



Pashov Audit Group

Nucleus Security Review

July 29th 2025 - July 30th 2025



Contents

1. About Pashov Audit Group	3
2. Disclaimer	3
3. Risk Classification	3
4. About Nucleus	4
5. Executive Summary	4
7. Findings	5
Low findings	6
[L-01] Decoder silently accepts unknown <code>actionID</code>	6
[L-02] Salt-counter can be exhausted	6
[L-03] Incorrect function name in <code>CoreWriterDecoderAndSanitizer</code> affects vault	6



1. About Pashov Audit Group

Pashov Audit Group consists of 40+ freelance security researchers, who are well proven in the space - most have earned over \$100k in public contest rewards, are multi-time champions or have truly excelled in audits with us. We only work with proven and motivated talent.

With over 300 security audits completed — uncovering and helping patch thousands of vulnerabilities — the group strives to create the absolute very best audit journey possible. While 100% security is never possible to guarantee, we do guarantee you our team's best efforts for your project.

Check out our previous work [here](#) or reach out on Twitter [@pashovkrum](#).

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact

- **High** - leads to a significant material loss of assets in the protocol or significantly harms a group of users
- **Medium** - leads to a moderate material loss of assets in the protocol or moderately harms a group of users
- **Low** - leads to a minor material loss of assets in the protocol or harms a small group of users

Likelihood

- **High** - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost
- **Medium** - only a conditionally incentivized attack vector, but still relatively likely
- **Low** - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive



4. About Nucleus

Nucleus WHLP is an extensible smart contract system for automating Hyperliquid HLP Vault fund management via the CoreWriter precompile, eliminating manual multisig operations and improving capital efficiency. It orchestrates deterministic account creation, spot/perp transfers, and vault deposits/withdrawals while ensuring funds never leave controlled vault–account–HLP pathways.

5. Executive Summary

A time-boxed security review of the [Ion-Protocol/nucleus-boring-vault](#) repository was done by Pashov Audit Group, during which Pashov Audit Group engaged to review **Nucleus**. A total of 3 issues were uncovered.

Protocol Summary

Project Name	Nucleus
Protocol Type	Vault Management
Timeline	July 29th 2025 - July 30th 2025

Review commit hash:

- [c4fc29b6663296f07823766f5b864cb5a66431ab](#)
(Ion-Protocol/nucleus-boring-vault)

Fixes review commit hash:

- [2ff06ca5e138701c711a4d62ca5ec4e291c06176](#)
(Ion-Protocol/nucleus-boring-vault)

Scope

[CoreWriterDecoderAndSanitizer.sol](#) [WHLPDecoderAndSanitizer.sol](#) [HLPAccount.sol](#)
[HLPController.sol](#)



6. Findings

Findings count

Severity	Amount
Low	3
Total findings	3

Summary of findings

ID	Title	Severity	Status
[L-01]	Decoder silently accepts unknown <code>actionID</code>	Low	Resolved
[L-02]	Salt-counter can be exhausted	Low	Resolved
[L-03]	Incorrect function name in <code>CoreWriterDecoderAndSanitizer</code> affects vault	Low	Resolved



Low findings

[L-01] Decoder silently accepts unknown `actionID`

The `SendRawAction` function reaches its end without a final `else revert`, returning empty bytes when an unrecognised `actionID` is supplied. The Merkle tree then hashes an empty payload, potentially admitting unintended `CoreWriter` actions.

It is recommended to add an explicit revert `InvalidActionID()` (or equivalent custom error) when `actionID` does not match a supported branch, ensuring that any unknown or future action fails.

[L-02] Salt-counter can be exhausted

The controller advances one unit of the salt value used every time a new account is deployed. After a certain number of deployments, the increment wraps into the high-order bytes that encode the address(`this`), producing a salt that would be able to overflow and revert every next call.

Even though such an event is unlikely, it is recommended to increment the salt value inside `unchecked` code blocks so overflows are able to wrap-around without causing transactions to revert.

[L-03] Incorrect function name in `CoreWriterDecoderAndSanitizer` affects vault

The `CoreWriterDecoderAndSanitizer` contract is intended to support Merkle-verified interactions with the Hyperliquid `coreWriter` precompile by decoding and sanitizing calls made through `sendRawAction(bytes)`. This mechanism is crucial for secure and permissioned interactions originating from the BoringVault via the merkle verification system.

However, the decoder function is incorrectly named:

```
function SendRawAction(bytes calldata data) external view returns (bytes memory);
```

This does not match the function signature expected by the Merkle call routing system, which dynamically dispatches to a decoder function that must exactly match the target function being called.

This means that any attempt to call `sendRawAction(bytes)` via the vault will fail Merkle verification, as the decoder does not implement the expected interface. The system will revert with `BaseDecoderAndSanitizer__FunctionNotImplemented`.



This breaks the vault's ability to execute **CoreWriter interactions**, including those required for executing `SpotSend` and `LimitOrder` actions — both of which are essential to the WHLP withdrawal path:

"The vault then places a buy order for USDHL and we bridge back to the EVM to service our withdraw."

Since the decoder fails to recognize and permit the underlying `sendRawAction(...)` call, **vault-initiated withdrawals will fail entirely**, resulting in stuck funds.

Proof of Concept:

```
function _buildCoreWriterCall(
    bytes memory encodedAction,
    bytes1 actionID
) internal returns (bytes memory) {
    bytes memory data = new bytes(4 + encodedAction.length);
    data[0] = 0x01;
    data[1] = 0x00;
    data[2] = 0x00;
    data[3] = actionID;
    for (uint256 i = 0; i < encodedAction.length; i++) {
        data[4 + i] = encodedAction[i];
    }
    return abi.encodeWithSignature("sendRawAction(bytes)", data);
}

function test_sendRawAction_decodeAndSanitize() public {
    WHLPDecoderAndSanitizer decoderAndSanitizer = new WHLPDecoderAndSanitizer(
        address(0),
        address(0)
    );

    bytes memory decodeAndSanitizeCallData = _buildCoreWriterCall(
        // We don't really care about the values here, just the structure
        abi.encode(
            uint32(0),
            true,
            uint64(0),
            uint64(0),
            true,
            uint8(0),
            uint128(0)
        ),
        0x01
    );

    vm.expectRevert(
        abi.encodeWithSelector(
            BaseDecoderAndSanitizer
                .BaseDecoderAndSanitizer__FunctionNotImplemented
                .selector,
            decodeAndSanitizeCallData
        )
    );
    // demos the call made in `ManagerWithMerkleVerification::_verifyCallData`
```



```
bytes memory packedArgumentAddresses = abi.decode(
    address(decoderAndSanitizer).functionStaticCall(
        decodeAndSanitizeCallData
    ),
    (bytes)
);
}
```

Recommendations

Rename the decoder function to match the exact interface name used by `coreWriter` :

```
function sendRawAction(
    bytes calldata data
) external view override returns (bytes memory addressesFound) {
    ...
}
```

This change will allow the Merkle call validation system to successfully resolve the decoder and properly authorize `coreWriter` interactions through the vault.