SHERLOCK

Security Review For Timeswap

Collaborative Audit Prepared For: **Timeswap**
Lead Security Expert(s): **0x73696d616f**
ctf_sec
newspacexyz
ParthMandale
Date Audited: **September 15 - November 6, 2025**

Introduction

This report reviews the TIME protocol suite, composed of:

- Universal Permission Relay (UPR)
- TimeBoundToken (TBT)
- Timeswap V3
- Periphery contracts and adapters

Executive Summary

The protocol introduces a modular, time-aware financial architecture where permissions, balances, and liquidity are modeled explicitly rather than implicitly. Each component is designed to maximize composability while preserving deterministic accounting and invariant safety.

- **Universal Permission Relay (UPR)**

The Universal Permission Relay enhances contract composability by decoupling execution authority from msg.sender. It enables users and contracts to relay scoped permissions through signatures or pre-approved contexts, allowing coordinated multi-contract workflows without permanently transferring control.

The audit focuses on:

- Permission scoping and isolation
- Replay protection and context integrity
- Execution and simulation consistency
- Reentrancy and privilege escalation risks

UPR serves as a foundational composability layer across the TIME ecosystem.

- **TimeBoundToken (TBT)**

TimeBoundToken implements a timeline-based accounting mechanism. Rather than treating balances as static values, TBT models ownership and yield as explicit, time-indexed state transitions. Balance updates occur through deterministic synchronization along a structured timeline. Growth and yield are applied via defined time deltas, enabling programmable time-bound assets and accurate historical state reconstruction.

The audit evaluates:

- Correctness of time-indexed accounting
- Synchronization ordering and invariant preservation
- Precision and overflow safety

- Adapter hook interactions
- Historical balance consistency

The timeline mechanism is central to TBT's design and enables composable yield primitives across defined time horizons.

- **Timeswap V3**

Timeswap V3 introduces an AMM Book architecture for time-indexed assets. Liquidity is organized into structured rate and time buckets within an automated market making framework. Rather than relying on a single pooled curve, the AMM Book distributes liquidity across discrete pricing intervals while preserving deterministic AMM invariants.

The review evaluates:

- Bucket-level accounting correctness
- AMM invariant preservation across rate bands
- Routing and shared liquidity behavior
- Edge-case handling and cross-component safety

- **Periphery and Adapters**

Periphery contracts and adapters provide integration layers between core protocol components and external systems. Adapters enable external yield sources, vaults, or strategies to integrate with TBT's timeline accounting model, while periphery contracts simplify user interaction and routing.

The audit focuses on:

- Input validation and parameter safety
- External call handling and trust assumptions
- Adapter hook correctness
- Permission boundary enforcement

While not core to invariant enforcement, these components materially affect protocol safety and integration behavior.

- **Conclusion**

The TIME protocol combines composable permission relaying, timeline-based token accounting, and an AMM Book market structure. This architecture enables expressive and modular financial primitives.

Scope

Repository: Timeswap-Labs/Time-V3

Audited Commit: f30cdecf4d817ec9a3af08240cdfa22a06f26d71

Final Commit: 076c4bb8b34977a1951dd64f5d8a7d2c925e18ff

Final Commit Hash

076c4bb8b34977a1951dd64f5d8a7d2c925e18ff

Findings

Each issue has an assigned severity:

- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
20	8	10

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue H-1: Incorrect Day Validation in MonthBased-TimeBound.sol [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/102

Summary

The `isActive` function in `MonthBasedTimeBound.sol` has several logical flaws in its day validation logic that can cause incorrect results

Vulnerability Detail

- Contradictory Condition: This condition can never be true as a day cannot be both greater than and equal to monthBased.day simultaneously.

```
(currentDay > monthBased.day) && (currentDay == monthBased.day)
```

- Inconsistent Day Overflow Handling: For the time parameter, it properly handles day overflow with `(day == monthBased.day || (monthBased.day > daysInMonth && day == daysInMonth))`, but for the current parameter, it only checks `currentDay == monthBased.day` without considering day overflow. For months with fewer days than monthBased.day (e.g., February with day 31), the function will never consider the last day of the month as valid for the current parameter. This could lead to incorrect isActive results when the current time is on the last day of a short month.
- Incorrect Days in Month Calculation: The function calculates daysInMonth for the time parameter's month but doesn't recalculate it for the current parameter's month, which could lead to incorrect validation.

Impact

The function may return an incorrect `isActive` status

Code Snippet

```
result = ((monthsPassedPlusOne - monthBased.startedFrom) % monthBased.period == 0)
    // two cases: day exactly matched monthBased.day OR giving day equal to the last
    ↪  day of the month due to monthBased.day > daysInMonth
    && (day == monthBased.day || (monthBased.day > daysInMonth && day ==
    ↪  daysInMonth))
        && (secondsOfTheDayPassed == monthBased.secondsOfTheDay);

    if (result) {
```

```
        (uint256 currentYear, uint256 currentMonth, uint256 currentDay, uint256
        ↪  currentHour, uint256 currentMinute, uint256 currentSecond) =
        ↪  DateTimeLibrary.timestampToDateTime(current);

        uint256 currentMonthPassedPlusOne = (currentYear - 1970) * 12 + currentMonth;
        uint256 currentSecondsOfTheDayPassed = currentHour * 3600 + currentMinute *
        ↪  60 + currentSecond;

        if (
            // testing if current month also satisfied
            ((currentMonthPassedPlusOne - monthBased.startedFrom) %
            ↪  monthBased.period == 0)
                && (currentDay > monthBased.day)
                && (
                    currentDay == monthBased.day
                        && currentSecondsOfTheDayPassed > monthBased.secondsOfTheDay
                )
        )
```

Recommendation

`isActive` function needs to be mitigated:

- Correct the contradictory condition `(currentDay > monthBased.day) && (currentDay == monthBased.day)`
- Apply the same day overflow logic to both time and current parameters:

```
// Get days in current month
uint256 currentDaysInMonth = DateTimeLibrary._getDaysInMonth(currentYear,
↪  currentMonth);

if (
    ((currentMonthPassedPlusOne - monthBased.startedFrom) % monthBased.period == 0)
    ↪  &&
    (currentDay > monthBased.day ||
        (
            (currentDay == monthBased.day ||
             (monthBased.day > currentDaysInMonth && currentDay ==
             ↪  currentDaysInMonth)
            ) &&
            (currentSecondsOfTheDayPassed > monthBased.secondsOfTheDay)
        )
    )
)
```

Issue H-2: Incorrect condition in `MonthBasedTimeBound::isActive` function [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/103

Vulnerability Detail

In `isActive`, the following condition is used which is always `false`:

```
(currentDay > monthBased.day) &&
(currentDay == monthBased.day &&
    currentSecondsOfTheDayPassed > monthBased.secondsOfTheDay)
```

Impact

`isActive` always falls through to the else case.

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/main/Time-V3/src/TimeBound/Implementation/MonthBasedTimeBound/MonthBasedTimeBound.sol#L122-L130

Tool Used

Manual Review

https://github.com/sherlock-audit/2025-09-timeswap-v3/pull/57#discussion_r2351935715

Recommendation

Review the mitigation in report
https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/104

Issue H-3: Incorrect day overflow handling in `MonthBasedTimeBound::isActive` [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/104

Summary

The `isActive` function checks day overflow for the `time` parameter but **ignores day overflow for the `current` parameter**.

Vulnerability Detail

Current Logic (Inconsistent):

```
// For 'time' parameter - CORRECT
(day == monthBased.day || (monthBased.day > daysInMonth && day == daysInMonth))

// For 'current' parameter - MISSING day overflow check
(currentDay == monthBased.day && currentSecondsOfTheDayPassed >
↪  monthBased.secondsOfTheDay)
```

When `monthBased.day > daysInMonth` (e.g., day 31 in February), the current logic doesn't properly handle the case where `currentDay == daysInMonth`.

Example:

- Config: Monthly on 31st
- Current: Feb 28, 2024 (last day of Feb)
- Current logic: `currentDay > monthBased.day` (28 > 31) = false
- Should be: `currentDay == daysInMonth` (28 == 28) = true

Impact

Incorrect result from `isActive` function.

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/main/Time-V3/src/TimeBound/Implementation/MonthBasedTimeBound/MonthBasedTimeBound.sol#L126-L129

Tool Used

Manual Review

https://github.com/sherlock-audit/2025-09-timeswap-v3/pull/57#discussion_r2351939567

Recommendation

```
if (
    ((currentMonthPassedPlusOne - monthBased.startedFrom) % monthBased.period == 0)
    ↪  &&
    (currentDay > monthBased.day ||
        (
            (currentDay == monthBased.day) || (monthBased.day > daysInMonth &&
            ↪  currentDay == daysInMonth)) &&
            (currentSecondsOfTheDayPassed > monthBased.secondsOfTheDay)
        )
    )
) {
    ...
}
```

Issue H-4: Incorrect `MonthBasedTimeBound::isActive` evaluation when current is before first activation [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/105

Vulnerability Detail

The `isActive` function doesn't check if the `current` time is **before the first possible activation** of the time-bound configuration. If `current` is before the first activation time, `maxConcurrent` check is performed using the period from `monthBased.startedFrom` (not `current`) to test time.

Similar issue occurs in `SecondBasedTimeBound.isActive` function.

Example:

- Config: Quarterly starting from month 13 (Sep 2025)
- `time`: Jan 15, 2026 (valid activation)
- `current`: Aug 31, 2025 (before first activation)

Impact

Incorrect result from `isActive` function.

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/main/Time-V3/src/TimeBound/Implementation/MonthBasedTimeBound/MonthBasedTimeBound.sol#L129-L134

Tool Used

Manual Review

https://github.com/sherlock-audit/2025-09-timeswap-v3/pull/57#discussion_r2351949082

Recommendation

Review the mitigation in report
https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/104

Issue H-5: `MonthBasedTimeBound::next` and `MonthBasedTimeBound::previous` function ignore `Day-of-Month` [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/106

Vulnerability Detail

The `next` function only considers month/year and time-of-day, but **ignores the day of the month** when calculating the next activation.

Similar issue also occurs in `previous` function.

Current Logic (Wrong):

```
if (secondsPassed > monthBased.secondsOfTheDay) {
    frequency += 1; // Only checks time, not day and month
}
```

Example:

- Config: Monthly on 15th at 11:00 AM
- Input: Jan 20, 2024 10:00 AM
- Current: Returns Jan 15, 2024 11:00 AM (past date!)
- Correct: Should return Feb 15, 2024 11:00 AM

Impact

Incorrect `nextTime` from `next` function.

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/main/Time-V3/src/TimeBound/Implementation/MonthBasedTimeBound/MonthBasedTimeBound.sol#L162-L169

Tool Used

Manual Review

https://github.com/sherlock-audit/2025-09-timeswap-v3/pull/57#discussion_r2352045090

Recommendation

Fix of `next` function:

```
uint256 frequency = (month + (year - 1970) * 12 - monthBased.startedFrom) /
↪  monthBased.period;
uint256 secondsPassed = hour * 3600 + minute * 60 + second;

// Check if current month and day is in the activation period
if ((month + (year - 1970) * 12 - monthBased.startedFrom) % monthBased.period == 0)
↪  {
    // We're in an activation period, check if we've passed the activation day/time
    if (day > monthBased.day ||
        (day == monthBased.day && secondsPassed > monthBased.secondsOfTheDay)) {
        frequency += 1;
    }
} else {
    frequency += 1;
}
```

Issue H-6: Incorrect `frequency` increment in `MonthBasedTimeBound::next` and `MonthBasedTimeBound::previous` function [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/107

Vulnerability Detail

In `next` function, `frequency` calculation always adds `+ 1` at the end, forcing it to look for the **next** period instead of checking if we're still within the **current** period.

Similar issue also occurs in `previous` function.

Current Logic (Wrong):

```
    uint256 frequency = ((month + (year - 1970) * 12 -
@>      (monthBased.startedFrom - 1)) / monthBased.period) + 1;
    ...
```

Example:

- Config: Quarterly on 5th (`period: 3`)
- Input: March 1, 1970
- Current: Calculates frequency = 2 and Returns June 5, 1970
- Correct: Should return Apr 5, 1970
- Problem: March 1st is still in Period 1, but function jumps to Period 2

Impact

Incorrect `nextTime` from `next` function.

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/main/Time-V3/src/TimeBound/Implementation/MonthBasedTimeBound/MonthBasedTimeBound.sol#L162-L169

Tool Used

Manual Review

https://github.com/sherlock-audit/2025-09-timeswap-v3/pull/57#discussion_r2352044129

Recommendation

Review the mitigation in report
https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/106

Issue H-7: Incorrect Bitmasking in `MonthBasedTimeBound::isTimeConfigValid` and `MonthBasedTimeBound::toMonthBased` Functions [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/109

Summary

The bit shifting calculations in both `isTimeConfigValid` and `toMonthBased` functions had incorrect mask values.

Vulnerability Detail

According to the field layout documentation, the bit positions should be:

- `day`: 1 byte at bits 64-71 (8 bits)
- `secondsOfTheDay`: 3 bytes at bits 40-63 (24 bits)
- `startedFrom`: 2 bytes at bits 24-39 (16 bits)
- `period`: 2 bytes at bits 8-23 (16 bits)
- `maxConcurrent`: 1 byte at bits 0-7 (8 bits)

The bit shifting calculations in both `isTimeConfigValid` and `toMonthBased` functions had incorrect mask values:

```
monthBased.secondsOfTheDay = uint24((uint256(timeConfig) >> 40) % (1 << 8));   //
↪  Wrong: 8 bits
monthBased.startedFrom = uint16((uint256(timeConfig) >> 24) % (1 << 32));      //
↪  Wrong: 32 bits
monthBased.period = uint16((uint256(timeConfig) >> 8) % (1 << 48));            //
↪  Wrong: 48 bits
```

Impact

Improper decoding of time config fields.

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/main/Time-V3/src/TimeBound/Implementation/MonthBasedTimeBound/Type/MonthBased.sol#L50-L58

Tool Used

Manual Review

https://github.com/sherlock-audit/2025-09-timeswap-v3/pull/58#issuecomment-3294568415

Recommendation

```
monthBased.secondsOfTheDay = uint24((uint256(timeConfig) >> 40) % (1 << 24));  //
↪  Correct: 24 bits
monthBased.startedFrom = uint16((uint256(timeConfig) >> 24) % (1 << 16));      //
↪  Correct: 16 bits
monthBased.period = uint16((uint256(timeConfig) >> 8) % (1 << 16));            //
↪  Correct: 16 bits
```

Issue H-8: Incorrect principal calculation in `TimeswapV3AddLiquidityMidware::calculateAddLiquidity` [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/110

Vulnerability Detail

The `calculateAddLiquidity` function currently scales the principal using `approvedLiquidityGivenPrincipal`, but `approvedLiquidityGivenInterest` is the intended proportional factor.

```
    if (
        approvedLiquidityGivenPrincipal < approvedLiquidityGivenInterest
    ) {
        liquidityParameter.principal = maxPrincipal;
        liquidityParameter.liquidity = approvedLiquidityGivenPrincipal;
        liquidityParameter.interest = binInterest.mulDiv(
            approvedLiquidityGivenPrincipal,
            binLiquidity,
            Math.Rounding.Ceil
        );
    } else {
        liquidityParameter.interest = maxInterest;
        liquidityParameter.liquidity = approvedLiquidityGivenInterest;
        liquidityParameter.principal = binPrincipal.mulDiv(
@>          approvedLiquidityGivenPrincipal,
            binLiquidity,
            Math.Rounding.Ceil
        );
    }
```

Impact

Incorrect `liquidityParameter.principal` calculation

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/main/Time-V3/src/Midware/TimeswapV3AddLiquidityMidware/TimeswapV3AddLiquidityMidware.sol#L201-L205

Tool Used

Manual Review

https://github.com/sherlock-audit/2025-09-timeswap-v3/pull/11#discussion_r2362243399

Recommendation

```
} else {
    liquidityParameter.interest = maxInterest;
    liquidityParameter.liquidity = approvedLiquidityGivenInterest;
    liquidityParameter.principal = binPrincipal.mulDiv(
--      approvedLiquidityGivenPrincipal,
++      approvedLiquidityGivenInterest,
        binLiquidity,
        Math.Rounding.Ceil
    );
}
```

Issue H-9: Incorrect principal calculation in `TimeswapV3SwapForwardMidware::simulateSwapForwardGivenTotal` [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/111

Vulnerability Detail

When the computed interest exceeds `binInterest`, the fallback recalculates `parameter.principal` by dividing the truncated interest by the full exchange rate instead of `(exchangeRate - 1)`.

```
    {
        (uint256 duration, , , uint256 binInterest, , ) = timeswapV3
            .binState(liquidityTokenId);

        Float128x128 exchangeRate = bidRate
            .multiply(duration)
            .exp();

        parameter.principal = bucketTotal.divide(
            exchangeRate,
            true
        );
        parameter.interest = bucketTotal - parameter.principal;

        if (parameter.interest > binInterest) {
            parameter.interest = binInterest;
            parameter.principal = parameter.interest.divide(
@>              exchangeRate,
                true
            );
        }
    }
```

Impact

Incorrect `parameter.principal` calculation

Code Snippet

http://github.com/sherlock-audit/2025-09-timeswap-v3/blob/main/Time-V3/src/Midware/TimeswapV3SwapForwardMidware/TimeswapV3SwapForwardMidware.sol#L288-L304

Tool Used

Manual Review

https://github.com/sherlock-audit/2025-09-timeswap-v3/pull/27#discussion_r2366540807

Recommendation

```
        Float128x128 exchangeRate = bidRate
            .multiply(duration)
            .exp();

        parameter.principal = bucketTotal.divide(
            exchangeRate,
            true
        );
        parameter.interest = bucketTotal - parameter.principal;

        if (parameter.interest > binInterest) {
            parameter.interest = binInterest;
            parameter.principal = parameter.interest.divide(
--              exchangeRate,
++              exchangeRate - Float128x128Library.ONE,
                true
            );
        }
```

Issue H-10: Incorrect `interest` and `bucketTotal` calculation in `TimeswapV3SwapBackwardMidware::simulateSwapBackwardGivenTotal` [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/112

Vulnerability Detail

The function recalculates interest after potentially truncating principal but current implementation is fully incorrect and this forces `bucketTotal` to drop to .

```
    {
        ...

        Float128x128 exchangeRate = askRate
            .multiply(duration)
            .exp();

        parameter.principal = bucketTotal.divide(
            exchangeRate,
            false
        );

        if (parameter.principal > binPrincipal) {
            parameter.principal = binPrincipal;
        }

@>      parameter.interest = bucketTotal - parameter.principal;
    }

    bucketTotal -= (parameter.principal + parameter.interest);
```

Impact

Incorrect `parameter.interest` calculation leads to zero `bucketTotal`

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/main/Time-V3/src/Midware/TimeswapV3SwapBackwardMidware/TimeswapV3SwapBackwardMidware.sol#L292-L305

Tool Used

Manual Review

https://github.com/sherlock-audit/2025-09-timeswap-v3/pull/19#discussion_r2429218305

Recommendation

```
        {
            ...
            Float128x128 exchangeRate = askRate
                .multiply(duration)
                .exp();
            parameter.principal = bucketTotal.divide(
                exchangeRate,
                false
            );
++          parameter.interest = bucketTotal - parameter.principal;
            if (parameter.principal > binPrincipal) {
                parameter.principal = binPrincipal;
++              parameter.interest = parameter.principal.multiply(
++                  exchangeRate - Float128x128Library.ONE,
++                  false
++              );
            }
--          parameter.interest = bucketTotal - parameter.principal;
        }
        bucketTotal -= (parameter.principal + parameter.interest);
        interest += parameter.interest;
```

Issue H-11: `timeBoundERC4626.fee` is never initialized

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/113

Vulnerability Detail

`timeBoundERC4626.fee` is never initialized in the `create` function, so its value is always 0.

`timeBoundERC4626.fee` is used to accumulate the `timeBoundERC4626.fees` so `timeBoundERC4626.fees` is always 0.

```
history.fees = yield.sideStreamYields[0].multiply(
    timeBoundERC4626.fee,
    false
);

yield.sideStreamYields[0] -= history.fees;

timeBoundERC4626.fees += history.fees;
```

Impact

`timeBoundERC4626.fees` is never accumulated.

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/main/Time-V3/src/TimeBoundERC4626/TimeBoundERC4626.sol#L690-L697

Tool Used

Manual Review

https://github.com/sherlock-audit/2025-09-timeswap-v3/pull/54#discussion_r2388276113

Recommendation

Set `timeBoundERC4626.fee` in `create` funciton.

Issue H-12: Missed maturities in TimeBoundLiquidityMining due to incorrect starting timestamp `previous` [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/121

Summary

`TimeBoundLiquidityMining::beforePrepareHistory()` goes through every base change sequentially and checks for maturity expiration in between. However, the starting point is wrong.

Vulnerability Detail

When syncing, it calls `TimeBoundLiquidityMining::beforePrepareHistory()`, and it goes through the underlying base token history, and assigns rewards if the maturity exceeds the `previous + diff.timeElapsed` time. However, `previous` is set to `timeBoundToken.lastUpdated()` after it is synced, so this will be block.timestamp, getting all rewards wrong.

Impact

Lost rewards

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/pull/69/files#diff-4467cdfdaca66633b40b923e80fae5ed9a649ccd0cc8560b7546ab3f45461933R633

Tool Used

Manual Review

Recommendation

Fetch the last update before syncing the base token.

Issue H-13: `TimeBoundLiquidityMining::beforePrepareHistory()` doesn't check if the maturity is before the next base updated [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/122

Summary

`TimeBoundLiquidityMining::beforePrepareHistory()` is missing maturity timeline checks, causing maturities to be incorrectly rewards.

Vulnerability Detail

Here is the vulnerable code:

```
for (uint256 i; i < baseHistory.diffHistories.length; ++i) {
    uint256 nextBase = previous +
        baseHistory.diffHistories[i].timeElapsed;

    while (previous < nextBase) {
        uint256 nextLiquidityMining = timeBound.next(
            timeConfig,
            previous
        ); //@audit missing nextLiquidityMining < nextBase check

        mapping(uint256 => uint256)
            storage _rewards = timeBoundLiquidityMining
                .rewardsPerMaturity[nextLiquidityMining];

        scratchDiffHistories[scratchLength].timeElapsed =
            nextLiquidityMining -
            previous;

        for (uint256 j; j < liquidityMiningSideStreamCount; ++j) {
            scratchDiffHistories[scratchLength].rewardAmounts[
                    j
                ] = _rewards[j];
            diffYields[i].sideStreamYields[
                j + baseSideStreamCount
            ] += _rewards[j];
            delete _rewards[j];
        }

        scratchLength++;

        previous = nextLiquidityMining;
```

```
    }
}
```

As can be seen, there is a missing nextLiquidityMining < nextBase check.

Impact

Maturities will be incorrectly attributed.

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/pull/69/files#diff-4467cdfdaca66633b40b923e80fae5ed9a649ccd0cc8560b7546ab3f45461933R648

Tool Used

Manual Review

Recommendation

Add the check.

Issue H-14: Incorrect `TimeBoundLiquidityMining::beforePrepareHistory()` variable `previous` update after maturity loop [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/123

Summary

`TimeBoundLiquidityMining::beforePrepareHistory()` goes through all maturities in between base changes timestamps, and updates previous correspondingly. However, due to this, once it tries to go to the next base change and iterate again, previous will be wrongly the last maturity mined, and the time change will be incorrect.

Vulnerability Detail

As can be seen below, `previous` will be incorrect after the maturity while loop.

```
for (uint256 i; i < baseHistory.diffHistories.length; ++i) {
    uint256 nextBase = previous +
        baseHistory.diffHistories[i].timeElapsed;

    while (previous < nextBase) {
        uint256 nextLiquidityMining = timeBound.next(
            timeConfig,
            previous
        );

        mapping(uint256 => uint256)
            storage _rewards = timeBoundLiquidityMining
                .rewardsPerMaturity[nextLiquidityMining];

        scratchDiffHistories[scratchLength].timeElapsed =
            nextLiquidityMining -
            previous;

        for (uint256 j; j < liquidityMiningSideStreamCount; ++j) {
            scratchDiffHistories[scratchLength].rewardAmounts[
                    j
                ] = _rewards[j];
            diffYields[i].sideStreamYields[
                j + baseSideStreamCount
            ] += _rewards[j];
            delete _rewards[j];
        }

        scratchLength++;
```

```
            previous = nextLiquidityMining; //@audit will be wrong when setting nextBase
        }
    }
```

Impact

Incorrect rewards.

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/pull/69/files#diff-4467cdfdaca66633b40b923e80fae5ed9a649ccd0cc8560b7546ab3f45461933R673

Tool Used

Manual Review

Recommendation

Previous should be a separate variable.

Issue H-15: `TimeBoundToken::_validate()` doesn't check that the deltas are in correct order [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/124

Summary

`TimeBoundToken::_validate()` lacks a check for the delta values in the TimeBalance to be in chronological order.

Vulnerability Detail

`TimeBoundToken::_validate()` never checks that the deltas are ordered.

```
function _validate(
    TimeBalance memory timeBalance,
    TimeBoundInterface timeBound,
    bytes32 timeConfig
) private view {
    uint256 accumulator = timeBalance.amount;

    for (uint256 i; i < timeBalance.timeDeltas.length; ++i) {
        if (
            !timeBound.isActive(
                timeConfig,
                timeBalance.timeDeltas[i].time,
                block.timestamp
            )
        ) {
            revert TimeBoundTokenError.InvalidTimeBalance(timeBalance);
        }

        accumulator = accumulator.addDelta(
            timeBalance.timeDeltas[i].deltaAmount
        );
    }
}
```

As a result, anybody can mint a time balance in incorrect order and DoS the TimeBoundToken. This happens because it will underflow when calculating the times elapsed in `Total.prepareHistory()`, link.

Impact

DoSed TimeBound token due to underflow.

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/main/Time-V3/src/TimeBoundToken/TimeBoundToken.sol#L882

Tool Used

Manual Review

Recommendation

Check that the deltas are in chronological order.

Issue H-16: `midRate.multiply(duration).exp()` check can DOS `createBin` in `TimeswapV3` [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/126

Vulnerability Detail

In `TimeswapV3` when computing the `LiquidityRatio`,

```
bin.principalLiquidityRatio =
                midRate.multiply(duration).exp() -
                Float128x128Library.ONE;
```

the code just revert with no reason `midRate.multiply(duration).exp()`

```
/// @dev Exponentiate a float.
  /// @notice The function can only handle precision of up to 48 bits. The last 80
   ↪  bits of the given float must be zero.
  /// @notice The function can only handle values of up to 16.
  /// @notice Reverts when float is outside the range explained above.
  /// @param float The float to exponentiate.
  /// @return result The result of the exponentiation.
  function exp(Float128x128 float) internal pure returns (Float128x128 result) {
      if (Float128x128.unwrap(float) % (1 << 80) != 0) {
          revert();
      }
```

the exp function requires that `The last 80 bits of the given float must be zero.` must `midRate.multiply(duration)` easily break this requirement.

POC:

- change the branch to `TimeSwapV1-exp-no-reason-revert`
- run test: `forge test --match-test "test_TSV3_CreateBin_Happy" -vv`

```
Failing tests:
Encountered 1 failing test in
 ↪  test/TimeswapV3/TimeswapV3.t.sol:HappyPath_TimeBoundERC4626_Create
[FAIL: EvmError: Revert] test_TSV3_CreateBin_Happy() (gas: 436906)
```

Impact

No bin can be traded in `TimeswapV3` so no trade can be conducted.

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/149bab1c27f1008a6584f86eda12662d7da855c8/Time-V3/src/Primitive/Float128x128.sol#L457

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/149bab1c27f1008a6584f86eda12662d7da855c8/Time-V3/src/Primitive/Float128x128.sol#L456

Recommendation

The fix spans multiple places.

```
function _powQ128x128(Float128x128 base, uint256 n) private pure returns
↪  (Float128x128 out) {
      uint256 b = Float128x128.unwrap(base);
      uint256 r = _Q128; // 1.0
      while (n != 0) {
          if (n & 1 == 1) {
             r = Math.mulDiv(r, b, _Q128);
         }
         b = Math.mulDiv(b, b, _Q128);
         n >>= 1;
     }
     out = Float128x128.wrap(r);
 }
```

In `createBin` , apply the change:

```
// Discrete compounding: PLR = (midRate^duration) - 1
   bin.principalLiquidityRatio = _powQ128x128(midRate, duration) -
   ↪  Float128x128Library.ONE;

   bin.minimumInterest = timeswapV3.minimumPrincipal.multiply(
       bin.principalLiquidityRatio,
       true
   );
```

In `verifySwapForward` and `verifyBackward`, apply the change

```
// @audit fix
// Float128x128 exchangeRate = bin.bidRate.multiply(duration).exp();
// Discrete compounding: (bidRate)^duration
 Float128x128 exchangeRate = _powQ128x128(bin.bidRate, duration);
```

This make the test runs.

Issue H-17: `FullMath` implementation contains fundamental math flaw when `mulDivRoundingUp` [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/127

Vulnerability Detail

```
function mulDivRoundingUp(
        uint256 a,
        uint256 b,
        uint256 denominator
    ) internal pure returns (uint256 result) {
        result = mulDiv(a, b, denominator);
        if (mulmod(a, b, denominator) > 0) {
            require(result < type(uint256).max);
            result++;
        }
    }
```

In both the fast path (prod1 == 0) and the 512→256 path, the code always increments the truncated quotient when roundUp is true—even if the division is exact (remainder = 0).

That's not "round up if there's a remainder,"

that's "always add 1," which over-rounds exact divisions

In Uniswap V3 Full math cod, if remainder is 0, there is no ++

Impact

Fundamental math error

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/149bab1c27f1008a6584f86eda12662d7da855c8/Time-V3/src/Primitive/FullMath.sol#L106

Recommendation

Change the math implementation to round up if there's a remainder.

Issue H-18: A newly created `timeBoundToken` with no new history cannot be minted [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/131

Vulnerability Detail

It is expected that user calls `syncAccount` before any tbt state changing action, otherwise transaction revert in the line of code below.

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/149bab1c27f1008a6584f86eda12662d7da855c8/Time-V3/src/TimeBoundToken/TimeBoundToken.sol#L527

```
if (
    (account.lastTimeUpdated != block.timestamp ||
        account.lastTimestep != timeBoundToken.total.lastTimestep)
) {
    revert TimeBoundTokenError.UserNotSynced(receiver);
}
```

the `account.lastTimestep != timeBoundToken.total.lastTimestep` is false even we called `syncAccount` for a newly created tbt token with no new history.

When the code calls `syncAccount`, the code below runs.

```
hasNewHistory = history.isNotEmpty();

if (hasNewHistory) {
```

For a newly created token, there is no new history so the if branch does not run because `hasNewHistory` is `false`

So hasNewHistory is `false`, then `account.applySync` which update account.lastTimestep never run.

POC

Change the branch to `TimeBoundERC4626-test-v1` to run the POC.

```
git fetch origin
git checkout -b TimeBoundERC4626-test-v1
```

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/250f4506f2fa56aa79e08091eccdbf04fdc955a9/Time-V3/test/ERC4626Adapter/TimeBoundERC4626.t.sol#L294

Can run the test:

```
forge test -vv --match-test "test_MintGivenAmount_Happy_EOAPermissions"
```

transaction revert with

```
[FAIL: UserNotSynced(0x23DCc02fa40dE69d2BcE5DcEA4584d6E9741Cf4e)]
↪  test_MintGivenAmount_Happy_EOAPermissions() (gas: 622061)
```

the transaction revert here when minting TBT.

Impact

A newly created `timeBoundToken` with no new history cannot be minted.

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/149bab1c27f1008a6584f86eda12662d7da855c8/Time-V3/src/TimeBoundToken/Type/Account.sol#L377

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/149bab1c27f1008a6584f86eda12662d7da855c8/Time-V3/src/TimeBoundToken/TimeBoundToken.sol#L527

Recommendation

Even if the TBT has no new history, still properly sync the `account.lastTimestep`

Discussion

Mathepreneur

The syncTotal and syncAccount always sync starting from timestamp 0. Thus when there is a newly created tbt, syncTotal and syncAccount will still have a new history starting from timestamp 0.

ctf-sec

Whether the token has new history depends on:

```
hasNewHistory = history.isNotEmpty();
```

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/149bab1c27f1008a6584f86eda12662d7da855c8/Time-V3/src/TimeBoundToken/Type/TimeBoundTokenSync.sol allowbreak #L79

```
function isNotEmpty(
    History memory history
) internal pure returns (bool result) {
```

```
        result =
            history.diffHistories.length > 0 ||
            history.timeElapsed > 0 ||
            history.amountGrowthIncrease.amount > 0;
        if (!result) {
            for (
                uint256 j;
                j < history.sideStreamRebasingAmountGrowthIncreases.length;
                ++j
            ) {
                if (
                    history
                        .sideStreamRebasingAmountGrowthIncreases[j]
                        .amountGrowth
                        .amount > 0
                ) {
                    result = true;
                    break;
                }
            }
        }
    }
```

when there is a newly created tbt, syncTotal and syncAccount will still have a new history starting from timestamp 0.

`result` is `False` in this case.

Mathepreneur

When it started from timestamp 0, then history.timeElapsed > 0. Thus it is not empty.

Mathepreneur

I believe this is due to not having this code line in prepareHistory

upcoming.timeElapsed = block.timestamp - lastTimeUpdated;

Which I added in the fix. Kindly do check.

Issue H-19: Incorrect pro-rata side stream in `Bin.sol` [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/132

Summary

`Bin.sol::beforePrepareHistory()` else branch, corresponding to `nextBase > nextLiquidity`, has an incorrect pro-rata calculation leading to incorrect rewards.

Vulnerability Detail

The code is:

```
creditSideStreamsProRata(
    sideStreamCount,
    diffYields[syncStep.iLiquidity],
    history
        .diffHistories[syncStep.iBase]
        .sideStreamRebasingAmountGrowthIncreases,
    upcoming.timeElapsed,
    nextChange.timeElapsed
);
```

The step is set as upcoming.timeElapsed, and total is nextChange.timeElapsed. However, since nextBase > nextLiquidity, the total elapsed for this diff yield is diffHistory.timeElapsed, and the step time elapsed is the liquidity one, so nextChange.timeElapsed.

Impact

Incorrect yield

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/pull/85/files#diff-5c0afe3b84822b75478237014b09828e95f53f2a4dba60f73b2ffb28045dab4aR559

Tool Used

Manual Review

Recommendation

Fix the pro-rata calculation

Issue H-20: Incorrect maturity check in the changes component after the last diff in `Bin.sol` [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/133

Summary

`Bin::beforePrepareHistory()` incorrectly always uses `history.timeElapsed` in the last maturity check, causing premature maturities.

Vulnerability Detail

The mentioned code is:

```
(syncStep.previous, bin.afterMaturityPrincipal) = advanceAndLatch(
    syncStep.previous,
    history.timeElapsed,
    bounds.maturity,
    syncStep.base,
    bin.afterMaturityPrincipal
);
```

As can be seen, it always uses history.timeElapsed, but this value may be stale, and upcoming.timeElapsed should be used instead.

For example, suppose iLiquidity and iBase are at the last diff and second to last diff, respectively, and these 2 end at the same time. The else branch when ending at the same time is processed, and both iLiquidity and iBase are incremented. Now, it runs the first branch in the loop above, and decreases upcoming.timeElapsed by the last iBase diff and increases the syncStep.previous, but history.timeElapsed will stay the full value, being inflated and causing early maturity.

Impact

Early maturity, causing incorrect accounting.

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/pull/85/files#diff-5c0afe3b84822b75478237014b09828e95f53f2a4dba60f73b2ffb28045dab4aR595

Tool Used

Manual Review

Recommendation

Use upcoming.timeElapsed if smaller.

Issue M-1: TimeBound token users can DoS themselves [ACKNOWLEDGED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/96

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

There is currently no limit on the number of elements in the total and account timelines, possibly DoSing users for too many time changes.

Vulnerability Detail

As an example, take a look at `Total::increase()`, which loops over all elements in the timeline:

```
for (uint256 i; i < length; ) {
    TimeDelta memory timeDelta = timeBalance.timeDeltas[i];

    if (next == 0 || timeDelta.time < next) {
        timeline[previous] = timeDelta.time;
        timeline[timeDelta.time] = next;

        Diff storage diff = diffs[timeDelta.time];
        // Invariant: Since the node does not exist,
        // guarantees the current diff.deltaAmount is zero.
        diff.deltaAmount = timeDelta.deltaAmount;

        accumulator = accumulator.addDelta(diff.deltaAmount);
        previous = timeDelta.time;

        i++;
        continue;
    }

    if (timeDelta.time == next) {
        Diff storage diff = diffs[timeDelta.time];
        diff.deltaAmount += timeDelta.deltaAmount;

        accumulator = accumulator.addDelta(diff.deltaAmount);
        // Invariant: Even if resulting diff.deltaAmount is 0,
        // numberOfUsers > 0 cause there must be accounts that is non zero,
        // thus we do not need to prune the node.

        previous = next;
        next = timeline[previous];
```

```
        i++;
        continue;
    }

    accumulator = accumulator.addDelta(diffs[next].deltaAmount);

    previous = next;
    next = timeline[previous];
}
```

There seems to be no limit on the number of elements, potentially DoSing users.

Impact

DoSed users due to OOG.

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/main/Time-V3/src/TimeBoundToken/Type/Total.sol#L818-L857

Tool Used

Manual Review

Recommendation

The only check is if the TimeBound is active by that time, which still allows too much granularity:

```
function isActive(
    bytes32 timeConfig,
    uint256 time,
    uint256 current
) external pure override returns (bool result) {
    SecondBased memory secondBased = timeConfig.toSecondBased();

    (bool success, uint256 difference) = Math.trySub(
        time,
        secondBased.startedFrom
    );
    if (!success) {
        return false;
    }
```

```
    result =
        difference % secondBased.period == 0 &&
        (time - current) / secondBased.period < secondBased.maxConcurrent;
}
```

Discussion

Mathepreneur

What do you recommend? Here, I think this should be fine. It will be the responsibility of the devs and users to use the right TimeBound contract.

Issue M-2: Array out of bounds in `TimeBoundLiquidityMining` [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/98

Summary

`TimeBoundLiquidityMining::beforePrepareHistory()` doesn't initialize the `scratchDiffHistories[scratchLength].rewardAmounts` array.

Vulnerability Detail

As linked in the code below:

```
for (uint256 j; j < liquidityMiningSideStreamCount; ++j) {
    scratchDiffHistories[scratchLength].rewardAmounts[
            j
        ] = _rewards[j];
    diffYields[i].sideStreamYields[
        j + baseSideStreamCount
    ] += _rewards[j];
    delete _rewards[j];
}
```

The rewardAmounts array is never initialized.

Impact

DoSed contract

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/pull/69/files/cd5ca3ce30cd7212f61891329b903b8157a5dbcd#diff-4467cdfdaca66633b40b923e80fae5ed9a649ccd0cc8560b7546ab3f45461933R662

Tool Used

Manual Review

Recommendation

Initialize the array first

Issue M-3: SafeTransfer is not used [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/99

Summary

Some instances in the code use `.transfer()` directly, which fails for many tokens due to specific, but common implementations.

Vulnerability Detail

Tokens that do not return a bool on transfer, for example, revert when calling `.transfer()` directly.

Impact

DoSed funds, but only on certain tokens.

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/pull/69/files/cd5ca3ce30cd7212f61891329b903b8157a5dbcd#diff-4467cdfdaca66633b40b923e80fae5ed9a649ccd0cc8560b7546ab3f45461933R895

Tool Used

Manual Review

Recommendation

Use `safeTransfer()`

Issue M-4: Free minting due to incorrect rounding directions causing insolvency in `TimeBoundERC4626::mintGivenAmount()` [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/100

Summary

`TimeBoundERC4626::mintGivenAmount()` rounding direction is incorrect, allowing users to mint TimeBoundTokens without transferring shares.

Vulnerability Detail

As can be seen below, `updatedAssets` rounds down, and so does `convertToShares`, so an attacker is able to specify an amount of 1 and transfer in 0 shares.

```
(TimeBalance memory totalSupply, ) = timeBoundToken.balance(
    address(0),
    tokenId
);

uint256 updatedAssets = totalSupply.amount != 0
    ? (totalSupply.amount + amount).mulDiv(
        timeBoundERC4626.assets,
        totalSupply.amount
    )
    : amount;

uint256 updatedShares = timeBoundERC4626.erc4626.convertToShares(
    updatedAssets
);

shares = updatedShares - timeBoundERC4626.shares;

timeBoundERC4626.erc4626.transferFrom(sender, address(this), shares);
```

Impact

Depends on the decimals of the tokens involved, worse for lower decimal tokens.

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/pull/54/files/8a2aff9eeb7867ccb2535af6e1b5619bf280c626#diff-25d674c3ca5a1134d1a6bbbb360e3deb731927cfac96ff62acc62aa7af23e30eR285-R294

Tool Used

Manual Review

Recommendation

Round up correctly.

Issue M-5: Incorrect Transfer Condition in `TimeBoundToken.sol` [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/108

Summary

The transfer function in `TimeBoundToken.sol` has an incorrect condition that could lead to unintended transfers when the sender and receiver are different, even when the `timeBalance.isAlwaysZero()` is True.

Vulnerability Detail

It allows the transfer logic to execute when sender != receiver or when timeBalance is not zero. Whereas the state change is supposed to happen only when sender != receiver and timeBalance is not zero

Impact

The contract may perform Unnecessary State Changes.

Code Snippet

Code link

```
if (sender != receiver || !timeBalance.isAlwaysZero())
```

Recommendation

Replace the condition with:

```
if (sender != receiver && !timeBalance.isAlwaysZero())
```

Issue M-6: Loss of reward if external claim call fails sliently

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/128

Vulnerability Detail

If ERC4626 adapter is used, and `afterClaim` sliently revert (because of token transfer failure or other error), the _claim still consume side stream and the yield is lost because in ERC4626 adapter is in charge of the sending out the yield.

```
function _claim(
       TimeBoundTokenState storage timeBoundToken,
       address harvester,
       address beneficiary,
       uint256 tokenId,
       uint256[] memory amounts
   ) private {
       timeBoundToken.total.claim(amounts);

       timeBoundToken.accounts[harvester].claim(amounts);

       emit TimeBoundTokenEvent.Claim(
           harvester,
           beneficiary,
           tokenId,
           amounts
       );
   }
```

Impact

Loss of reward.

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/149bab1c27f1008a6584f86eda12662d7da855c8/Time-V3/src/TimeBoundERC4626/TimeBoundERC4626.sol#L824

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/149bab1c27f1008a6584f86eda12662d7da855c8/Time-V3/src/TimeBoundToken/TimeBoundToken.sol#L946

Recommendation

when `TimeBoundERC4626.afterClaim` reverts, we could track the amount that was supposed to be transferred to the beneficiary.

Then, we can introduce a mechanism that allows the beneficiary to manually claim that tracked amount later - once everything is functioning correctly

Issue M-7: Incorrect `TimeswapV3.nextBid` token id is used in `TimeswapV3SwapFlattenMidware.sol` [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/129

Vulnerability Detail

`TimeswapV3.nextAsk(nextLiquidityTokenId)` and `TimeswapV3.nextBid(nextLiquidityTokenId)` accept the current bin's liquidity token id, not a market/base token id

```
if (total > 0) {
    liquidityTokenId = timeswapV3.nextBid(
        baseTokenId
    );
} else {
    delete liquidityTokenId;
}
```

Impact

Incorrect swap execution and revert swap execution.

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/149bab1c27f1008a6584f86eda12662d7da855c8/Time-V3/src/Midware/TimeswapV3SwapFlattenMidware/TimeswapV3SwapFlattenMidware.sol#L351-L358

Recommendation

Always Pass in the current bin's liquidity token id when calling `nextAsk` and `nextBid`

Issue M-8: Uninitiated state in `TimeswapV3TraderPeriphery` and `TimeswapV3LiquidityProviderPeripheryy.sol` [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/130

Vulnerability Detail

In, `TimeswapV3TraderPeriphery`, the state below remains unintialized.

```
/// @inheritdoc TimeswapV3TraderPeripheryInterface
TimeswapV3SwapForwardMidwareInterface
    public immutable
    override timeswapV3SwapForwardMidware;

/// @inheritdoc TimeswapV3TraderPeripheryInterface
TimeswapV3SwapBackwardMidwareInterface
    public immutable
    override timeswapV3SwapBackwardMidware;

/// @inheritdoc TimeswapV3TraderPeripheryInterface
TimeswapV3SwapFlattenMidwareInterface
    public immutable
    override timeswapV3SwapFlattenMidware;
```

In `TimeswapV3LiquidityProviderPeriphery.sol`, the state below remains uninitialized.

```
/// @inheritdoc TimeswapV3LiquidityProviderPeripheryInterface
 TimeswapV3AddLiquidityMidwareInterface
     public immutable
     override timeswapV3AddLiquidityMidware;

 /// @inheritdoc TimeswapV3LiquidityProviderPeripheryInterface
 TimeswapV3RemoveLiquidityMidwareInterface
     public immutable
     override timeswapV3RemoveLiquidityMidware;

 /// @inheritdoc TimeswapV3LiquidityProviderPeripheryInterface
 TimeBoundERC4626Interface public immutable override timeBoundERC4626;

 /// @inheritdoc TimeswapV3LiquidityProviderPeripheryInterface
 TimeBoundLiquidityMiningInterface
     public immutable
     override timeBoundLiquidityMining;
```

Impact

Unusable `TimeswapV3TraderPeriphery` and `TimeswapV3LiquidityProviderPeriphery.sol`

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/149bab1c27f1008a6584f86eda12662d7da855c8/Time-V3/src/Periphery/TimeswapV3TraderPeriphery/TimeswapV3TraderPeriphery.sol#L52

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/149bab1c27f1008a6584f86eda12662d7da855c8/Time-V3/src/Periphery/TimeswapV3LiquidityProviderPeriphery/TimeswapV3LiquidityProviderPeriphery.sol#L54

Recommendation

Assign these state properly in the Periphery smart contract constructor.

Issue L-1: `TimeBoundToken::claim()` is not empty loop can be optimized [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/97

Summary

`TimeBoundToken::claim()` loops through all amounts to check for non empty. It could break once found.

Vulnerability Detail

The loop below can be optimized by breaking once found.

```
bool isNotEmpty;
for (uint256 j; j < amounts.length; ++j) {
    isNotEmpty = isNotEmpty || (amounts[j] > 0);
}
```

Impact

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/pull/75/files#diff-d2154bfdefe487f01ca5c19fd5f303e24439b3f52f9b264b6920632c3999c3a6R818-R819

Tool Used

Manual Review

Recommendation

Break once a positive amount is found.

Issue L-2: Inconsistent operation in `Float128x128::tryMultiply(Float128x128 a, uint256 b, bool roundUp)` [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/101

Summary

The result is not Float128x128, as it divides by Q128, and b is simple unsigned.

Vulnerability Detail

As can be seen, by dividing by `Q128`, it removes the decimal part, as only `a` has this component, `b` is a raw number.

```
/// @dev Multiplies a float with an unsigned integer and returns the result and a
 ↪  success flag.
/// @param a The float.
/// @param b The unsigned integer.
/// @param roundUp Whether to round up the result.
/// @return success True if the multiplication was successful, false otherwise.
/// @return result The result of the multiplication, or zero if the multiplication
 ↪  failed.
function tryMultiply(Float128x128 a, uint256 b, bool roundUp)
    internal
    pure
    returns (bool success, Float128x128 result)
{
    uint256 c;
    (success, c) = Float128x128.unwrap(a).tryMulDiv(b, Q128, roundUp);
    result = Float128x128.wrap(c);
}
```

Impact

Not used

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/pull/39/files/d284a70676e365f47d6da9eacfada88afd764c30#diff-698d795e088881c9720e91cc24d7ea0c9d2ecb2e452051c205ad4888a3e52593R247

Tool Used

Manual Review

Recommendation

Don't divide by Q128.

Issue L-3: Incorrect `day` check in `MonthBasedTimeBound::hasActivation` function [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/114

Vulnerability Detail

`hasActivation` function has incorrect `day` check logic (compare to `isActive` function):

```
    function hasActivation(
        ...
    ) external pure override returns (bool result) {
        ...

        result =
            ((monthsPassedPlusOne - monthBased.startedFrom) %
                monthBased.period ==
                0) &&
@>          (day == monthBased.day || daysInMonth < monthBased.day) &&
            (secondsOfTheDayPassed == monthBased.secondsOfTheDay);
    }

    function isActive(
        ...
    ) external pure override returns (bool result) {
        ...

        // test if the time is satisfied with period
        result =
            ((monthsPassedPlusOne - monthBased.startedFrom) %
                monthBased.period ==
                0) &&
@>          (day == monthBased.day || (monthBased.day > daysInMonth && day ==
↪   daysInMonth)) &&
            (secondsOfTheDayPassed == monthBased.secondsOfTheDay);

        ...
    }
```

Impact

Informal

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/main/Time-V3/src/TimeBound/Implementation/MonthBasedTimeBound/MonthBasedTimeBound.sol#L61-L65

Tool Used

Manual Review

https://github.com/sherlock-audit/2025-09-timeswap-v3/pull/57#discussion_r2352053522

Recommendation

Update `hasActivation` function similar to `isActive` function.

Issue L-4: Underflow in `iMonthBasedTimeBound::isActive` function [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/115

Vulnerability Detail

`iMonthBasedTimeBound.isActive` function performs division operations without checking if the numerator is positive, which can cause **underflow** and **revert**, instead of returning `false`:

```
    if (
        // testing if current month also satisfied
@>      ((currentMonthPassedPlusOne - monthBased.startedFrom) %
            monthBased.period ==
            0) &&
        (currentDay > monthBased.day) &&
        (currentDay == monthBased.day &&
            currentSecondsOfTheDayPassed > monthBased.secondsOfTheDay)
    ) {
        result =
@>          (monthsPassedPlusOne - currentMonthPassedPlusOne - 1) /
                monthBased.period <
            monthBased.maxConcurrent;
    } else {
        result =
@>          (monthsPassedPlusOne - currentMonthPassedPlusOne) /
                monthBased.period <
            monthBased.maxConcurrent;
    }
```

Line 124: `(currentMonthPassedPlusOne - monthBased.startedFrom) / monthBased.period`

- If `currentMonthPassedPlusOne < monthBased.startedFrom`, this underflows

Line 132: `(monthsPassedPlusOne - currentMonthPassedPlusOne - 1) / monthBased.period`

- If `monthsPassedPlusOne < currentMonthPassedPlusOne + 1`, this underflows

Line 137: `(monthsPassedPlusOne - currentMonthPassedPlusOne) / monthBased.period`

- If `monthsPassedPlusOne < currentMonthPassedPlusOne`, this underflows

Impact

`isActive` reverts instead of returning `false`

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/main/Time-V3/src/TimeBound/Implementation/MonthBasedTimeBound/MonthBasedTimeBound.sol#L122-L140

Tool Used

Manual Review

https://github.com/sherlock-audit/2025-09-timeswap-v3/pull/57#discussion_r2352055387

Recommendation

Add underflow check and returns `false`.

Issue L-5: Underflow on zero difference in `SecondBasedTimeBound::previous` function [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/116

Vulnerability Detail

When the `difference` is 0, `previous` function calculates `frequency = 0` but then performs `frequency--`, causing an **underflow revert**, instead of return `0`:

```
    function previous(
        bytes32 timeConfig,
        uint256 time
    ) external pure override returns (uint256 previousTime) {
        ...
        uint256 frequency = difference / secondBased.period;
@>      if (difference % secondBased.period == 0) {
@>          frequency--;
        }

        previousTime = secondBased.startedFrom + secondBased.period * frequency;
    }
```

Impact

View function revert so informal

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/main/Time-V3/src/TimeBound/Implementation/SecondBasedTimeBound/SecondBasedTimeBound.sol#L121-L124

Tool Used

Manual Review

https://github.com/sherlock-audit/2025-09-timeswap-v3/pull/63#discussion_r2352584744

Recommendation

```
--      if (!success) {
++      if (!success || difference == 0) {
            return 0;
        }
```

Issue L-6: `TimeswapV3::createMarket()` can be called with `baseTokenId=0` [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/117

Vulnerability Detail

The validation logic could be improved and made consistent with the `createBin` function:

```
    function createMarket(
        ...
    ) external override returns (uint256 marketId) {
        ...

@>      if (baseTokenId > timeBoundToken.latestTokenId()) {
            revert TimeswapV3Error.BaseTokenIdIsInvalid(baseTokenId);
        }
        ...
    }

    function createBin(
        uint256 marketId,
        uint256 maturity,
        int24 bidTick,
        int24 askTick
    ) external override returns (uint256 liquidityTokenId) {
@>      if (marketId == 0 || marketId > marketIdCounter) {
            revert TimeswapV3Error.MarketNotCreated(marketId);
        }
        ...
    }
```

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/main/Time-V3/src/TimeswapV3/TimeswapV3.sol#L364-L376

Tool Used

Manual Review

https://github.com/sherlock-audit/2025-09-timeswap-v3/pull/84#discussion_r2382659043

Recommendation

```
-       if (baseTokenId > timeBoundToken.latestTokenId()) {
+       if (baseTokenId == 0 || baseTokenId > timeBoundToken.latestTokenId()) {
```

Issue L-7: Improvement of `DeltaMath::toDelta` function [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/118

Vulnerability Detail

If the difference `b - a` or `a - b` exceeds `type(int256).max`, the cast will silently wrap around without reverting.

```
    function toDelta(
        uint256 a,
        uint256 b
    ) internal pure returns (int256 result) {
@>      result = a <= b ? int256(b - a) : -int256(a - b);
    }
```

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/main/Time-V3/src/Primitive/DeltaMath.sol#L53-L58

Tool Used

Manual Review

https://github.com/sherlock-audit/2025-09-timeswap-v3/pull/38#discussion_r2392072970

Recommendation

Just like `addDelta()` and `subtractDelta()` handle edge cases (like `type(int256).min`), `toDelta()` could be updated to handle it too:

```
function toDelta(uint256 a, uint256 b) internal pure returns (int256 result) {
    if (a <= b) {
        uint256 diff = b - a;
        require(diff <= uint256(type(int256).max), "DeltaMath: overflow");
        result = int256(diff);
    } else {
        uint256 diff = a - b;
        require(diff <= uint256(type(int256).max), "DeltaMath: overflow");
        result = -int256(diff);
```

```
    }
}
```

Issue L-8: Improvement of `liquidityParameter.interest` calculation in `TimeswapV3AddLiquidityMidware::calculateAddLiquidity` [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/119

Vulnerability Detail

It's better to set `liquidityParameter.interest` to the minimum of `liquidityParameter.interest` and `maxInterest` at the end, since `liquidityParameter.interest` can exceed `maxInterest` (ofc, `liquidityParameter.interest - maxInterest` could be `1 wei` at max):

```
liquidityParameter.interest = binInterest.mulDiv(
    approvedLiquidityGivenPrincipal,
    binLiquidity,
    Math.Rounding.Ceil
);
```

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/main/Time-V3/src/Midware/TimeswapV3AddLiquidityMidware/TimeswapV3AddLiquidityMidware.sol#L198-L205

Tool Used

Manual Review

https://github.com/sherlock-audit/2025-09-timeswap-v3/pull/11#discussion_r2413862543

Recommendation

```
    liquidityParameter.interest = binInterest.mulDiv(
        approvedLiquidityGivenPrincipal,
        binLiquidity,
        Math.Rounding.Ceil
--  );
++  ).min(
++      maxInterest
++  );
```

A similar logic can be applied to `liquidityParameter.principal` and `maxPrincipal` in the `else` condition.

Issue L-9: `diffYields[i].baseYield` is set in TimeBoundLiquidityMining even though it is non rebasing [RESOLVED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/120

Summary

`TimeBoundLiquidityMining::beforePrepareHistory()` sets `diffYields[i].baseYield`, but the token is non rebasing, so it will revert.

Vulnerability Detail

Tokens created by the `TimeBoundLiquidityMining` have `rebasingEnabled == false`, so there can't be BaseYield.

Impact

The underlying token, is likely not supposed to be rebasing, in this case having no impact.

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/pull/69/files#diff-4467cdfdaca66633b40b923e80fae5ed9a649ccd0cc8560b7546ab3f45461933R618

Tool Used

Manual Review

Recommendation

Either allow rebasing tokens on creation and support base yield or just remove that line.

Issue L-10: Lack of valid `ratePolicy` check when creating `TimswapV3` market [ACKNOWLEDGED]

Source: https://github.com/sherlock-audit/2025-09-timeswap-v3/issues/125

This issue has been acknowledged by the team but won't be fixed at this time.

Vulnerability Detail

The user can pass in any `ratePolicy` when creating a `TimeswapV3` market, and so the `ratePolicy.isValid` is not effectice.

from `RatePolicy.sol` docs:

> Without a RatePolicy, anyone could post nonsense rates that break pricing or allow unfair trades. RatePolicy ensures every bin is aligned with the protocol's rate configuration.

and one baseTokenId can have a few market, (a few valid market but a lot of invalid / malicious market)

the recommendation is whiteliste the `ratePolicy`

Impact

Malicious market with invalid `ratePolicy` can be created.

Code Snippet

https://github.com/sherlock-audit/2025-09-timeswap-v3/blob/149bab1c27f1008a6584f86eda12662d7da855c8/Time-V3/src/TimeswapV3/TimeswapV3.sol#L364-L370

Recommendation

Maintain a list of whitelisted `ratePolicy` address.

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.