SHERLOCK

Security Review For Nuva Labs

Collaborative Audit Prepared For: **Nuva Labs**
Lead Security Expert(s): **defsec**
Kirkeelee
Date Audited: **April 2 - April 6, 2026**

Introduction

NUVA unlocks global access to the world's best real-world assets (RWAs). With no minimums and no lockups, users of NUVA earn yield from institutional-grade RWAs and have the ability to user their tokens freely across DeFi. NUVA was co-created by Animoca Brands and Nuva Labs (formerly Provenance Blockchain Labs), and is stewarded by the NUVA Foundation for progressive decentralization. Learn more at NUVA.finance.

Scope

Repository: ProvLabs/nuva-evm-contracts

Audited Commit: 811305a1076b1283d9029b6b42719d050c29b12f

Final Commit: aca3c7aac3f0109303e6782193401ef816d26ada

Files:

- contracts/modules/utils/BytesLib.sol
- contracts/modules/utils/ICustomToken.sol
- contracts/modules/wormhole/ICCTPv1WithExecutor.sol
- contracts/prime/DedicatedVaultRouter.sol
- contracts/prime/NuvaVault.sol
- contracts/prime/RedemptionProxy.sol

Final Commit Hash

aca3c7aac3f0109303e6782193401ef816d26ada

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
1	3	16

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue H-1: Unconditional proxy mapping deletion permanently orphans user funds after partial or multi-tranche sweeps [RESOLVED]

Source: https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/issues/7

Summary

`sweepRedemptions` in `DedicatedVaultRouter` unconditionally deletes the `redemptionProxyToUser` mapping entry for a proxy after completing a single sweep, regardless of whether the proxy still holds additional funds.

Vulnerability Detail

This creates two independent paths to permanent fund loss:

Path 1 : Async multi-tranche delivery: The external async vault may settle a redemption request in multiple tranches, delivering a portion of the redeemed USDC now and the remainder later. After the first tranche is swept, the mapping entry is deleted. When the second tranche arrives and KEEPER attempts to sweep again, the guard evaluates `user == address(0)` (the deleted mapping), silently skips the proxy, and the remaining funds are permanently stranded.

Path 2 : KEEPER partial sweep: The KEEPER role supplies explicit amounts via the `_amounts` parameter. There is no on-chain validation that `_amounts[i]` equals the proxy's full balance. If the KEEPER provides a partial amount (intentionally or by operational error), the function sweeps that amount, deletes the mapping, and leaves the remainder permanently inaccessible.

In both cases, after the mapping is deleted:

1. Subsequent `sweepRedemptions` calls silently skip the proxy (the `user != address(0)` check fails).
2. The user cannot call `proxy.sweep()` directly, `RedemptionProxy.sweep()` is guarded by `onlyRouter`.
3. No emergency recovery or re-registration function exists in either contract.

The funds are irretrievably stranded in the proxy clone with no mechanism to recover them.

Proof Of Concept

```
function test_fundsStrandedAfterFirstSweep() public {
    // === STEP 1: Verify initial state ===
```

```
assertEq(usdc.balanceOf(address(proxy)), 1000e6, "proxy should hold 1000 USDC");
assertEq(harness.redemptionProxyToUser(address(proxy)), user, "proxy mapped to
↪  user");
uint256 userBalBefore = usdc.balanceOf(user);

// === STEP 2: First sweep - keeper sweeps 600 USDC (tranche 1) ===
address[] memory proxies = new address[](1);
proxies[0] = address(proxy);
uint256[] memory amounts = new uint256[](1);
amounts[0] = 600e6;

harness.sweepRedemptions(proxies, amounts);

// User received 600 USDC
assertEq(usdc.balanceOf(user), userBalBefore + 600e6, "user got tranche 1");
// Proxy still has 400 USDC
assertEq(usdc.balanceOf(address(proxy)), 400e6, "proxy still has 400 USDC");

// === STEP 3: Verify the BUG - mapping is now deleted ===
assertEq(
    harness.redemptionProxyToUser(address(proxy)),
    address(0),
    "BUG: mapping deleted after first sweep"
);

// === STEP 4: Second sweep attempt - keeper tries to sweep remaining 400 USDC
↪  ===
amounts[0] = 400e6;
uint256 userBalBeforeSweep2 = usdc.balanceOf(user);
harness.sweepRedemptions(proxies, amounts);

// === STEP 5: Assert HARM - user did NOT receive the remaining 400 USDC ===
assertEq(
    usdc.balanceOf(user),
    userBalBeforeSweep2,
    "HARM: user received 0 from second sweep"
);
assertEq(
    usdc.balanceOf(address(proxy)),
    400e6,
    "HARM: 400 USDC permanently stranded in proxy"
);

// === STEP 6: Verify NO recovery path exists ===
// proxy.sweep() has onlyRouter modifier, so only harness can call it
// But harness.sweepRedemptions skips when user == address(0)
// Direct call to proxy.sweep() from any non-router address will revert
vm.prank(user);
vm.expectRevert(abi.encodeWithSignature("OnlyRouter()"));
proxy.sweep(400e6);
```

```
    vm.prank(keeper);
    vm.expectRevert(abi.encodeWithSignature("OnlyRouter()"));
    proxy.sweep(400e6);

    // The 400 USDC is permanently stranded. No function can retrieve it.
    emit log_string("CONFIRMED: 400 USDC (40% of user's redemption) permanently
    ↪  stranded in proxy");
}
```

Impact

Any redemption proceeds delivered by the underlying async vault after the first tranche are permanently locked in the proxy clone. The amount at risk is any portion of the user's redemption that settles after the initial sweep.

Code Snippet

```
// DedicatedVaultRouter.sol L530-543
for (uint256 i = 0; i < length; ++i) {
    address proxyAddress = _proxyAddresses[i];
    uint256 amountToSweep = _amounts[i];
    address user = redemptionProxyToUser[proxyAddress];
    users[i] = user;

    if (user != address(0) && amountToSweep > 0) {
        IRedemptionProxy redemptionProxy = IRedemptionProxy(proxyAddress);
        totalSwept += redemptionProxy.sweep(amountToSweep);

        delete redemptionProxyToUser[proxyAddress]; // ← deletes unconditionally
        ↪  after first sweep
    }
}
```

```
// RedemptionProxy.sol L174-189 - user has no direct sweep path
function sweep(uint256 _amount) external onlyRouter returns (uint256 sweptAmount) {
    // ...
}
```

Tool Used

Manual Review

Recommendation

Only delete the proxy-to-user mapping when the proxy's underlying asset balance has been fully drained to zero. This allows subsequent sweeps for additional tranches and protects against partial-amount operational errors.

```
  totalSwept += redemptionProxy.sweep(amountToSweep);

- delete redemptionProxyToUser[proxyAddress];
+ // Only delete mapping when the proxy has been fully drained
+ if (IERC20(asset).balanceOf(proxyAddress) == 0) {
+     delete redemptionProxyToUser[proxyAddress];
+ }
```

Discussion

scirner22

resolved https://github.com/ProvLabs/nuva-evm-contracts/commit/fb1d074bf57334d727775ce9ea8a484506b8e33a

defsec

Fixed with https://github.com/ProvLabs/nuva-evm-contracts/commit/fb1d074bf57334d727775ce9ea8a484506b8e33a.

Issue M-1: `NuvaVault` violates ERC4626 spec, `maxDeposit/maxMint/maxWithdraw/maxRedeem` return non-zero when paused [RESOLVED]

Source: https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/issues/8

Summary

`NuvaVault` adds `whenNotPaused` to `deposit`, `mint`, `withdraw`, and `redeem`, meaning these functions revert when the vault is paused. However, the corresponding ERC4626 view functions – `maxDeposit`, `maxMint`, `maxWithdraw`, `maxRedeem`, are not overridden to return `0` during the paused state. This directly violates the ERC4626 specification and breaks integrating protocols that rely on these functions to determine if an operation will succeed.

Vulnerability Detail

The EIP-4626 spec is explicit (emphasis from the standard):

> `maxDeposit`: **MUST factor in both global and user-specific limits, like if deposits are entirely disabled (even temporarily) then MUST return 0.**

OpenZeppelin's base `ERC4626Upgradeable` returns unbounded values by default:

```
// @openzeppelin/contracts-upgradeable ERC4626Upgradeable.sol
function maxDeposit(address) public view virtual returns (uint256) {
    return type(uint256).max;
}
function maxMint(address) public view virtual returns (uint256) {
    return type(uint256).max;
}
function maxWithdraw(address owner) public view virtual returns (uint256) {
    return _convertToAssets(balanceOf(owner), Math.Rounding.Floor);
}
function maxRedeem(address owner) public view virtual returns (uint256) {
    return balanceOf(owner);
}
```

`NuvaVault` does not override any of these. When the vault is paused:

Function	Returns	Should Return
`maxDeposit(user)`	`type(uint256).max`	`0`

Function	Returns	Should Return
`maxMint(user)`	`type(uint256).max`	`0`
`maxWithdraw(user)`	user's convertible assets	`0`
`maxRedeem(user)`	user's share balance	`0`

An integrating protocol (e.g., a yield aggregator, the `DedicatedVaultRouter` itself, or an auto-compounder) that checks `maxDeposit > 0` before calling `deposit` would proceed with the call and get an unexpected `EnforcedPause` revert.

Impact

ERC4626 compliance violation.

Code Snippet

https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/blob/main/nuva-evm-contracts/contracts/prime/NuvaVault.sol#L98

Tool Used

Manual Review

Recommendation

Override all four `max*` functions to return `0` when paused:

```
function maxDeposit(address) public view override returns (uint256) {
    return paused() ? 0 : super.maxDeposit(address(0));
}

function maxMint(address) public view override returns (uint256) {
    return paused() ? 0 : super.maxMint(address(0));
}

function maxWithdraw(address owner) public view override returns (uint256) {
    return paused() ? 0 : super.maxWithdraw(owner);
}

function maxRedeem(address owner) public view override returns (uint256) {
    return paused() ? 0 : super.maxRedeem(owner);
}
```

Discussion

scirner22

resolved https://github.com/ProvLabs/nuva-evm-contracts/commit/295eb786b016f4ca71fc3dcd72753f217e112e4d

defsec

Fix is confirmed on the https://github.com/ProvLabs/nuva-evm-contracts/commit/295eb786b016f4ca71fc3dcd72753f217e112e4d.

Issue M-2: AML enforcement exists only at the router level [RESOLVED]

Source: https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/issues/9

Summary

The `DedicatedVaultRouter` enforces AML signature verification on every deposit and redemption. However, `NuvaVault` itself is a fully permissionless ERC4626 vault with no access control beyond `whenNotPaused`. Any user who obtains the vault's underlying asset (staking vault shares) can deposit directly into `NuvaVault`, and any holder of NuvaVault shares can redeem directly, completely bypassing AML checks.

Vulnerability Detail

`NuvaVault`'s `deposit` function has no caller restriction:

```
// NuvaVault.sol, lines 98-103
function deposit(
    uint256 assets,
    address receiver
) public override whenNotPaused returns (uint256 shares) {
    return super.deposit(assets, receiver);
}
```

No `onlyRouter`, no whitelist, no AML check. The same applies to `mint`, `withdraw`, and `redeem`.

A user can bypass the entire AML-enforced router path:

1. Acquire token.
2. Call `assetVault.deposit(usdc, self)` directly → receive assetVault shares.
3. Call `stakingVault.deposit(assetShares, self)` directly → receive stakingVault shares.
4. Call `nuvaVault.deposit(stakingShares, self)` directly → receive NuvaVault shares.

No AML signature required at any step. The reverse path (direct `nuvaVault.redeem` → `stakingVault.redeem`) is equally unrestricted.

Impact

Complete AML bypass.

Code Snippet

https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/blob/main/nuva-evm-contracts/contracts/prime/NuvaVault.sol#L98

Tool Used

Manual Review

Recommendation

If AML must be enforced protocol-wide, add access control to `NuvaVault`:

```
mapping(address => bool) public authorizedDepositors;

modifier onlyAuthorized() {
    if (!authorizedDepositors[msg.sender]) revert Unauthorized();
    _;
}

function deposit(uint256 assets, address receiver)
    public override whenNotPaused onlyAuthorized returns (uint256 shares)
{
    return super.deposit(assets, receiver);
}
```

Grant the `DedicatedVaultRouter` the authorized depositor status. Apply the same pattern to `mint`, `withdraw`, and `redeem`.

Discussion

scirner22

resolved https://github.com/ProvLabs/nuva-evm-contracts/commit/ecf102391ae413b460b2a4bb36dd57fc18291665

defsec

Fix is confirmed : https://github.com/ProvLabs/nuva-evm-contracts/commit/ecf102391ae413b460b2a4bb36dd57fc18291665

scirner22

@defsec as a follow up here, I added one direct depositWithPermit + aml signature function directly to NuvaVault in this commit https://github.com/ProvLabs/nuva-evm-contracts/commit/7a0edefb2b1e6a97f4e53f416589cfaec22ae0d7.

defsec

Thank you for informing, I will recheck new commit

iramiller

> @defsec as a follow up here, I added one direct depositWithPermit + aml signature function directly to NuvaVault in this commit ProvLabs/nuva-evm-contracts@7a0edef.

This version aligns with our business requirements better than the initial fix from yesterday.

scirner22

For completeness I added redeemWithPermit to NuvaVault as well. https://github.com/ProvLabs/nuva-evm-contracts/commit/e3e44b827bbc3ecb66711499ddc060e334775886

Issue M-3: Lack of slippage and zero-amount checks in redemptions can cause silent loss of funds [RESOLVED]

Source: https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/issues/22

Summary

The multi-hop redemption process does not let users specify a minimum expected output. If vault exchange ratios change naturally, users will receive fewer assets. Additionally, conversions can round down to zero in intermediate hops, causing users to burn shares for nothing.

Vulnerability Detail

When users deposit, they are protected by `_minVaultSharesOut`, `_minStakingVaultSharesOut`, and `_minNuvaVaultSharesOut`. However, when they redeem via `requestRedeem`, they only pass the `_amountNuvaShares` they want to burn. The `RedemptionProxy` sequentially unwinds these shares through the `Nuva Vault`, `Staking Vault`, and `Asset Vault`. Because there are no minimum output checks:

1. If the exchange rate of any vault drops due to normal market conditions before the transaction is executed, the user accepts the loss with no safety net.
2. ERC4626 vaults round down during redemptions. If the asset output of a middle hop rounds down to 0, the proxy will continue executing the next hop with 0 assets. The user will burn their initial shares but receive nothing.

Impact

Users can lose significant value simply from normal vault ratio fluctuations. In edge cases, rounding truncations will result in users burning Nuva shares for exactly 0 underlying assets without the transaction reverting.

Code Snippet

https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/blob/0b8844303af3c1fa3e6f1640b60209ad35309d18/nuva-evm-contracts/contracts/prime/DedicatedVaultRouter.sol#L404-L408

Tool Used

Manual Review

Recommendation

Update `requestRedeem` to require a user-defined minimum output parameter (e.g., `uint256 minAssetsOut`). Pass this parameter down to the `RedemptionProxy`. Add a check inside the proxy to revert if the final output is less than `minAssetsOut`, or if any intermediate unwinding step returns exactly 0.

Discussion

scirner22

resolved https://github.com/ProvLabs/nuva-evm-contracts/commit/e18f3c57f158f79bdf02fc73ee06995ccab8b600

Kirkeelee

@scirner22 The above commit implements a partial fix by adding slippage protection for the synchronous unwinding of the outer Nuva and Staking vaults into Asset Vault shares. The final asynchronous redemption step still lacks this protection. It will not be an issue if the asset vault locks the exchange rate at the time the request is made.

scirner22

It seems like this is something we will accept for now and revisit as we have more information. I believe the exchange from wYLDS to USDC will always be 1:1 but I'm not certain of that.

Issue L-1: `NuvaVault` inherits `ReentrancyGuardUpgradeable` but never applies the `nonReentrant` modifier [RESOLVED]

Source: https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/issues/10

Summary

`NuvaVault` inherits and initializes `ReentrancyGuardUpgradeable`, but none of its four state-changing entry points (`deposit`, `mint`, `withdraw`, `redeem`) use the `nonReentrant` modifier. The guard occupies a storage slot and consumes gas on initialization, but provides zero protection.

Vulnerability Detail

```
// NuvaVault.sol, lines 20-28
contract NuvaVault is
    Initializable,
    ERC4626Upgradeable,
    ERC20PermitUpgradeable,
    Ownable2StepUpgradeable,
    UUPSUpgradeable,
    PausableUpgradeable,
    ReentrancyGuardUpgradeable  // <-- inherited
{
```

```
// NuvaVault.sol, line 63
__ReentrancyGuard_init();  // <-- initialized
```

```
// NuvaVault.sol, lines 98-103
function deposit(
    uint256 assets,
    address receiver
) public override whenNotPaused returns (uint256 shares) {  // <-- no nonReentrant
    return super.deposit(assets, receiver);
}
```

All four overridden functions (`deposit`, `mint`, `withdraw`, `redeem`) only add `whenNotPaused` but never `nonReentrant`.

Impact

Unused reentrancy guard.

Code Snippet

https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/blob/main/nuva-evm-contracts/contracts/prime/NuvaVault.sol#L27

Tool Used

Manual Review

Recommendation

Remove the unused inheritance to save gas.

Discussion

scirner22

resolved https://github.com/ProvLabs/nuva-evm-contracts/commit/41b61fe74d925a8b076acd4bad8af9fbfc8ba039

defsec

Fix is confirmed on the https://github.com/ProvLabs/nuva-evm-contracts/commit/41b61fe74d925a8b076acd4bad8af9fbfc8ba039.

Issue L-2: AML signature couples slippage parameters to offchain approval [RESOLVED]

Source: https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/issues/11

Summary

The EIP-712 `DEPOSIT_TYPEHASH` includes all three slippage parameters (`minVaultShares`, `minStakingShares`, `minNuvaVaultShares`) as fields in the signed message. This means the AML signer's approval is bound to a specific set of slippage values at the moment the signature is issued. If market conditions change between the time the AML signature is issued and when the user submits their transaction, causing their original slippage tolerance to be either too tight (the transaction reverts) or unnecessarily loose (the user accepts worse terms than current conditions warrant), the user must request an entirely new AML signature to retry with different slippage parameters.

Vulnerability Detail

The AML signature's intended purpose is identity/compliance verification, not price validation. Coupling it to transient price parameters forces the off-chain AML infrastructure to re-issue signatures whenever market conditions move, creating latency and UX friction.

```
// DedicatedVaultRouter.sol:L165-L168
bytes32 private constant DEPOSIT_TYPEHASH =
    keccak256(
        "Deposit(address sender,uint256 amount,address receiver,uint256
        ↪  minVaultShares,uint256 minStakingShares,uint256
        ↪  minNuvaVaultShares,uint256 deadline)"
    );
```

Impact

In volatile market conditions, users may frequently find their AML signatures stale with respect to current exchange rates, requiring repeated interactions with the AML approval system before a deposit can succeed. This creates operational friction and may delay time-sensitive deposits.

Code Snippet

https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/blob/main/nuva-evm-contracts/contracts/prime/DedicatedVaultRouter.sol#L167

Tool Used

Manual Review

Recommendation

Consider separating the compliance identity check from the slippage parameters. The AML signature could cover only identity-related fields (`sender`, `amount`, `receiver`, `deadline`) while slippage parameters are supplied by the user at execution time without being part of the signed message.

Discussion

scirner22

resolved https://github.com/ProvLabs/nuva-evm-contracts/commit/7c24affe72f32b0be2c589c997b6c058e85751f6

defsec

@scirner22 thank you for the fix! Do we need any protection against collusion on the signature side?

scirner22

Do you mind expanding on the collusion angle so I fully understand?

defsec

Hi @scirner22 , Sorry for confusion! Since there's no nonce, could reuse of the same (sender, amount, receiver, deadline) signature enable collusion for multiple operations?

Issue L-3: Fee on transfer tokens cause permanent deposit failure [RESOLVED]

Source: https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/issues/12

Summary

The `_doDeposit` function in `DedicatedVaultRouter` transfers the user's tokens into the router, then immediately approves and deposits the same nominal `_amount` to the external vault. When the base asset is a fee-on-transfer token (one that deducts a fee during `transfer`/`transferFrom`), the router receives `_amount - fee` but attempts to deposit the original `_amount`. The vault's internal `transferFrom` call then reverts with an insufficient balance error, making deposits permanently impossible for any such token.

Vulnerability Detail

The sequence :

```
// DedicatedVaultRouter.sol L356-359
asset.safeTransferFrom(msg.sender, address(this), _amount);
// ^ router receives (_amount - fee) if asset is fee-on-transfer

asset.forceApprove(address(assetVault), _amount);
// ^ approves the FULL _amount, not what was actually received

uint256 vaultShares = assetVault.deposit(_amount, address(this));
// ^ vault calls transferFrom(router, vault, _amount)
// ^ reverts: router only holds (_amount - fee), not _amount
```

The `FundsStuck` check executes only after `assetVault.deposit()` succeeds, it is never reached when the vault deposit reverts. The revert propagates atomically back to the user.

Proof Of Concept

```
/// @notice
function test_Revert() public {
    uint256 depositAmount = 1000e18;
    uint256 deadline = block.timestamp + 1 hours;

    bytes memory sig = _signDeposit(
        user,
        depositAmount,
        user,
```

```
        0, // minVaultShares
        0, // minStakingShares
        0, // minNuvaShares
        deadline
    );

    // Verify the user has enough tokens
    uint256 userBal = fotToken.balanceOf(user);
    assertGe(userBal, depositAmount, "User should have enough tokens");

    // Verify the router has 0 tokens before
    uint256 routerBalBefore = fotToken.balanceOf(address(router));
    assertEq(routerBalBefore, 0, "Router should start with 0");

    // Attempt deposit -- should REVERT because:
    // 1. safeTransferFrom(user, router, 1000e18) delivers only 990e18 (1% fee)
    // 2. forceApprove(assetVault, 1000e18) approves full 1000e18
    // 3. assetVault.deposit(1000e18, router) calls transferFrom(router, vault,
    ↪  1000e18)
    //    but router only has 990e18 --> REVERT
    vm.prank(user);
    vm.expectRevert();
    router.deposit(
        depositAmount,
        user,
        0,
        0,
        0,
        sig,
        deadline
    );


}
```

Impact

Any fee-on-transfer token configured as the protocol's base asset renders the deposit function permanently non-functional.

Code Snippet

https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/blob/main/nuva-evm-contracts/contracts/prime/DedicatedVaultRouter.sol#L356

Tool Used

Manual Review

Recommendation

Use the actual received balance instead of the nominal `_amount` for the vault approval and deposit call. Measure the router's balance before and after the transfer to compute the true received amount:

```
+ uint256 balanceBefore = asset.balanceOf(address(this));
  asset.safeTransferFrom(msg.sender, address(this), _amount);
- asset.forceApprove(address(assetVault), _amount);
- uint256 vaultShares = assetVault.deposit(_amount, address(this));
+ uint256 actualReceived = asset.balanceOf(address(this)) - balanceBefore;
+ asset.forceApprove(address(assetVault), actualReceived);
+ uint256 vaultShares = assetVault.deposit(actualReceived, address(this));
```

Discussion

scirner22

resolved https://github.com/ProvLabs/nuva-evm-contracts/commit/fe94c1c7186c628ed81217a9c64cef9f13b4812a

defsec

Fix is confirmed on the https://github.com/ProvLabs/nuva-evm-contracts/commit/fe94c1c7186c628ed81217a9c64cef9f13b4812a

Issue L-4: AML deadline allows `type(uint256).max` [RESOLVED]

Source: https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/issues/13

Summary

`_verifyAML()` checks `block.timestamp > _deadline` but applies no upper bound on the deadline value. An AML signer can issue a signature with `_amlDeadline = type(uint256).max`, creating a permanently valid signature.

Vulnerability Detail

```
if (block.timestamp > _deadline) revert AmlSignatureExpired();
// No check: if (_deadline > block.timestamp + MAX_DEADLINE) revert ...
```

The `usedSignatures` mapping provides replay protection once consumed, but unused signatures with infinite deadlines remain a liability indefinitely. From a compliance perspective, AML approvals should be time-bound as they reflect a point-in-time assessment

Impact

Stolen AML signatures with infinite deadlines remain exploitable forever.

Code Snippet

https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/blob/main/nuva-evm-contracts/contracts/prime/DedicatedVaultRouter.sol#L589

Tool Used

Manual Review

Recommendation

Add a maximum deadline cap:

```
uint256 constant MAX_AML_DEADLINE = 7 days;
if (_deadline > block.timestamp + MAX_AML_DEADLINE) revert
↪  AmlSignatureDeadlineTooFar();
```

Discussion

scirner22

resolved https://github.com/ProvLabs/nuva-evm-contracts/commit/97174e5a6a1276340b79fc9953dc6fcde00ccda8

Issue L-5: Zero amount sweep leaves stale mapping entry causing operational state pollution [RESOLVED]

Source: https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/issues/14

Summary

The `sweepRedemptions` loop in `DedicatedVaultRouter` silently skips any proxy entry where `amountToSweep == 0`. The combined guard `if (user != address(0) && amountToSweep > 0)` prevents both the `sweep()` call and the subsequent `delete redemptionProxyToUser[proxyAddress]` from executing when the amount is zero.

Vulnerability Detail

```
// DedicatedVaultRouter.sol L530-543
for (uint256 i = 0; i < length; ++i) {
    address proxyAddress = _proxyAddresses[i];
    uint256 amountToSweep = _amounts[i];
    address user = redemptionProxyToUser[proxyAddress];
    users[i] = user;

    if (user != address(0) && amountToSweep > 0) {
        // ← zero-amount entries skip this entire block
        IRedemptionProxy redemptionProxy = IRedemptionProxy(proxyAddress);
        totalSwept += redemptionProxy.sweep(amountToSweep);
        delete redemptionProxyToUser[proxyAddress];
    }
}
emit RedemptionsSwept(_proxyAddresses, users, _amounts, totalSwept); // @audit
↪  emits even for zero-amount skips
```

The function does not revert on zero amounts, it emits a `RedemptionsSwept` event with `totalSwept = 0` for those entries, giving off-chain tooling and indexers the false impression that the sweep was processed. The mapping entry persists indefinitely without being cleaned up.

Impact

Stale mapping entries accumulate for any proxy where a zero-amount sweep was submitted.

Code Snippet

https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/blob/main/nuva-evm-contracts/contracts/prime/DedicatedVaultRouter.sol#L536

Tool Used

Manual Review

Recommendation

Replace the silent skip with an explicit revert for invalid inputs. This enforces correct KEEPER behavior and prevents stale state accumulation:

```
- if (user != address(0) && amountToSweep > 0) {
+ if (user == address(0)) revert InvalidProxy();
+ if (amountToSweep == 0) revert InvalidAmount();
+ {
      IRedemptionProxy redemptionProxy = IRedemptionProxy(proxyAddress);
      totalSwept += redemptionProxy.sweep(amountToSweep);
      delete redemptionProxyToUser[proxyAddress];
  }
```

Discussion

scirner22

The `if (user != address(0) && amountToSweep > 0)` check was an attempt at making this idempotent and not hard fail batches when only a subset of the redemptions aren't valid. Not reverting the whole batch is even more important now since after the 7 day timeout the user, and not the keeper, can sweep their own funds. This could get the keeper into a state where it thinks it needs to sweep a redemption address that's no longer in the `redemptionProxyToUser` map. Also, if the keeper requests amountToSweep of 0, then the function doesn't sweep, it's doing what the keeper asked of it so I don't think that should revert the full batch either.

I do understand the concern around stale records that are never cleaned up. What I would like to do is in the case of `amountToSweep == 0` clear those lookup values in cases where the redemptionProxy is holding 0 tokens.

scirner22

resolved https://github.com/ProvLabs/nuva-evm-contracts/commit/d25d9feb4a5addda9e37cde013e469daa4f31b53

Issue L-6: `RedemptionAlreadyPending` error defined but guard never enforced [RESOLVED]

Source: https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/issues/15

Summary

The contract declares `error RedemptionAlreadyPending()` but never uses it anywhere, suggesting a planned guard against duplicate redemption requests per user was not implemented.

Vulnerability Detail

```
// DedicatedVaultRouter.sol, line 159
error RedemptionAlreadyPending();  // declared but never referenced
```

Each call to `requestRedeem` deploys a new clone without checking if the user already has an active proxy. The `redemptionProxyToUser` mapping is proxy-to-user (not user-to-proxy), so there is no way to check on-chain whether a user already has a pending redemption.

Impact

A user with valid AML signatures can deploy arbitrarily many clones, bloating state and complicating keeper operations. The intended protection (suggested by the error name) is absent, indicating incomplete implementation.

Code Snippet

https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/blob/main/nuva-evm-contracts/contracts/prime/DedicatedVaultRouter.sol#L159

Tool Used

Manual Review

Recommendation

Add a `mapping(address => address) public userToRedemptionProxy` to track active proxies per user. Revert with `RedemptionAlreadyPending()` if a user tries to create a second proxy while one is active.

Discussion

scirner22

I commented in the related issue in more detail about this.

resolved: https://github.com/ProvLabs/nuva-evm-contracts/commit/16a10e4b0c2d6d6a9bce2894fea2284840076a54

defsec

Fixed with https://github.com/ProvLabs/nuva-evm-contracts/commit/16a10e4b0c2d6d6a9bce2894fea2284840076a54.

Issue L-7: Dual access control (`Ownable2Step` + `AccessControl`) desynchronizes on ownership transfer [RESOLVED]

Source: https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/issues/16

Summary

`DedicatedVaultRouter` uses both `Ownable2StepUpgradeable` and `AccessControlUpgradeable`. During initialization, the owner is granted `DEFAULT_ADMIN_ROLE`. When ownership is transferred, the new owner does not automatically receive `DEFAULT_ADMIN_ROLE`, and the old owner retains it.

Vulnerability Detail

```
// DedicatedVaultRouter.sol, lines 207-208
_grantRole(DEFAULT_ADMIN_ROLE, _initialOwner);
_grantRole(KEEPER_ROLE, _initialOwner);
```

After `transferOwnership(newOwner)` + `acceptOwnership()`:

- **New owner:** Has `onlyOwner` powers but **no** `DEFAULT_ADMIN_ROLE`, cannot manage `KEEPER_ROLE`.
- **Old owner:** Lost `onlyOwner` powers but **retains** `DEFAULT_ADMIN_ROLE`, can still grant `KEEPER_ROLE` to arbitrary addresses.

Impact

After ownership transfer, the previous owner retains the ability to manage all `AccessControl` roles, including granting `KEEPER_ROLE` to malicious addresses.

Code Snippet

https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/blob/main/nuva-evm-contracts/contracts/prime/DedicatedVaultRouter.sol#L207

Tool Used

Manual Review

Recommendation

Keep single role management contract.

Discussion

scirner22

resolved https://github.com/ProvLabs/nuva-evm-contracts/commit/cf7fa137571bc8fc5c85a643e724d13a0864f12f

Issue L-8: `renounceOwnership()` not overridden [RESOLVED]

Source: https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/issues/17

Summary

Both `NuvaVault` and `DedicatedVaultRouter` use `Ownable2StepUpgradeable` for safe ownership transfers. However, neither overrides `renounceOwnership()` from `OwnableUpgradeable`, which is a single-step irreversible action that sets `owner = address(0)`.

Vulnerability Detail

`Ownable2StepUpgradeable` (OZ v5) does not override `renounceOwnership()`. The parent's implementation is inherited unchanged:

```
// OwnableUpgradeable.sol
function renounceOwnership() public virtual onlyOwner {
    _transferOwnership(address(0));
}
```

Neither NuvaVault nor DedicatedVaultRouter override it. After `renounceOwnership()`:

- `owner()` returns `address(0)`.
- All `onlyOwner` functions permanently revert.

Impact

Lose of ownership.

Tool Used

Manual Review

Recommendation

Override `renounceOwnership` to revert in both contracts:

```
function renounceOwnership() public pure override {
    revert("Ownership renouncement disabled");
}
```

Discussion

scirner22

resolved https://github.com/ProvLabs/nuva-evm-contracts/commit/ce64890b7d7839937eb79cda1166e19224f7cb7f

Issue L-9: `RedemptionProxy` implementation contract lacks `_disableInitializers()` in constructor [RESOLVED]

Source: https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/issues/18

Summary

Unlike `NuvaVault` and `DedicatedVaultRouter`, the `RedemptionProxy` contract does not call `_disableInitializers()` in its constructor, allowing anyone to initialize the implementation.

Vulnerability Detail

```
// RedemptionProxy.sol - no constructor
contract RedemptionProxy is Initializable {
    // ...
}
```

An attacker could initialize the master copy, setting `router` to their own address, granting them `onlyRouter` access on the implementation

Impact

Best practice violation.

Code Snippet

https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/blob/main/nuva-evm-contracts/contracts/prime/RedemptionProxy.sol#L27

Tool Used

Manual Review

Recommendation

Add `_disableInitializers();`.

```
constructor() {
    _disableInitializers();
}
```

Discussion

scirner22

resolved https://github.com/ProvLabs/nuva-evm-contracts/commit/16a10e4b0c2d6d6a9bce2894fea2284840076a54

defsec

Fixed with https://github.com/ProvLabs/nuva-evm-contracts/commit/16a10e4b0c2d6d6a9bce2894fea2284840076a54.

Issue L-10: Hardcoded `user` address in `RedemptionProxy` [ACKNOWLEDGED]

Source: https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/issues/19

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

Summary

`RedemptionProxy` stores the user's address as a permanently fixed state variable set once in `initialize()`. The `sweep()` function transfers settled token directly to this address. There is no mechanism to update the destination address after initialization, no setter, no cancel, no redirect.

Vulnerability Detail

```
// RedemptionProxy.sol, line 94 @audit set once in initializer, never changeable
user = _user;
```

```
// RedemptionProxy.sol, line 182 @audit always sends to the hardcoded user
asset.safeTransfer(user, _amount);
```

The `initialize()` function has the `initializer` modifier, so it can only be called once per clone. No other function in the contract writes to `user`.

The risk window is the async period between `requestRedeem()` (when the user burns their NuvaVault shares and the proxy is deployed) and `sweepRedemptions()` (when the KEEPER delivers the settled token).

Impact

If a user's wallet is compromised between requesting a redemption and the KEEPER sweeping, the redeemed assets are delivered to the compromised address.

Code Snippet

https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/blob/main/nuva-evm-contracts/contracts/prime/RedemptionProxy.sol#L94

Tool Used

Manual Review

Recommendation

Add a router-callable function to update the user address on an active proxy, requiring fresh AML verification.

Discussion

scirner22

I'm not certain I understand this one. If the user's key is compromised, when a request comes in to update the user's address we can't differentiate that being called by the user trying to move the withdraw to a safe place or an attacker moving the withdraw to a separate place. It seems like this would cause a race between the correct user and malicious user to update the withdrawal address, but the malicious user always has the advantage because the correct user would not know they were compromised as quickly as the malicious user would.

defsec

Hi @scirner22 , You're right, allowing the address to be updated would just introduce a race condition that the attacker is likely to win.

This isn't really a bug, but a design tradeoff: the payout address is fixed, there's an async window, and the user has no recovery option if compromised.

So the issue is more about lack of safety mechanisms rather than preventing theft after key compromise.

Issue L-11: Owner Holds Both `UUPS` upgrade power and `DEFAULT_ADMIN_ROLE` [RESOLVED]

Source: https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/issues/20

Summary

Owner Holds Both `UUPS` upgrade power and `DEFAULT_ADMIN_ROLE`.

Vulnerability Detail

During `initialize()`, the same `_initialOwner` address is granted both `DEFAULT_ADMIN_ROLE` (which controls role management including `KEEPER_ROLE` assignment) and the `Ownable2Step` ownership that authorizes UUPS upgrades via `_authorizeUpgrade()`. A single key therefore controls: (1) the ability to upgrade the proxy logic contract to any new implementation, (2) the ability to grant or revoke `KEEPER_ROLE` on any address, (3) the ability to change the `amlSigner`, and (4) the ability to set the `redemptionProxyImplementation`.

```
// DedicatedVaultRouter.sol:L207-L208
_grantRole(DEFAULT_ADMIN_ROLE, _initialOwner);
_grantRole(KEEPER_ROLE, _initialOwner);
```

Impact

A single compromised private key results in total protocol takeover.

Tool Used

Manual Review

Recommendation

Consider separating the upgrade authority from the role management authority using a multi-signature wallet or a governance timelock for the owner role.

Discussion

scirner22

I removed the `DEFAULT_ADMIN_ROLE` in favor of allowing `onlyOwner` to manage the `KEEPER_ROLE`.

This new posture is done purposefully, because after deployment the ownership is transferred to a foundation MPC wallet. The threshold rules of that group is what we leverage to keep the owner's powers safe.

Issue L-12: `Deposited` event missing `_receiver` field [RESOLVED]

Source: https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/issues/21

Summary

The `Deposited` event records `msg.sender` as the `user` field but does not include the `_receiver` address. When `msg.sender != _receiver` (deposit-on-behalf-of), off-chain indexers tracking per-user vault positions from router events cannot determine who actually received the NuvaVault shares.

Vulnerability Detail

```
// DedicatedVaultRouter.sol, lines 102-108
event Deposited(
    address indexed user,    // msg.sender - the depositor
    uint256 assets,
    uint256 shares,
    uint256 stakingShares,
    uint256 nuvaShares
    // _receiver is MISSING
);

// Line 380: shares go to _receiver, not msg.sender
nuvaShares = nuvaVault.deposit(stakingShares, _receiver);

// Line 387-393: event uses msg.sender only
emit Deposited(msg.sender, _amount, vaultShares, stakingShares, nuvaShares);
```

The AML signature explicitly allows `msg.sender != _receiver` (the `DEPOSIT_TYPEHASH` includes both `sender` and `receiver`), confirming deposit-on-behalf-of is an intentional feature. But the event doesn't capture who received the shares, creating a blind spot for any indexer relying solely on router events.

Impact

Indexers, dashboards, and portfolio trackers that use the `Deposited` event to attribute NuvaVault shares to users will attribute them to `msg.sender` instead of `_receiver`.

Code Snippet

https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/blob/main/nuva-evm-contracts/contracts/prime/DedicatedVaultRouter.sol#L387

Tool Used

Manual Review

Recommendation

Add `_receiver` to the event:

```
event Deposited(
    address indexed user,
    address indexed receiver,
    uint256 assets,
    uint256 shares,
    uint256 stakingShares,
    uint256 nuvaShares
);

emit Deposited(msg.sender, _receiver, _amount, vaultShares, stakingShares,
↪  nuvaShares);
```

Discussion

scirner22

resolved https://github.com/ProvLabs/nuva-evm-contracts/commit/46cd006b3022f3dfee7cae24029a9f80aea9f44c

defsec

Fixed with https://github.com/ProvLabs/nuva-evm-contracts/commit/46cd006b3022f3dfee7cae24029a9f80aea9f44c.

Issue L-13: KEEPER inaction has no timeout [RESOLVED]

Source: https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/issues/23

Summary

After a user calls `requestRedeem()`, the following sequence occurs:

1. A `RedemptionProxy` clone is deployed and registered in `redemptionProxyToUser`.
2. The user's NuvaVault shares are burned through the three-hop unwinding.
3. The underlying async vault eventually settles token into the proxy.

At this point the user has irreversibly burned their shares. The only mechanism to receive the settled token is `sweepRedemptions()`, which is gated by `onlyRole(KEEPER_ROLE)`. The proxy's `sweep()` function is gated by `onlyRouter`, so neither the user nor any other party can initiate the transfer.

Vulnerability Detail

There is no timeout, deadline, or self-service escape hatch anywhere in either contract:

```
// DedicatedVaultRouter.sol L517-546
function sweepRedemptions(
    address[] calldata _proxyAddresses,
    uint256[] calldata _amounts
) external onlyRole(KEEPER_ROLE) {  // ← exclusive KEEPER gating, no expiry
    // ...
}

// RedemptionProxy.sol L174-176
function sweep(uint256 _amount) external onlyRouter returns (uint256 sweptAmount) {
    // ← user has no direct access, ever
}
```

No `block.timestamp` check exists anywhere in the sweep path. There is no function that allows the user to self-serve after a configurable waiting period. The `DEFAULT_ADMIN_ROLE` holder can grant `KEEPER_ROLE` to a new address as a governance recovery, but this requires the admin key to be responsive, an additional single-key dependency.

Impact

Users who have initiated a redemption, and thus irreversibly burned their vault shares, can be left without access to their funds indefinitely if the KEEPER becomes unavailable.

Tool Used

Manual Review

Recommendation

Add a configurable timeout after which the beneficiary user can self-sweep directly, removing the indefinite lock on their funds.

Discussion

scirner22

resolved https://github.com/ProvLabs/nuva-evm-contracts/commit/bc182bbfd34d893959c60816340a92ee55a5641f

Issue L-14: NuvaVault pause does not freeze share transfers [RESOLVED]

Source: https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/issues/24

Summary

`NuvaVault` applies `whenNotPaused` to `deposit`, `mint`, `withdraw`, and `redeem`. However, ERC20 `transfer()` and `transferFrom()` are not overridden with `whenNotPaused`. During a pause, NuvaVault shares can be freely transferred between addresses even though they cannot be minted or burned.

Vulnerability Detail

`NuvaVault` applies `whenNotPaused` to `deposit`, `mint`, `withdraw`, and `redeem`. However, ERC20 `transfer()` and `transferFrom()` are not overridden with `whenNotPaused`. During a pause, NuvaVault shares can be freely transferred between addresses even though they cannot be minted or burned.

Impact

A paused vault prevents new share issuance and redemption, but secondary market activity in NuvaVault shares remains fully active.

Tool Used

Manual Review

Recommendation

Document the intended behavior explicitly in NatSpec: either ”share transfers are intentionally permitted during pause to preserve secondary market access” or override `_update()` to check `whenNotPaused` if full transfer restriction during pause is intended.

Discussion

scirner22

resolved https://github.com/ProvLabs/nuva-evm-contracts/commit/71f4200c6a847e1afc2db67227fea9e0cc7d00a5

Issue L-15: `sweepRedemptions` CEI Violation [RESOLVED]

Source: https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/issues/26

Summary

The `sweepRedemptions` function performs an external call to `proxy.sweep()` before deleting the `redemptionProxyToUser` mapping, violating CEI pattern.

Vulnerability Detail

During the `sweep()` callback, the `redemptionProxyToUser` mapping is still populated. If the user address is a smart contract and also holds `KEEPER_ROLE`, it could theoretically re-enter `sweepRedemptions`. However, verification found that the specific harm (double extraction exceeding entitlement) is blocked by the proxy's own balance check in `sweep()`, after the first sweep empties the proxy, a second sweep of the same amount reverts with `InsufficientBalance`. The transfer also goes to the `user` address, limiting the reentrancy surface.

Impact

The reentrancy window exists and could become exploitable if future contract changes alter the proxy's balance check behavior or the sweep transfer target.

Code Snippet

```
// DedicatedVaultRouter.sol:530-543
for (uint256 i = 0; i < length; ++i) {
    address proxyAddress = _proxyAddresses[i];
    uint256 amountToSweep = _amounts[i];
    address user = redemptionProxyToUser[proxyAddress];
    users[i] = user;

    if (user != address(0) && amountToSweep > 0) {
        IRedemptionProxy redemptionProxy = IRedemptionProxy(proxyAddress);
        totalSwept += redemptionProxy.sweep(amountToSweep);  // @audit

        delete redemptionProxyToUser[proxyAddress];           // @audit CEI
        ↪  violation
    }
}
```

Tool Used

Manual Review

Recommendation

Add the `nonReentrant` modifier to `sweepRedemptions` or reorder the state mutation before the external call to follow CEI

Discussion

scirner22

resolved https://github.com/ProvLabs/nuva-evm-contracts/commit/c8eac88877dd275bd4bb2707a0ff25951efe1394

Issue L-16: Unused state variables and external calls in `RedemptionProxy` initialization [RESOLVED]

Source: https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/issues/27

Summary

The `RedemptionProxy` contract declares two state variables, `stakingAsset` and `nuvaAsset`, which are initialized via external calls but are never read or utilized anywhere in the contract.

Vulnerability Detail

In `RedemptionProxy.sol`, `stakingAsset` and `nuvaAsset` are declared as state variables. During the initialize function, external calls are made to `stakingVault.asset()` and `nuvaVault.asset()` to set these variables. Since `RedemptionProxy` is deployed as a clone for every single redemption request, making unused external calls and writing to unused storage slots significantly inflates the gas cost per redemption.

Impact

Users pay excess gas during the deployment and initialization of every `RedemptionProxy` clone due to unnecessary external calls and storage writes.

Code Snippet

https://github.com/sherlock-audit/2026-04-nuva-finance-apr-2nd/blob/0b8844303af3c1fa3e6f1640b60209ad35309d18/nuva-evm-contracts/contracts/prime/RedemptionProxy.sol#L101-L102

Tool Used

Manual Review

Recommendation

Remove the `stakingAsset` and `nuvaAsset` state variables and their associated initialization logic.

Discussion

scirner22

resolved https://github.com/ProvLabs/nuva-evm-contracts/commit/16a10e4b0c2d6d6a9bce2894fea2284840076a54

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.