Security Audit Report

Project: Swaylend
Auditor: SafeEdges
Date: July 2025
Version: 1.0

Executive Summary

This security audit report presents the findings from a comprehensive security assessment of the Swaylend lending protocol. The audit identified critical vulnerabilities in oracle management and price feed validation that could lead to fund losses and market manipulation. All identified issues, including a recently reported low-severity issue in collateral value calculation, have been remediated by the development team.

Scope

The audit focused on the oracle integration and price feed management functionality within the Swaylend lending market contract, specifically examining:

- Pyth oracle integration
- Price feed update mechanisms
- Oracle data validation
- Fund management for oracle operations
- Decimal scaling logic in collateral and borrowing calculations

Methodology

The security assessment employed a combination of manual code review and automated analysis techniques to identify potential vulnerabilities. The evaluation followed industry-standard security practices and considered common attack vectors in DeFi protocols.

Findings

1. Missing Funds Validation in `update_price_feeds`

Severity: High
Status: SOLVED

Description

The `update_price_feeds` function responsible for updating Pyth oracles failed to validate that the funds sent by users matched the required `update_fee` parameter. This vulnerability allowed attackers to update price feeds using the market's internal balance instead of their own funds, resulting in direct fund loss for depositors.

Technical Details

```
#[storage(read, write), payable]
fn update_price_feeds(update_data: Vec<Bytes>) {
    require(
        msg_asset_id() == AssetId::base(),
        PythError::FeesCanOnlyBePaidInTheBaseAsset,
    );

    let required_fee = total_fee(total_number_of_updates,
storage.single_update_fee);
    require(msg_amount() >= required_fee, PythError::InsufficientFee);
}
```

```
#[payable, storage(read)]
fn update_price_feeds_internal(update_fee: u64, update_data: Vec<Bytes>) {
    let contract_id = storage.pyth_contract_id.read();
    require(contract_id != ZERO_B256, Error::OracleContractIdNotSet);

    let oracle = abi(PythCore, contract_id);
    oracle.update_price_feeds {
        asset_id: FUEL_ETH_BASE_ASSET_ID, coins: update_fee
    }(update_data);
}
```

Impact

- Direct fund loss from market reserves
- Exploitation by malicious actors to drain protocol funds
- Potential for repeated attacks until market balance depletion

Risk Score

BVSS: AO:A/AC:L/AX:M/C:N/I:N/A:N/D:C/Y:N/R:N/S:C (8.4)

Remediation

Proper validation was added to ensure the user-sent amount matches or exceeds `update_fee` and the asset used is correct.

Fix Commit: 622bd073b48cfc211e8b7d6d0474186c61903e6f

---

2. Missing Staleness Checks in Oracle Queries

Severity: Medium
Status: SOLVED

Description

The protocol failed to implement staleness checks when querying Pyth price feeds, allowing the use of outdated price data that may not reflect current market conditions.

Technical Details

```
pub struct Price {
    pub confidence: u64,
    pub exponent: u32,
    pub price: u64,
    pub publish_time: u64,  // TAI64 timestamp
}
```

```
#[storage(read)]
fn get_price_internal(price_feed_id: PriceFeedId) -> Price {
    let contract_id = storage.pyth_contract_id.read();
    require(contract_id != ZERO_B256, Error::OracleContractIdNotSet);

    let oracle = abi(PythCore, contract_id);
    let price = oracle.price(price_feed_id);
    price
}
```

Impact

- Market manipulation through exploitation of stale prices
- Incorrect valuation of collateral and borrowed assets
- Risk of liquidations based on outdated pricing

Risk Score

BVSS: AO:A/AC:M/AX:L/C:N/I:N/A:M/D:M/Y:N/R:N/S:C (5.2)

Remediation

Implemented time validation checks to reject stale prices.

Fix Commit: 622bd073b48cfc211e8b7d6d0474186c61903e6f

---

3. Missing Price Feed Validation

Severity: Medium
Status: SOLVED

Description

The oracle price retrieval function lacked validation of returned values such as zero price, high confidence intervals, or invalid exponents.

Technical Details

```
#[storage(read)]
fn get_price_internal(price_feed_id: PriceFeedId) -> Price {
    let contract_id = storage.pyth_contract_id.read();
    require(contract_id != ZERO_B256, Error::OracleContractIdNotSet);

    let oracle = abi(PythCore, contract_id);
    let price = oracle.price(price_feed_id);
    price
}
```

Impact

- Acceptance of invalid or zero price data
- Inaccurate financial calculations and liquidations

Risk Score

BVSS: AO:A/AC:L/AX:L/C:N/I:L/A:L/D:L/Y:N/R:N/S:C (4.7)

Remediation

Sanity checks added:

- Non-zero price
- Acceptable confidence ratio
- Reasonable exponent validation

Fix Commit: 622bd073b48cfc211e8b7d6d0474186c61903e6f

---

4. Decimal Mismatch May Cause Function Revert or Miscalculation

Severity: Low
Status: SOLVED

Description

Functions such as `collateral_value_to_sell()` and `available_to_borrow()` failed when the base token had more decimals than the collateral token, either reverting or producing incorrect calculations.

Technical Details

```
// Example causing underflow:
let scale = u256::from(10_u64).pow(
    collateral_configuration.decimals -
storage.market_configuration.read().base_token_decimals,
);

// Example causing wrong borrow limit:
let scale = u256::from(10_u64).pow(
    collateral_configuration.decimals + price_exponent -
market_configuration.base_token_decimals,
);
borrow_limit += amount * price / scale;
```

Impact

- External integrations (bots, APIs) revert unexpectedly
- Borrowing limits underestimated
- Potential market inefficiency or minor griefing

Risk Score

BVSS: AO:A/AC:H/AX:M/C:N/I:L/A:L/D:L/Y:N/R:N/S:L (2.1)

Remediation

Implemented conditional logic to compare and compute proper scale depending on which decimal is greater, including fallback if division is not needed.

Fix Commit: 9132747331188b86dd8cbf9a1ca37b811d08dddb

Recommendations

Immediate Actions

1. **Oracle Monitoring**: Implement real-time monitoring of price anomalies.
2. **Circuit Breakers**: Add fallback mechanisms to halt operations on extreme deviation.
3. **Decimal Handling Checks**: Normalize decimal logic across markets for consistency.

Long-term Improvements

1. **Test Coverage**: Expand unit and fuzz testing for edge cases.
2. **Documentation**: Update oracle and pricing behavior documentation.
3. **Scheduled Reviews**: Perform periodic re-audits post-upgrades.

Conclusion

The audit identified **four** total vulnerabilities, including one high, two medium, and one low severity issue. All vulnerabilities have been fixed and validated by the SafeEdges team. Swaylend's oracle and financial logic now include robust validation and fallback measures to protect user funds.

Risk Assessment Summary

- **High Severity Issues**: 1 (Resolved)
- **Medium Severity Issues**: 2 (Resolved)
- **Low Severity Issues**: 1 (Resolved)
- **Total Issues**: 4 (All Resolved)

Follow-up Recommendations

SafeEdges recommends conducting a follow-up security review within six months or immediately after significant code changes to maintain a strong security posture.

Audit Conducted By: SafeEdges Security Team
Report Generated: July 25, 2025