Aave Adaptor and Permission Script Smart Contract Audit

Paravel, 13 February 2026

Contents

1. Introduction

iosiro was commissioned by Paravel to perform a smart contract audit of their Aave V3 leverage adaptor & micro-manager, and associated Foundry permission scripts used to configure vault access control. Two auditors conducted the audit between January 27, 2026, and February 3, 2026, using 10 audit days.

Overview

Paravel extended the Boring Vault framework into a multi-agent Aave V3 leverage management system with Uniswap V3 swaps and cross-vault fund movement. The audit focused on the contracts and scripts central to this system: the `PrvlAaveBorrow` adaptor, the legacy `PrvlFlashloanAaveBorrowV5` micro-manager, and 12 Foundry permission scripts in `script/Permissions/`.

The audit identified 11 findings: 1 medium, 4 low, and 6 informational-risk, along with 9 code quality improvement suggestions. The most impactful finding was that the `PrvlAaveBorrow` adaptor was deployed without the necessary role capabilities configured, rendering it unusable by the vault system. Other finding themes included gaps in input validation for numeric parameters, token-handling edge cases, and the maintainability of the permission scripts.

During the audit, Paravel removed the in-scope contracts (`PrvlAaveBorrow.sol` and `PrvlFlashloanAaveBorrowV5.sol`) and associated deployment scripts from the codebase. Leverage management operations (borrow, repay, and settle) are now composed off-chain and submitted directly to the vault via `manageVaultWithMerkleVerification` on `ManagerWithMerkleVerification.sol`, with no on-chain adaptor in production. As a result, all medium- and low-risk findings were closed. One informational finding related to the permission scripts remained open at the conclusion of the audit.

	Critical	High	Medium	Low	Informational
Open	0	0	0	0	1
Resolved	0	0	0	0	0
Closed	0	0	1	4	5

Scope

The assessment focused on the source files listed below, with all other files considered out of scope. Any out-of-scope code interacting with the assessed code was presumed to operate correctly without introducing functional or security vulnerabilities.

- **Project name:** boring-vault
- **Initial audit commit:** `b5154a8`
- **Final review commit:** `5662920`
- **Files:** `src/adaptors/PrvlAaveBorrow.sol`, `src/micro-managers/PrvlFlashloanAaveBorrowV5.sol`, `script/Permissions/*.s.sol`

A specification is available in the Specification section of this report.

2. Disclaimer

This report aims to provide an overview of the assessed smart contracts' risk exposure and a guide to improving their security posture by addressing identified issues. The audit, limited to specific source code at the time of review, sought to:

- Identify potential security flaws.
- Verify that the smart contracts' functionality aligns with their documentation.

Off-chain components, such as backend web application code and keeper functionality, were explicitly not within the scope of this audit.

Given the unregulated nature and ease of cryptocurrency transfers, operations involving these assets face a high risk from cyber attacks. Maintaining the highest security level is crucial, necessitating a proactive and adaptive approach that accounts for the experimental and rapidly evolving nature of blockchain technology. To encourage secure code development, developers should:

- Integrate security throughout the development lifecycle.
- Employ defensive programming to mitigate the risks posed by unexpected events.
- Adhere to current best practices wherever possible.

3. Methodology

The audit was conducted using the techniques described below.

Code review	The source code was manually inspected to identify potential security flaws. Code review is a useful approach for detecting security flaws, discrepancies between the specification and implementation, design improvements, and high-risk areas of the system.
Dynamic analysis	The contracts were compiled, deployed, and tested in a test environment, both manually and through the test suite provided. Dynamic analysis was used to identify additional edge cases, confirm that the code was functional, and to validate the reported issues.
Automated analysis	Automated tooling was used to detect the presence of various types of security vulnerabilities. Static analysis results were reviewed manually, and any false positives were removed. Any true positive results are included in this report.

4. Audit findings

The table below provides an overview of the audit's findings, with detailed write-ups in the following sections.

ID	Issue	Risk	Status
IO-PRV-APS-001	Missing role capabilities configuration for PrvlAaveBorrow adaptor	Medium	Closed
IO-PRV-APS-002	Removing token configuration clears approval	Low	Closed
IO-PRV-APS-003	Quoter/execution mismatch	Low	Closed
IO-PRV-APS-004	Weak slippage controls	Low	Closed
IO-PRV-APS-005	Ignored repay return value	Low	Closed
IO-PRV-APS-006	Missing SafeTransferLib limits token compatibility	Informational	Closed
IO-PRV-APS-007	Missing quoter zero-check	Informational	Closed
IO-PRV-APS-008	Missing authority zero-check	Informational	Closed
IO-PRV-APS-009	repay() amountIn not validated against supplyAmount	Informational	Closed
IO-PRV-APS-010	settle() amountIn unknowable due to interest accrual	Informational	Closed
IO-PRV-APS-011	Foundry key management	Informational	Open

Each issue identified during the audit has been assigned a risk rating. The rating is determined based on the criteria outlined below.

Critical risk The issue could result in the theft of funds from the contract or its users.

High risk The issue could result in the loss of funds for the contract owner or its users.

Medium risk The issue resulted in the code being dysfunctional or the specification being implemented incorrectly.

Low risk A design or best practice issue that could affect the ordinary functioning of the contract.

Informational An improvement related to best practice or a suboptimal design pattern.

In addition to a risk rating, each issue is assigned a status:

Open The issue remained present in the code as of the final commit reviewed and may still pose a risk.

Resolved The issue was identified during the audit and has since been satisfactorily addressed, removing the risk it posed.

Closed	The issue was identified during the audit and acknowledged by the developers as an acceptable risk without actioning any change.

IO-PRV-APS-001

Missing role capabilities configuration for PrvlAaveBorrow adaptor

Medium	Closed	script/DeployPrvlBorrow.s.sol

The `PrvlAaveBorrow` adaptor is deployed using `DeployPrvlBorrow.s.sol`, but no role capabilities are configured for its vault functions in the deploy script or any other script in the `Permissions/` directory. The adaptor's `supply()`, `reducePosition()`, and `settle()` functions are all gated by Solmate's `requiresAuth` modifier, which checks the `Roles Authority` contract for role-based access.

Without role capabilities, calls from the vault into the adaptor revert due to missing authorization. The vault is not the owner of the adaptor, and no role has been granted capability to call these functions.

Recommendation

The role capabilities for `PrvlAaveBorrow` should be configured during deployment or via a dedicated permission script using the following function calls:

```
rolesAuthority.setRoleCapability(STRATEGIST_ROLE, adaptor,
PrvlAaveBorrow.supply.selector, true);
rolesAuthority.setRoleCapability(STRATEGIST_ROLE, adaptor,
PrvlAaveBorrow.reducePosition.selector, true);
rolesAuthority.setRoleCapability(STRATEGIST_ROLE, adaptor,
PrvlAaveBorrow.settle.selector, true);
rolesAuthority.setUserRole(VAULT, STRATEGIST_ROLE, true);
```

A distinct adaptor is required for each vault, so this role configuration would need to be applied for each adaptor/vault pair.

Client response

Closed due to the removal of the legacy file `PrvlAaveBorrow` in `5662920`.

IO-PRV-APS-002

Removing token configuration clears approval

Low	Closed	PrvlAaveBorrow.sol#L222

`PrvlAaveBorrow.setTokenConfig()` grants token-level approvals from the adaptor to Uniswap and Aave for the configured base and deposit tokens:

```
ERC20(config.baseToken).approve(address(uniswapV3Router), type(uint256).max);
ERC20(config.baseToken).approve(address(aave), type(uint256).max);
ERC20(config.depositToken).approve(address(uniswapV3Router), type(uint256).max);
ERC20(config.depositToken).approve(address(aave), type(uint256).max);
```

`PrvlAaveBorrow.removeTokenConfig()` then unconditionally zeroes out those same approvals:

```
ERC20(config.baseToken).approve(address(uniswapV3Router), 0);
ERC20(config.baseToken).approve(address(aave), 0);
ERC20(config.depositToken).approve(address(uniswapV3Router), 0);
ERC20(config.depositToken).approve(address(aave), 0);
```

If two or more token configurations share the same base or deposit token, removing one configuration clears the approvals required by the other still-active configurations. This would temporarily cause subsequent calls that use the remaining token configs to fail until approvals are restored, e.g., by removing all configs and re-adding the required ones.

Recommendation

The potential for this issue should be documented for callers of `PrvlAaveBorrow.removeTokenConfig()`. Alternatively, per-token reference counts could be maintained so that approvals are only zeroed when no remaining config needs them.

Client response

Closed due to the removal of the legacy file `PrvlAaveBorrow` in `5662920`.

IO-PRV-APS-003

Quoter/execution mismatch

Low | Closed | PrvlFlashloanAaveBorrowV5.sol#L203

When calling `PrvlFlashloanAaveBorrowV5.getBorrowUserData()`, a swap amount is calculated internally as `swapAmount = collateralAmount + borrowAmount` and used to obtain a quote from the Uniswap V3 Quoter. However, the actual swap amount encoded in the flashloan callback data is taken from `exactInputParams.amountIn`, which is provided by the caller. These two values are never checked against each other and could differ:

```
uint256 swapAmount = collateralAmount + borrowAmount;
uint256 supplyAmount = quoter.quoteExactInput(exactInputParams.path, swapAmount);

bytes memory swapData = abi.encodeWithSelector(
    EXACT_INPUT_SELECTOR,
    exactInputParams  // includes amountIn which could differ from swapAmount
);

bytes memory supplyData = abi.encodeWithSelector(
    SUPPLY_SELECTOR, depositToken, supplyAmount, boringVault, 0
);
```

The `supplyAmount` is quoted using `swapAmount`, but the actual swap will use `exactInputParams.amountIn`. If `amountIn < swapAmount`, the swap produces less output than `supplyAmount` and the Aave supply reverts. If `amountIn > swapAmount`, extra vault tokens may be consumed silently.

Recommendation

The `exactInputParams` struct may not be necessary as the values could be derived in the function call, saving calldata and mitigating this issue. Alternatively, explicit validation should be added to ensure `exactInputParams.amountIn` equals the computed `swapAmount`.

Client response

Closed due to the removal of the legacy file `PrvlFlashloanAaveBorrowV5` in `5662920`.

IO-PRV-APS-004

Weak slippage controls

Low	Closed	PrvlAaveBorrow.sol#L108, #L138, #L167

The `swapMinOut` slippage limit in `PrvlAaveBorrow.supply()`, `PrvlAaveBorrow.reducePosition()`, and `PrvlAaveBorrow.settle()` is confirmed to be non-zero, but a value of 1 wei would still be accepted:

```
if (swapMinOut == 0) revert PrvlAaveBorrow__minOutCannotBeZero();
```

A compromised or incorrectly configured strategist could execute swaps with effectively zero slippage protection. Because the Boring Vault's merkle verification system constrains address arguments but not numeric parameters, the `swapMinOut` value is entirely under the strategist's control.

Recommendation

The functions should derive a slippage limit value from the swap amount. A slippage tolerance (e.g., 1%) could be configured for the contract and used to calculate a minimum acceptable output based on the swap input amount. This would require an oracle or quoter price for the output token.

Client response

Closed due to the removal of the legacy file `PrvlAaveBorrow` in `5662920`.

IO-PRV-APS-005

Ignored repay return value

Low	Closed	PrvlAaveBorrow.sol#L143-L145

Both `PrvlAaveBorrow.reducePosition()` and `PrvlAaveBorrow.settle()` pull base token from the vault and call `aave.repay()`, but the contract ignores the returned `uint256` repaid amount:

```
ERC20(config.baseToken).transferFrom(msg.sender, address(this), repayAmount);
aave.repay(config.baseToken, repayAmount, config.aaveVariableRate, msg.sender);
```

In Aave V3, `repay()` returns the actual amount repaid and caps repayment at the outstanding debt. If the caller supplies more than required, the surplus remains in the adaptor instead of being returned to the vault. This breaks the invariant that the adaptor should hold no funds. The stranded tokens would require a separate `PrvlAaveBorrow.sweepERC20()` call to recover.

Recommendation

The amount returned by `repay()` should be captured, and any excess should be refunded to the vault in the same transaction:

```
uint256 repaid = aave.repay(config.baseToken, repayAmount, config.aaveVariableRate, msg.sender);
if (repayAmount > repaid) {
    ERC20(config.baseToken).transfer(msg.sender, repayAmount - repaid);
}
```

Client response

Closed due to the removal of the legacy file `PrvlAaveBorrow` in `5662920`.

IO-PRV-APS-006

Missing SafeTransferLib limits token compatibility

Informational | Closed | PrvlAaveBorrow.sol

`PrvlAaveBorrow` uses Solmate's `ERC20` for all `transfer`, `transferFrom`, and `approve` calls but does not use `SafeTransferLib`. Solmate's `ERC20` ABI-decodes return data as `bool`, which reverts when a token returns no data (0 bytes). Some ERC-20 tokens, most notably USDT, return `void` on these functions.

Fork tests confirm all current production tokens (WETH, USDC, wstETH, aWstETH, variableDebtWETH) return `bool` and work correctly. If a future `setTokenConfig` attempts to register a non-standard token, the transaction reverts unconditionally – this is a denial of service (broken config), not a fund-loss vector.

Recommendation

The ERC20 return value requirement should be documented on `setTokenConfig`. Using `SafeTransferLib` should be considered for forward compatibility.

Client response

Closed due to the removal of the legacy file `PrvlAaveBorrow` in `5662920`.

IO-PRV-APS-007

Missing quoter zero-check

Informational | Closed | PrvlFlashloanAaveBorrowV5.sol#L47-L78

The `PrvlFlashloanAaveBorrowV5` constructor validates that all addresses are non-zero except for `_quoter`. If `_quoter` is `address(0)`, the contract deploys successfully but `PrvlFlashloanAaveBorrowV5.borrow()` permanently reverts since it calls `quoter.quoteExactInput()` in `getBorrowUserData()`. The `repay()` and `settle()` functions still work since `getRepayUserData()` does not use the quoter.

Since `quoter` is `immutable`, a misconfigured deployment cannot be fixed and must be redeployed.

Recommendation

The `_quoter` parameter should be added to the zero-address validation for consistency with other parameters.

Client response

Closed due to the removal of the legacy file `PrvlFlashloanAaveBorrowV5` in `5662920`.

IO-PRV-APS-008

Missing authority zero-check

Informational | Closed | PrvlAaveBorrow.sol#L80-L94

The `PrvlAaveBorrow` constructor validates `_owner`, `_uniswapV3Router`, `_aave`, and `_vault` for zero address, but does not validate `_authority`:

```
constructor(
    address _owner,
    address _authority,       // NOT zero-checked
    address _uniswapV3Router,
    address _aave,
    address _vault
) Auth(_owner, Authority(_authority)) {
    if (_owner == address(0)) revert PrvlAaveBorrow__invalidZeroAddress();
    if (_uniswapV3Router == address(0)) revert PrvlAaveBorrow__invalidZeroAddress();
    if (_aave == address(0)) revert PrvlAaveBorrow__invalidZeroAddress();
    if (_vault == address(0)) revert PrvlAaveBorrow__invalidZeroAddress();
    // _authority not checked
```

With `authority == address(0)`, the first condition in Solmate's `isAuthorized()` is always false, so only `user == owner` grants access. This means vault functions cannot be called by the vault via `RolesAuthority`, rendering the adaptor unusable by the vault system.

Recommendation

The `_authority` parameter should be added to the zero-address validation for consistency with other parameters.

Client response

Closed due to the removal of the legacy file `PrvlAaveBorrow` in `5662920`.

IO-PRV-APS-009

repay() amountIn not validated against supplyAmount

Informational | Closed | PrvlFlashloanAaveBorrowV5.sol#L156-L158

In `PrvlFlashloanAaveBorrowV5.repay()`, the caller provides three independent parameters: `borrowAmount`, `supplyAmount`, and `exactInputParams`. The function passes these directly to `getRepayUserData()` without validating their coherence. The repay flow is:

1. Repay `borrowAmount` of debt to Aave
2. Withdraw `supplyAmount` of collateral (deposit token) from Aave
3. Swap the withdrawn deposit token to base token using `exactInputParams`

For the swap to consume all withdrawn collateral, `exactInputParams.amountIn` should equal `supplyAmount`. However, there is no validation that this is the case. Mismatched values would either cause a revert (safe failure) or leave unused tokens in the vault.

Recommendation

Validation should be added to ensure `exactInputParams.amountIn` equals `supplyAmount`.

Client response

Closed due to the removal of the legacy file `PrvlFlashloanAaveBorrowV5` in `5662920`.

IO-PRV-APS-010

settle() amountIn unknowable due to interest accrual

Informational | Closed | PrvlFlashloanAaveBorrowV5.sol#L162-L166

In `PrvlFlashloanAaveBorrowV5.settle()`, the function withdraws all collateral using `type(uint256).max` but requires the caller to provide `exactInputParams.amountIn` for the subsequent swap:

```
function settle(DecoderCustomTypes.ExactInputParamsRouter02 calldata
exactInputParams)
    external requiresAuth
{
    uint256 debtAmount = ERC20(debtToken).balanceOf(boringVault);
    bytes memory innerUserData = getRepayUserData(
        type(uint256).max, type(uint256).max, exactInputParams
    );
    _executeFlashloan(innerUserData, debtAmount.mulDivDown(105, 100));
}
```

The actual collateral amount withdrawn depends on the aToken balance at execution time, which accrues interest block by block. The caller must provide `amountIn` before knowing the exact withdrawal amount.

If `amountIn` is less than the actual withdrawn amount, some deposit token remains unused in the vault after the swap. If `amountIn` is greater than the actual withdrawn amount, the swap reverts.

Recommendation

It should be documented that `settle()` may leave dust and require a follow-up sweep operation. Alternatively, the aToken balance should be read and validated against the provided `amountIn`.

Client response

Closed due to the removal of the legacy file `PrvlFlashloanAaveBorrowV5` in `5662920`.

IO-PRV-APS-011

Foundry key management

Informational	Open	script/Permissions/

The permission scripts load private keys from a `.env` file to sign messages as the Paravel deployer. This approach requires the private key to be exported from the wallet and stored in plain text on the host machine. If the host were compromised, an attacker would have direct access to the key.

Recommendation

Foundry's key management best practices support hardware wallets for signing messages. This approach is more robust, as the private key is not exported from the hardware device; instead, Foundry constructs the message, sends it to the wallet for signing, and receives the signed output.

5. Code quality improvement suggestions

Code improvement suggestions without security implications are listed below.

ID	Location	Details
IO-PRV-APS-012	PrvlFlashloanAaveBorrowV5.sol#L38	`PrvlFlashloanAaveBorrowV5` defines `MAX_SLIPPAGE = 0.1e4`, which is never referenced anywhere in the contract. This creates the misleading impression that slippage is bounded. The unused constant should be removed or actual slippage validation should be implemented.
IO-PRV-APS-013	PrvlFlashloanAaveBorrowV5.sol#L77, PrvlAaveBorrow.sol#L206	When setting the `aaveVariableRate` in the `PrvlFlashloanAaveBorrowV5` constructor and `PrvlAaveBorrow.setTokenConfig()`, the value should be confirmed to be `2` for variable rates. Other values are invalid for Paravel vaults.
IO-PRV-APS-014	PrvlAaveBorrow.sol#L241-L243	The `PrvlAaveBorrow.sweepERC20()` function transfers tokens to the vault but does not emit an event. All other state-changing functions emit events, making this an inconsistency. A `TokensSwept` event should be added.
IO-PRV-APS-015	AddAgentTeamPermmisons.s.sol, SetPrvlPermisionsMainnetUSDC.s.sol#L47, SetPrvlPermisionsMainnetWETH.s.sol#L46	Several file names contain typos (e.g. “Permmisons” → “Permissions”, “Permisions” → “Permissions”). Additionally, “custome” should be “custom” in `SetPrvlPermisionsMainnetUSDC.s.sol` (L47) and `SetPrvlPermisionsMainnetWETH.s.sol` (L46).
IO-PRV-APS-016	script/Permissions/	All deployed contract addresses and role constants are declared independently within each script, with no shared constants file. This design has poor readability and is likely to cause copy-paste errors. A shared constants contract should be created and inherited by all scripts.
IO-PRV-APS-017	script/Permissions/	Scripts could be organised into subdirectories based on the target chain (e.g., `mainnet/`, `sepolia/`, `local/`). If scripts are intended to be executed in a specific order, a number could be appended to indicate sequence, or the sequence could be consolidated into a monolithic script.
IO-PRV-APS-018	SetPrvlPermisionsMainnetUSDC.s.sol, SetPrvlPermisionsMainnetWETH.s.sol, SetPrvlAgentPermisionsMainnetForkUSDC.s.sol, SetPrvlAgentPermisionsSepoliaUSDC.s.sol	Variables already typed as `address` are unnecessarily wrapped in `address()` when passed to `setUserRole()` in `SetPrvlPermisionsMainnetUSDC.s.sol`, `SetPrvlPermisionsMainnetWETH.s.sol`, `SetPrvlAgentPermisionsMainnetForkUSDC.s.sol`, and `SetPrvlAgentPermisionsSepoliaUSDC.s.sol`. The redundant casts should be removed.
IO-PRV-APS-019	TransferAuths.s.sol, SetPrvlPermisionsMainnetUSDC.s.sol, RemoveTestAddress.s.sol	Several scripts contain unused variables and imports: `TransferAuths.s.sol` has an unused `testAddress` variable; `SetPrvlPermisionsMainnetUSDC.s.sol` has an unused `BoringSolver` import; `RemoveTestAddress.s.sol` has an unused `updatedUserRoles` variable (missing verification check); and multiple files contain unused `Authority` and `StdJson` imports.

ID	Location	Details
IO-PRV-APS-020	GrantStrategistMultisigRole.s.sol, GrantAgentAuthorityRole.s.sol	`GrantStrategistMultisigRole.s.sol` and `GrantAgentAuthorityRole.s.sol` have network selection logic in `setUp()` but `run()` uses hardcoded addresses that ignore the selection. A helper function `getRolesAuthorityAddress()` exists but is never called. The helper function should be used in `run()`, or the network selection logic should be removed entirely.

Client response

IO-PRV-APS-012	Closed due to the removal of the legacy file `PrvlFlashloanAaveBorrowV5` in `5662920`.
IO-PRV-APS-013	Closed due to the removal of the legacy files `PrvlFlashloanAaveBorrowV5`, and `PrvlAaveBorrow` in `5662920`.
IO-PRV-APS-014	Closed due to the removal of the legacy file `PrvlAaveBorrow` in `5662920`.
IO-PRV-APS-015	Open.
IO-PRV-APS-016	Open.
IO-PRV-APS-017	Open.
IO-PRV-APS-018	Open.
IO-PRV-APS-019	Open.
IO-PRV-APS-020	Open.

6. Specification

The following section outlines the system's intended functionality at a high level, based on its implementation in the codebase. Any perceived points of conflict should be highlighted with the auditing team to determine the source of the discrepancy.

System architecture

The Paravel protocol uses a two-tier vault architecture built on the Boring Vault framework. End users deposit into client vaults (e.g., iPrvlWETH, iPrvlUSDC), which allocate capital to agent vaults via cross-vault transfers. Agent vaults execute DeFi strategies – primarily leveraged Aave V3 lending positions with Uniswap V3 swaps.

All vault operations are gated by Solmate's `Auth` and `RolesAuthority`. Each vault instance has its own `RolesAuthority` contract. The `ManagerWithMerkleVerification` contract restricts vault operations via merkle proofs, where each leaf encodes a decoder/sanitizer address, a target contract, whether ETH can be sent, a function selector, and packed address arguments.

PrvlAaveBorrow

`PrvlAaveBorrow` was a standalone adaptor contract that handled Aave V3 supply/borrow leverage positions with Uniswap V3 swaps. It inherited Solmate's `Auth` for access control and `ReentrancyGuard`.

The call flow was: Strategist → `ManagerWithMerkleVerification` → `BoringVault.manage()` → `PrvlAaveBorrow`.

The adaptor supported multiple token pairs via a `tokenConfigs` mapping. Each `TokenConfig` stored the base token (borrowed asset, e.g., WETH), deposit token (collateral, e.g., wstETH), Aave aToken/debtToken addresses, interest rate mode, and Uniswap V3 swap paths. The config ID was a deterministic hash of the four token addresses.

The three core functions were:

`supply()`	Opened or increased a leveraged position. Pulled base token from the vault, swapped to deposit token via Uniswap V3, supplied to Aave as collateral on behalf of the vault, borrowed base token from Aave on behalf of the vault, and returned borrowed tokens to the vault.
`reducePosition()`	Partially unwound a position. Pulled base token, repaid Aave debt, pulled aTokens, withdrew collateral from Aave, swapped deposit token back to base token and returned it to the vault.
`settle()`	Fully closed a position. Dynamically read the vault's current debt and collateral balances, repaid all debt, withdrew all collateral, and swapped everything back to base token.

Admin functions (`setTokenConfig`, `removeTokenConfig`, `sweepERC20`) managed the token configuration registry and handled emergency token recovery.

The adaptor transiently held tokens during operations – all tokens started and ended in the `BoringVault`. The vault must have pre-approved the adaptor for base token and aToken transfers, and must have called `approveDelegation()` on the Aave debt token to allow the adaptor to borrow on its behalf.

PrvlFlashloanAaveBorrowV5

`PrvlFlashloanAaveBorrowV5` extended `UManager` and used Balancer flash loans to execute the full leverage loop atomically. Unlike `PrvlAaveBorrow`, this contract stored merkle proofs on-chain via six setter functions and supported a single token pair per deployment (all token addresses were immutable).

The call flow involved two-level merkle verification: an **outer proof** authorized the vault to call `Manager.flashLoan()`, and **inner proofs** authorized the actual DeFi operations (swap, supply, borrow or repay, withdraw) that executed inside the flashloan callback.

The three core functions (`borrow`, `repay`, `settle`) each constructed nested calldata and executed it via `_executeFlashloan()`, which in turn called `manager.manageVaultWithMerkleVerification()`.

Both `PrvlAaveBorrow` and `PrvlFlashloanAaveBorrowV5` were removed during the audit. In the current production architecture, borrow, repay, and settle operations are composed off-chain and submitted directly to the vault via `manageVaultWithMerkleVerification` on `ManagerWithMerkleVerification.sol`. The merkle root, which is maintained by the team multisig / Vault System Owner, governs which operations are permitted. No on-chain adaptor currently fulfils this role.

Permission scripts

Twelve Foundry scripts in `script/Permissions/` configure the protocol's role-based access control system. These scripts grant roles such as `OWNER_ROLE`, `UPDATE_EXCHANGE_RATE_ROLE`, and `DEPOSITOR_VAULT_ROLE` primarily to Gnosis Safe multisig addresses and set role capabilities on target contracts. The mainnet permission scripts (`SetPrvlPermisionsMainnetUSDC.s.sol`, `SetPrvlPermisionsMainnetWETH.s.sol`) are the primary production scripts, while others handle testnet deployments, role grants, and administrative operations.