 8VC)
- PhD in Computer Science from UIUC focusing on programming languages
- Multiple ACM SIGSOFT Distinguished Paper Awards at top conferences (ICSE, OOPSLA, FSE)
- IBM PhD Fellowship recipient and NSF SBIR Award winner
- Key contributor to K Framework for language semantics and verification
- Research expertise in parallelism, race detection, and performance optimization