SHERLOCK

Security Review For Autopilot

Collaborative Audit Prepared For:	Autopilot
Lead Security Expert(s):	TessKimy
Date Audited:	July 9 - July 12, 2025
Final Commit:	b3b3e6a

Introduction

This security review focused on validating the security and correctness of the PermanentLocksPool system, particularly around voting strategies, reward distribution, and epoch synchronization. It also reviewed the upgradeable components–Swapper and DepositValidator–for safe integration and validation logic.

Scope

Repository: aeroclub-finance/autopilot-contracts

Audited Commit: ed4c8d8cfd888b14c07d23de6bc336f5e12a7121

Final Commit: b3b3e6a19d99e98777b0345868e42b30f938bd26

Files:

- contracts/aerodrome/IveNFT.sol
- contracts/autopilot/DepositValidatorV1.sol
- contracts/autopilot/IDepositValidator.sol
- contracts/autopilot/IPermanentLocksPoolV1.sol
- contracts/autopilot/PermanentLocksPoolV1.sol
- contracts/autopilot/RewardsVault.sol
- contracts/autopilot/SwapperV1.sol

Final Commit Hash

b3b3e6a19d99e98777b0345868e42b30f938bd26

Findings

Each issue has an assigned severity:

- 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.
- High issues are directly exploitable security vulnerabilities that need to be fixed.
- 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
1	0	11

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue H-1: Improper Voting Power Update in claim RebaseReward Enables Reward Manipulation

Source: https://github.com/sherlock-audit/2025-07-autopilot-july-9th/issues/8

Summary

The `claimRebaseReward` function fails to account for pending rewards tied to an NFT's voting power before that power is updated. This oversight allows to manipulate the reward distribution system. By updating the NFT's voting power *before* claiming rewards, the system uses outdated power values in its global reward rate calculation, leading to imbalanced and exploitable payouts.

Vulnerability Detail

The vulnerability lies in the sequence of operations inside the `claimRebaseReward` function. Specifically, it updates the voting power of NFTs before claiming any pending rewards tied to their previous voting power. Since the reward system relies on the total voting power to compute the global reward rate, modifying this value prematurely introduces inconsistencies.

The formula used for reward rate calculation is:

$$\text{rewardRate} = \frac{\text{reward} \times \text{scale}}{\text{totalVotingPower}}$$

If `totalVotingPower` is altered before the associated NFT’s reward is claimed, the numerator (reward) is distributed incorrectly.

1. Accumulating pending rewards on NFTs with lower voting power.
2. Calling `claimRebaseReward`, which increases voting power *before* claiming the reward.
3. Exploiting the fact that the global reward rate still reflects the lower `totalVotingPower` (from before the increase).
4. Receiving more reward than justified due to inflated individual power and outdated global power.

Impact

The issue allows for **extraction of unearned rewards**. Since the total voting power used in the reward rate denominator is outdated when rewards are calculated, an attacker can strategically amplify their earnings. This distorts the fairness of the distribution model.

Code Snippet

https://github.com/sherlock-audit/2025-07-autopilot-july-9th/blob/9dfa83824a97f6c982b1d089eeea2fe66f3704f3/autopilot-contracts/contracts/autopilot/PermanentLocksPoolV1.sol#L502-L509

https://github.com/sherlock-audit/2025-07-autopilot-july-9th/blob/9dfa83824a97f6c982b1d089eeea2fe66f3704f3/autopilot-contracts/contracts/autopilot/PermanentLocksPoolV1.sol#L898-L900

Tool Used

Manual Review

Recommendation

Claim pending rewards for given NFT in `claimRebaseRewards` function before updating voting power.

Discussion

Sergey988

https://github.com/aeroclub-finance/autopilot-contracts/blob/d81d2c282e6e0f5d982800d070d120e97f284874/contracts/autopilot/PermanentLocksPoolV1.sol allowbreak #L515C1-L519C8

We decided to avoid direct claims during rebase, so we just “postponed” the rewards till actual claim. We added `postponed_rewards` field to `LockInfo` which is incrementing with every rebase and clearing every claim.

Issue L-1: Incorrect Execution Order in deposit Function Prevents Valid Permanent Lock Upgrade

Source: https://github.com/sherlock-audit/2025-07-autopilot-july-9th/issues/9

Summary

The `deposit` function attempts to automatically convert eligible NFTs into permanently locked versions during deposit. However, due to incorrect execution order, the conversion to permanent lock occurs *after* a deposit validation check. In cases where an NFT has decayed in value over time, this check may fail because it validates against the current (lower) value instead of the upgraded permanent lock value. As a result, deposits that should succeed revert unexpectedly.

Vulnerability Detail

The function first calls `deposit_validator.validateDepositOrFail(...)` before converting the NFT into a permanently locked version. However, only permanently locked NFTs receive the full voting power restoration (typically to their highest historical value, per Aerodrome's mechanism).

This causes a problem when:

- An NFT is decaying (not permanently locked).
- Its current voting power falls below the minimum required by `deposit_validator`.
- The contract intends to upgrade it to permanent lock during deposit.
- But the validation check (which occurs *before* the upgrade) fails due to its decayed power.

This is a logic ordering flaw.

Impact

This flaw introduces an edge-case **denial-of-deposit** for users with time-decayed NFTs that would otherwise qualify upon permanent locking.

Code Snippet

https://github.com/sherlock-audit/2025-07-autopilot-july-9th/blob/9dfa83824a97f6c982b1d089eeea2fe66f3704f3/autopilot-contracts/contracts/autopilot/PermanentLocksPoolV1.sol#L300-L314

Tool Used

Manual Review

Recommendation

Corrected Order

```
if (nft_locks_contract.voted(_lock_id)) {
  voter_contract.reset(_lock_id);
}

if (!lock.isPermanent) {
  nft_locks_contract.lockPermanent(_lock_id);
}

if (address(deposit_validator) != address(0)) {
  deposit_validator.validateDepositOrFail(
    nft_locks_contract,
    _lock_id,
    msg.sender
  );
}
```

This ensures the NFT is permanently locked (and thus restored in value) *before* validation occurs.

Discussion

Sergey988

https://github.com/aeroclub-finance/autopilot-contracts/blob/d81d2c282e6e0f5d982800d070d120e97f284874/contracts/autopilot/PermanentLocksPoolV1.sol
allowbreak #L318-L333

Fixed exactly as in your recommendation

Issue L-2: voteWithNfts Function Susceptible to Unintended Reverts Due to Outdated Snapshot State

Source: https://github.com/sherlock-audit/2025-07-autopilot-july-9th/issues/10

Summary

The `voteWithNfts` function checks whether the system is within a special window using a `last_snapshot_id` variable. However, this variable can become outdated due to protocol inactivity. When outdated, the `_isInSpecialWindowOrFail(last_snapshot_id)` check may incorrectly revert, preventing otherwise valid votes from being cast. To prevent this denial-of-service condition, the function should proactively call `_emergencySnapshot()` before this check to ensure `last_snapshot_id` is current.

Vulnerability Detail

The `last_snapshot_id` is assumed to be fresh, but this assumption fails in periods of inactivity and whenever snapshot reward call is missed in current epoch. If an autopilot bot tries to vote during such a period, the system may falsely consider them to be outside the ”special window”, causing a revert.

Impact

This bug creates a **low-severity denial-of-service** for permitted operators attempting to vote.

Code Snippet

https://github.com/sherlock-audit/2025-07-autopilot-july-9th/blob/9dfa83824a97f6c982b1d089eeea2fe66f3704f3/autopilot-contracts/contracts/autopilot/PermanentLocksPoolV1.sol#L417-L421

Tool Used

Manual Review

Recommendation

Precede the window check with an emergency snapshot update:

```
_emergencySnapshot(); // Refresh state
_isInSpecialWindowOrFail(last_snapshot_id);
```

This ensures that voting is always evaluated against the most recent snapshot, avoiding unnecessary transaction failures.

Discussion

Sergey988

https://github.com/aeroclub-finance/autopilot-contracts/blob/d81d2c282e6e0f5d982800d070d120e97f284874/contracts/autopilot/PermanentLocksPoolV1.sol allowbreak #L440-L443

Fixed exactly as in your recommendation

Issue L-3: claimRebaseReward can be DoSed due to stale activePeriod in Aerodrome Minter

Source: https://github.com/sherlock-audit/2025-07-autopilot-july-9th/issues/11

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

Summary

The `claimRebaseReward` function depends on Aerodrome's `reward_distributor` contract, which internally checks whether the minter's `activePeriod` is current. If `activePeriod()` is stale, the claim reverts with an `UpdatePeriod()` error. This creates a denial-of-service (DoS) vector: unless an external party calls `updatePeriod()` on the minter contract beforehand, `claimRebaseReward` will always revert.

Vulnerability Detail

The call chain from `claimRebaseReward` leads to Aerodrome's reward distribution logic, which includes this guard clause:

```
if (IMinter(minter).activePeriod() < ((block.timestamp / WEEK) * WEEK)) revert
↪  UpdatePeriod();
```

- `activePeriod()` returns the last reward period timestamp.
- If it is older than the current period (rounded down to the nearest `WEEK`), the claim call reverts.

Because `activePeriod` is not automatically updated, this check prevents reward claiming until someone explicitly calls:

```
IMinter(minter).updatePeriod();
```

Impact

It's an edge case scenario that reverts transaction until the update (DoS)

Code Snippet

https://github.com/sherlock-audit/2025-07-autopilot-july-9th/blob/9dfa83824a97f6c982b1d089eeea2fe66f3704f3/autopilot-contracts/contracts/autopilot/PermanentLocksPoolV1.sol#L502

Tool Used

Manual Review

Recommendation

Before calling claim function check the active period and call update period function if it's not up-to-date.

Discussion

Sergey988

Nothing critical happens if it will not work. We can call it any time during the week, outside of special window. But we will check this from the bot side to avoid reverts and maybe postpone tx.

Issue L-4: Inactive Epochs Can Lead to Incorrect Reward Distribution Due to Missed Snapshot

Source: https://github.com/sherlock-audit/2025-07-autopilot-july-9th/issues/12

Summary

An edge case occurs when a user deposits an NFT during the special window at the end of epoch 0. This deposit is intended to be counted in epoch 1. However, if the `snapshotReward` function is not triggered during epoch 1 (either by the autopilot bot or via an emergency snapshot), the snapshot logic will incorrectly calculate rewards in the next epoch, using outdated tracking data from the last recorded snapshot.

> Note: Given example with epoch 0 and 1 is just an example, not specific for epoch 0 and 1

Vulnerability Detail

Reward distribution relies on the tracked weight recorded in the last snapshot:

```
if (total_tracked_weight[last_snapshot_id] > 0) {
    uint256 reward_scaled = (reward_amount * SCALE) /
 ↪  total_tracked_weight[last_snapshot_id];
    acc_reward_scaled += reward_scaled;
}
```

- `last_snapshot_id` is not updated if no snapshot is taken during epoch 1.
- Therefore, any user deposits made for epoch 1 (special window of epoch 0) are **not** included in the tracked weight used to calculate the reward coefficient (`reward_scaled`) in the next epoch.
- As a result, the computed `reward_scaled` value is inflated, since it's based on a smaller `total_tracked_weight`, and rewards are distributed disproportionately.

> Note: There is also another problem that if tracked_weight is equal to 0, it still sends the funds to reward vault which will lock it inside to contract. Fix will be given in recommendation

Impact

Users who should be eligible for rewards (e.g., those who deposited during the transition window) are not properly accounted and it will cause inflated reward calculation.

Code Snippet

https://github.com/sherlock-audit/2025-07-autopilot-july-9th/blob/9dfa83824a97f6c982b1d089eeea2fe66f3704f3/autopilot-contracts/contracts/autopilot/PermanentLocksPoolV1.sol#L589-L595

Tool Used

Manual Review

Recommendation

Following code solves the problem:

```
if(current_epoch > last_snapshot_id + 1) {
  if(total_tracked_weight[last_snapshot_id] + total_tracked_weight[last_snapshot_id
↪   + 1] > 0) {
    uint256 reward_scaled = (reward_amount * SCALE) /
↪   (total_tracked_weight[last_snapshot_id] + total_tracked_weight[last_snapshot_id
↪   + 1]);
    acc_reward_scaled += reward_scaled;
    SafeERC20.forceApprove(rewards_token, address(rewards_vault), reward_amount);
    rewards_vault.deposit(rewards_token, reward_amount);
  }
} else {
  if(total_tracked_weight[last_snapshot_id] > 0) {
    uint256 reward_scaled = (reward_amount * SCALE) /
↪   total_tracked_weight[last_snapshot_id];
    acc_reward_scaled += reward_scaled;
    SafeERC20.forceApprove(rewards_token, address(rewards_vault), reward_amount);
    rewards_vault.deposit(rewards_token, reward_amount);
  }
}
```

Discussion

Sergey988

Fixed by refactoring `acc_reward_scaled`. Now we are `snapshoting acc_reward_scaled` state for each epoch. We added `acc_reward_scaled_per_epoch` global variable which is a mapping.

Issue L-5: Special Window Duration Doesn’t Work as Documented Behavior

Source: https://github.com/sherlock-audit/2025-07-autopilot-july-9th/issues/13

Summary

The protocol documentation states that the ”special window” for deposits lasts a fixed duration (e.g., `X` minutes). However, in practice, this window ends immediately when the bot publishes a reward snapshot regardless of the actual elapsed time. This introduces a mismatch between implementation and documentation, which could confuse integrators or users relying on a predictable timing model.

Vulnerability Detail

The function that publishes reward snapshots also implicitly closes the special window because last snapshot id will be increased after execution and special window will be shifted to next one. Thus, the window's actual lifespan is tied to when the bot executes `snapshotReward`, not the constant duration described in the documentation.

Impact

Expected behavior based on documentation does not match contract logic

Code Snippet

https://github.com/sherlock-audit/2025-07-autopilot-july-9th/blob/9dfa83824a97f6c982b1d089eeea2fe66f3704f3/autopilot-contracts/contracts/autopilot/PermanentLocksPoolV1.sol#L606

https://github.com/sherlock-audit/2025-07-autopilot-july-9th/blob/9dfa83824a97f6c982b1d089eeea2fe66f3704f3/autopilot-contracts/contracts/autopilot/PermanentLocksPoolV1.sol#L347

Tool Used

Manual Review

Recommendation

Track block.timestamp instead of last snapshot id for special windows

Discussion

Sergey988

Fixed documentation in this commit

Issue L-6: Emergency Withdraw Functions Bypass Special Window Restriction if Snapshot Is Outdated

Source: https://github.com/sherlock-audit/2025-07-autopilot-july-9th/issues/14

Summary

The functions `emergencyWithdrawFromRewardsVault` and `emergencyWithdrawFromLocksVault` include a check to prevent execution during the special window:

```
_isNotInSpecialWindowOrFail(last_snapshot_id);
```

However, if `snapshotReward` has not been called in the current epoch, `last_snapshot_id` remains outdated. As a result, the special window check passes incorrectly, allowing emergency withdrawals during an active special window.

Impact

Requires `onlyOwner` access and does not present a direct risk to user funds, but violates design constraints.

Code Snippet

https://github.com/sherlock-audit/2025-07-autopilot-july-9th/blob/9dfa83824a97f6c982b1d089eeea2fe66f3704f3/autopilot-contracts/contracts/autopilot/PermanentLocksPoolV1.sol#L721-L725

https://github.com/sherlock-audit/2025-07-autopilot-july-9th/blob/9dfa83824a97f6c982b1d089eeea2fe66f3704f3/autopilot-contracts/contracts/autopilot/PermanentLocksPoolV1.sol#L736-L741

Tool Used

Manual Review

Recommendation

Call `_emergencySnapshot` before checking special window

Discussion

Sergey988

https://github.com/aeroclub-finance/autopilot-contracts/blob/d81d2c282e6e0f5d982800d070d120e97f284874/contracts/autopilot/PermanentLocksPoolV1.sol
allowbreak #L765-L792

Fixed exactly as in your recommendation

Issue L-7: Automation-Facing Functions Lack Fault Tolerance, Causing Preventable DoS

Source: https://github.com/sherlock-audit/2025-07-autopilot-july-9th/issues/15

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

Summary

Several core functions designed for off-chain automation such as `batchSwapMultiHop`, `claimRebaseRewards` perform batch operations using `for` loops without error isolation. If a single item in the batch fails (e.g., an NFT has been withdrawn or a swap fails), the entire transaction reverts, halting all other unrelated operations. This undermines the protocol's gas efficiency and reliability, both of which are critical to the economic model of Autopilot.

Vulnerability Detail

Each of the functions below contains a `for` loop that processes a list of user assets or instructions. However, the loop lacks resilience against individual errors:

batchSwapMultiHop

- Executes swaps in a loop.
- If a single swap fails, the entire function reverts.
- **Recommendation**: Use `try-catch` around each individual swap to allow the rest to proceed.

> Note: Swap can fail due to slippage check with high likelihood because there can be many swap command

claimRebaseRewards

- Iterates over NFT token IDs.
- If any NFT has been withdrawn and ownership check fails, the function reverts.
- **Recommendation**: Add ownership check and skip failed NFTs using `continue`.

Impact

- Automation bot executions may fail unpredictably.
- Wasted gas on failed transactions.

- Blocks protocol scalability in automated environments.

Tool Used

Manual Review

Discussion

Sergey988

We are handling this from the bot (offchain) side by simulating voting just before actual voting

Also when swapping tokens (the most possibly bugged stage) we are simulating which tokens are exactly causing the revert and removing it from the batch. After simulation done we receive a working txs and we know exactly which tokens were not swapped so we will try to swap it on next epoch.

Skipping this issue

Issue L-8: Potential Precision Loss When Snapshotting Low Reward Amounts

Source: https://github.com/sherlock-audit/2025-07-autopilot-july-9th/issues/16

Summary

The reward distribution logic may experience precision loss when a very small reward amount is snapshotted into a pool with high total value locked (TVL). Since rewards are in USDC (6 decimals) and the share values are scaled to 18 decimals, the computed per-share reward (`reward_scaled`) may truncate to zero, resulting in no effective reward distribution.

Vulnerability Detail

Reward calculation is performed using the formula:

```
uint256 reward_scaled = (reward_amount * SCALE) /
↪  total_tracked_weight[last_snapshot_id];
```

Where:

- `SCALE = 1e18`
- `reward_amount` is in 6-decimal USDC
- `total_tracked_weight` (TVL) is based on 18-decimal AERO

Example Scenario:

- `total_tracked_weight = 1e24` (□ 1,000,000 AERO)
- `reward_amount = 9e5` (0.9 USDC)
- Resulting `reward_scaled = 9e5 * 1e18 / 1e24 = 0` (rounded)

If `reward_amount` is too small relative to TVL, `reward_scaled` may round down to 0 due to integer division. This causes no rewards to be distributed despite a snapshot being taken.

> Note: This is very low likelihood situation because less than $1 reward for 1M AERO TVL is not a realistic scenario. If off-chain bots correctly takes the rewards and publish them as snapshot to pool, it's impossible to be equal to 0. (Maybe if AERO price goes down to 0.001, this scenario can be possible)

Impact

Deposited amount will be locked in reward vault contract

Tool Used

Manual Review

Recommendation

Increase `SCALE` value to 1e30 in order to calculate it in 1e18 precision

Discussion

Sergey988

This will require too many changes from offchain and onchain part, unfortunately will be fixed only in the next Version 2 Contract.

Skipping this issue

Ipetroulakis

Fixed in a previous commit.

Issue L-9: Unnecessary State Write When lock_payout == 0

Source: https://github.com/sherlock-audit/2025-07-autopilot-july-9th/issues/17

Summary

In the reward claim logic, the contract updates the `lock_info.reward_scaled_start` baseline even when the computed `lock_payout` is zero. Since no reward is being distributed in this case, this write operation is redundant.

Vulnerability Detail

The following code updates the user's reward baseline unconditionally when `delta_acc > 0`, even if no payout occurs:

```
if (delta_acc > 0) {
  uint256 lock_payout = (lock_weight * delta_acc) / SCALE;

  // Unconditional update
  lock_info.reward_scaled_start = acc_reward_scaled;

  if (lock_payout > 0) {
    rewards_vault.withdraw(rewards_token, msg.sender, lock_payout);
    emit Claim(msg.sender, lock_payout);
  }
}
```

In scenarios where `lock_payout == 0`, updating `reward_scaled_start` has no practical effect. This is very low likelihood situation. It happens only when user has really low amount of voting power. We can also count it as user mistake.

Impact

Impact is very low because it can only happen when voting power of user is very low.

Tool Used

Manual Review

Recommendation

Move the baseline update inside the conditional block:

```
if (delta_acc > 0) {
  uint256 lock_payout = (lock_weight * delta_acc) / SCALE;

  if (lock_payout > 0) {
    lock_info.reward_scaled_start = acc_reward_scaled;
    rewards_vault.withdraw(rewards_token, msg.sender, lock_payout);
    emit Claim(msg.sender, lock_payout);
  }
}
```

This ensures state is only mutated when necessary.

Discussion

Sergey988

https://github.com/aeroclub-finance/autopilot-contracts/blob/d81d2c282e6e0f5d982800d070d120e97f284874/contracts/autopilot/PermanentLocksPoolV1.sol allowbreak #L936C31-L948

Fixed in latest version

Issue L-10: Missed Snapshot Prevents Reward Claim for Users Who Withdraw Before Recovery

Source: https://github.com/sherlock-audit/2025-07-autopilot-july-9th/issues/18

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

Summary

In the Autopilot system, if the automation bot fails to call `snapshotReward` during the special window, the fallback mechanism `emergencySnapshot` is used to populate the snapshot for the missed epoch. This ensures that the system can continue operating into the next epoch. However, this mechanism fails to account for users who withdraw their locks before the next `snapshotReward` call causing them to permanently lose access to their entitled rewards.

Vulnerability Detail

When `snapshotReward` is missed:

- `emergencySnapshot` copies the previous epoch's tracked state into the current snapshot ID.

If a user:

1. Deposits a lock in epoch **N**
2. The bot misses the snapshot at the start of epoch **N+1**
3. The user **withdraws** the lock and he claimed 0 reward from the system
4. In the next epoch, reward is distributed for all previous users
5. Our user couldn't take any reward but he locked his lock at least 1 epoch.

Impact

Maybe we can count it as user mistake but we can't deny that user couldn't get any reward for his 1 week lock period.

Tool Used

Manual Review

Recommendation

Fix is not trivial, fix should be discussed because it needs many change in the codebase.

Discussion

Sergey988

https://github.com/aeroclub-finance/autopilot-contracts/blob/d81d2c282e6e0f5d982800d070d120e97f284874/contracts/autopilot/PermanentLocksPoolV1.sol allowbreak #L83-L85

Fixed by refactoring `acc_reward_scaled`. Now we are snapshoting `acc_reward_scaled` state for each epoch. We added `acc_reward_scaled_per_epoch` global variable which is a mapping.

Issue L-11: emergencySnapshot Can Be Exploited to DoS snapshotReward in Edge Case Epochs

Source: https://github.com/sherlock-audit/2025-07-autopilot-july-9th/issues/19

Summary

The `emergencySnapshot` function is designed to recover from missed `snapshotReward` calls by backfilling snapshot data using the last known state. However, in edge-case scenarios involving prolonged inactivity, a malicious actor can abuse `emergencySnapshot` to deliberately block the next legitimate `snapshotReward` call. This leads to a denial-of-service condition in the automated snapshot flow.

Vulnerability Detail

The `emergencySnapshot` function checks whether the system is currently in the *special window* of the **last snapshot ID**, and skips execution if so. But when `last_snapshot_id` is stale due to inactivity, this logic becomes exploitable:

Scenario:

1. **Epoch N ends** with valid deposits.
2. **Epoch N+1** passes without activity:
   - Bot fails to call `snapshotReward`.
   - No `emergencySnapshot` is triggered during this epoch.
3. Now in **Epoch N+2**, and **special window of Epoch N** is active (because `last_snapshot_id = N`).
4. **Malicious actor** calls `emergencySnapshot`, which:
   - Incorrectly assumes it's safe to update.
   - Proceeds to increment `last_snapshot_id` and creates a copy snapshot.
5. When the **automation bot** attempts to call `snapshotReward`, it reverts:
   - Because snapshot ID has already been updated.
   - Special window conditions are no longer met.

Impact

- **Denial of Service (DoS)**: Automation bot cannot complete `snapshotReward` for the current epoch.

Code Snippet

https://github.com/sherlock-audit/2025-07-autopilot-july-9th/blob/9dfa83824a97f6c982b1d089eeea2fe66f3704f3/autopilot-contracts/contracts/autopilot/PermanentLocksPoolV1.sol#L622-L625

Tool Used

Manual Review

Recommendation

Introduce a stricter validation in `emergencySnapshot` to ensure track the actual epoch (via timestamp) and prevent `emergencySnapshot` from overwriting the expected reward slot

Or

Ensure every epoch at least 1 time `_emergencySnapshot` is called

Discussion

Sergey988

I think this is not an issue. He cannot exploit this. So if he do emergency in N+2, new window will be in N+3, during window of N+3 he cannot call emergency, so our bots can work normally. He will need to wait till window ends.

This situation itself is not affecting user funds, anyway the bot is not working.

Ipetroulakis

`isInSpecialWindowOrFail()` function is added to `snapshotRewards`. Given scenario won't be followed by automation bot.

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.