RaidGuild Security Review for **Rate Hopper**

RaidGuild completed a manual and automated review of the Rate Hopper protocol smart contracts.

Audit prepared for: Rate Hopper
Security Lead: Cupojoseph

Date Audited: May 2025
Final Commit: d78cb62

Introduction

Rate Hopper is a defi tool that allows users to swap their loan positions between different money markets and lending platforms, using flashloans and atomic swaps, to pay less on borrowing and manage interest rates intelligently.

Scope

Repository: RateHopper/ratehopper-contracts
Audited Commit: 883100e78ccecebaf00921b28fab64036a0abee8
Final Commit: d78cb626702a31ee44245b85a45f89af8b85cfb0

Files:
DebtSwap.sol
LeveragedPosition.sol
ProtocolRegistry.sol
SafeModuleDebtSwap.sol
SafeModuleDebtSwapUpgradeable.sol_excluded
Types.sol
protocols/aaveV3Handler.sol
protocols/compoundHandler.sol
protocols/morphoHandler.sol
protocolsSafe/FluidSafeHandler.sol
protocolsSafe/MoonwellHandler.sol
interfaces/IProtocolHandler.sol
interfaces/aaveV3/DataTypes.sol
interfaces/aaveV3/IAaveProtocolDataProvider.sol
interfaces/aaveV3/IDebtToken.sol
interfaces/aaveV3/IPoolAddressesProvider.sol
interfaces/aaveV3/IPoolV3.sol
interfaces/aerodrome/IRouter.sol
interfaces/aerodrome/IWETH.sol
interfaces/compound/IComet.sol
interfaces/fluid/IFluidVault.sol

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
3	1	9

Issue Unresolved

High	Medium	Low
0	0	0

Disclaimers

RaidGuild and our individual contributors do not provide any 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.

interfaces/fluid/IFluidVaultResolver.sol
interfaces/moonwell/Comptroller.sol
interfaces/moonwell/IMToken.sol
interfaces/morpho/IMorpho.sol
interfaces/safe/ISafe.sol
interfaces/uniswapV3/IApproveAndCall.sol
interfaces/uniswapV3/IMulticallExtended.sol
interfaces/uniswapV3/ISwapRouter02.sol
interfaces/uniswapV3/IUniswapV3Pool.sol
interfaces/uniswapV3/IV2SwapRouter.sol
interfaces/uniswapV3/IV3SwapRouter.sol
dependencies/GPv2SafeERC20.sol
dependencies/IERC20.sol
dependencies/TransferHelper.sol
dependencies/fluid/adminModule/structs.sol
dependencies/fluid/liquidity/structs.sol
dependencies/fluid/structs.sol
dependencies/morpho/MarketParamsLib.sol
dependencies/morpho/MathLib.sol
dependencies/morpho/SharesMathLib.sol
dependencies/uniswapV3/CallbackValidation.sol
dependencies/uniswapV3/PoolAddress.sol
test/MaliciousContract.sol

Final Commit hash and link

https://github.com/RateHopper/ratehopper-contracts/commit/d78cb626702a31ee44245b85a45f89af8b85cfb0

High

Issue #5: Unauthorized withdrawFrom() calls in borrow() function

- **Status**: CLOSED
- **Severity**: Critical
- **Created**: 2025-05-24T15:33:42Z
- **Updated**: 2025-08-05T00:31:45Z

Summary: compoundHandler.sol is missing necessary access control checks. The borrow function contains a token leaking vulnerability due to improper access control. It lacks authorization checks for the onBehalfOf parameter. Any caller can execute withdrawals from any user's account by simply passing that user's address as onBehalfOf. The vulnerability allows unauthorized access to user funds through the withdrawFrom function, potentially draining user positions without their consent.

Recommendation: Add additional checks like this to Compound, Aave, and Morpho handlers:

```
require(
  msg.sender == onBehalfOf ||
  ISafe(onBehalfOf).isModuleEnabled(msg.sender), "Caller is not authorized"
);
```

Or more strictly require that only DebtSwap contract is allowed to access the handlers.

Issue #7: Front-runnable swapByParaswap() with unlimited approval can be called by anyone and drain balance.

- **Status**: CLOSED
- **Severity**: High
- **Created**: 2025-05-25T16:27:49Z
- **Updated**: 2025-08-05T00:21:15Z

Summary: Anyone can call public function to initiate swap, making it easy to front run and attack. Approval is infinite.

Recommendation:

1. Make this internal
2. Change the approve to only approve the amount of tokens that you want to swap. This should be in the txParams or elsewhere
3. Set approval to 0 at the end

```
function swapByParaswap(address asset, bytes memory _txParams) internal {
    IERC20(asset).approve(paraswapTokenTransferProxy, type(uint256).max); // should use the
amount you act
    (bool success, bytes memory returnData) = paraswapRouter.call(_txParams);
    require(success, "Token swap failed");

    //remove approval
    IERC20(asset).approve(paraswapTokenTransferProxy, 0);
}
```

Issue #6: Balance check manipulation in switchTo() collateral

- **Status**: CLOSED
- **Severity**: High
- **Created**: 2025-05-25T16:05:17Z
- **Updated**: 2025-08-06T09:16:57Z

Summary: The `switchTo` function could leak value in its collateral handling. When supplying collateral to the new position, it uses `currentBalance = IERC20(collateralAssets[i].asset).balanceOf(address(this))` to determine the amount. This balance check is vulnerable to manipulation since an attacker could transfer additional tokens to the contract before the operation happens, artificially inflating the currentBalance. The inflated balance would then be supplied as collateral, potentially allowing the attacker to borrow more assets than should be permitted based on their actual collateral.

Recommendation: Replace getting the current balance of the contract with a variable passed as parameter, that corresponds to the token.

```
function switchTo(
    address toAsset,
    uint256[] amount,
    address onBehalfOf,
    CollateralAsset[] memory collateralAssets,
    bytes calldata /* extraData */
) public override {
    address cContract = getCContract(toAsset);
    require(cContract != address(0), "Token not registered");

    IComet toComet = IComet(cContract);
    for (uint256 i = 0; i < collateralAssets.length; i++) {
        TransferHelper.safeApprove(collateralAssets[i].asset, address(cContract),
amount[i]);

        // supply collateral
        toComet.supplyTo(onBehalfOf, collateralAssets[i].asset, amount[i]);
    }
}
```

I also recommend adding additional checks, as discussed in #5, to make sure only approved users (like DebtSwap.sol) can call this.

Medium

Issue #3: proper slippage protection for paraswap

- **Status**: CLOSED
- **Severity**: Medium
- **Created**: 2025-05-22T13:43:13Z
- **Updated**: 2025-08-05T00:24:05Z

Summary: Slippage protection built into the swap data, using `srcAmount` field of `ParaswapParams`. But there should also be an error thrown if this fails for some reason, and a swap is performed outside the bounds.

Recommendation:

```
uint256 swappedAmount = swapByParaswap (
    decoded.toAsset,
    amountTotal,
    decoded.paraswapParams.swapData
);

require(
    swappedAmount >= amountTotal - [slippage calculation],
```

```
    "Excessive slippage"
);
```

Low

Issue #13: Moonwell operations missing checks for onBehalfOf before token transfer

- **Status**: CLOSED
- **Severity**: Low
- **Created**: 2025-08-05T00:05:52Z
- **Updated**: 2025-08-06T09:16:02Z

Summary: Depending on how the moonwell contracts work, token transfers shouldn't happen until after the other checks in the function have completed. Otherwise it could use another check to make sure onBehalfOf is allowed to receive those funds.

Reference: https://github.com/RateHopper/ratehopper-contracts/blame/9e88fd4105f6721157f0196e5b040b2a34d4cab9/contracts/protocolsSafe/MoonwellHandler.sol#L213

Issue #12: All handlers lack checks that tokens being moved are supported

- **Status**: CLOSED
- **Severity**: Low
- **Created**: 2025-06-01T08:51:16Z
- **Updated**: 2025-08-05T00:17:15Z

Summary: Debt switching functions in handlers lack validation on the collateralAssets array elements. Essentially, handlers should validate the individual collateral asset addresses before approving and transferring them. Each asset in the array is directly used in token approvals and protocol interactions without verification. This could lead to unintended approvals or interactions with malicious contracts.

Recommendation: Add additional checks that only allowed tokens can be transferred, or include this is debtSwap and limit access to the handler functions.

Issue #11: Switch functions could be made internal

- **Status**: CLOSED
- **Severity**: Low
- **Created**: 2025-06-01T08:38:39Z
- **Updated**: 2025-08-06T09:21:30Z

Summary: Access control is too open for the `switchTo` and `switchFrom`.

Recommendation: Since `switchTo` and `switchFrom` are intended to be called from the `switch` function in conjunction, they can be limited to internal (from external):

```
function switchFrom(
    address fromAsset,
    uint256 amount,
    address onBehalfOf,
    CollateralAsset[] memory collateralAssets,
    bytes calldata extraData
) internal override {
```

```
function switchTo(
    address toAsset,
    uint256 amount,
    address onBehalfOf,
    CollateralAsset[] memory collateralAssets,
```

```
    bytes calldata extraData
) internal override {
```

Issue #10: SafeHandler could check that the msg.sender is the owner of the safe

- **Status**: CLOSED
- **Severity**: Low
- **Created**: 2025-06-01T08:18:30Z
- **Updated**: 2025-08-05T00:21:36Z

Summary: Functions like `borrow` using safes should check who the owner is to always prevent unintended parties from accessing them.

Recommendation: More access control, as suggested elsewhere may be enough to fully handle this. But depending on what you intend users to do, I would recommend another check to require that only the owner of a particular safe is allowed to make calls from it.

```
function borrow(address asset, uint256 amount, address onBehalfOf, bytes calldata /*
extraData */) external {
    require(ISafe(onBehalfOf).isOwner(msg.sender), "Caller is not authorized");

    address mContract = getMContract(asset);
    if (mContract == address(0)) revert TokenNotRegistered();
```

The same for the handler for fluid too.

Issue #9: The uniswapV3FlashCallback function lacks validation for which pools or tokens can be used

- **Status**: CLOSED
- **Severity**: Low
- **Created**: 2025-06-01T07:58:38Z
- **Updated**: 2025-08-05T00:28:49Z

Summary: `uniswapV3FlashCallback` lacks validation that the collateral token and the pool token match. Because this function is fully external with no access control, malicious token contracts could be called here. That also means that the decoded value for `decoded.onBehalfOf` could be anything.

Recommendation: I suggest also making this function debtSwap only if that is the intended use.

Issue #8: Protocol Handler lacks validation or access controls in delegatecall()

- **Status**: CLOSED
- **Severity**: Low
- **Created**: 2025-05-31T16:45:45Z
- **Updated**: 2025-08-05T00:27:46Z

Summary: https://github.com/RateHopper/ratehopper-contracts/blob/4c38accf4c9422221aa00df0b082c540d3b44ae5/contracts/SafeModuleDebtSwap.sol#L170-L282

Recommendation: Make a check here to ensure enum is valid. If a protocol enum value is provided that hasn't been mapped to a handler address, the delegatecall would be made to address(0):

```
address fromHandler = protocolHandlers[decoded.fromProtocol];
require(fromHandler != address(0), "Invalid source protocol handler");
```

Also maybe the `handler` and `toHandler` should have non-0 checks or other validation checks if you intend this function to be external with no other controls.

Issue #4: Double Execution Risk in onlyOwnerOrExecutor Modifier

- **Status**: CLOSED
- **Severity**: Low
- **Created**: 2025-05-22T13:52:36Z
- **Updated**: 2025-08-05T00:14:06Z

Summary: The `onlyOwnerOrExecutor` modifier of `contracts/SafeModuleDebtSwap.sol` contains a structural flaw in its execution flow that could lead to unintended behavior. The modifier uses an early return pattern in the executor check path while also including the function execution placeholder (_;) in both paths. This creates two separate execution paths where the protected function could potentially be executed twice in a single transaction.

```
modifier onlyOwnerOrExecutor(address onBehalfOf) {
    if (msg.sender == executor) {
        _; // call another function which uses the modifier, but this time enter again using
the below check.
        return;
    }

    // Check if caller is any owner of the Safe
    require(ISafe(onBehalfOf).isOwner(msg.sender), "Caller is not authorized");
    _;
}
```

Recommendation: Update to:

```
modifier onlyOwnerOrExecutor(address onBehalfOf) {
    // Check if caller is any owner of the Safe or executor
    require(msg.sender == executor || ISafe(onBehalfOf).isOwner(msg.sender), "Caller is not
authorized");
    _;
}
```

Issue #2: `Repay` function can be DOS'd by high number of NFT positions

- **Status**: CLOSED
- **Severity**: Low
- **Created**: 2025-05-22T12:34:28Z
- **Updated**: 2025-08-05T00:11:44Z

Summary: FluidSafeHandler repay function can be attacked by a user holding a large number of NFTs, making it run out of gas and leaving users unable to repay, via the id iteration here:

```
// get nftId
uint256 nftId = 0;
(Structs.UserPosition[] memory userPositions_, Structs.VaultEntireData[] memory vaultsData_)
= resolver
    .positionsByUser(onBehalfOf);
for (uint256 i = 0; i < vaultsData_.length; i++) {
    if (vaultsData_[i].vault == vaultAddress) {
        nftId = userPositions_[i].nftId;
    }
}
```

Unlikely attack, especially on an L2, but users could theoretically be sent NFTs from an attacker too.

Recommendation: Decode the nftID instead, or get it another way, or acknowledge the risk:

```
(address vaultAddress, uint256 nftId) = abi.decode(
    extraData,
    (address, uint256)
);
require(nftId != 0, "Invalid NFT ID");
// other checks to make sure the NFT id is correct without looping through all owned ones.
```

Informational Issues

Issue #1: types.sol should be named Types.sol.

- **Status**: CLOSED
- **Severity**: Informational
- **Created**: 2025-05-22T12:23:14Z
- **Updated**: 2025-08-05T00:10:44Z

Summary: Some compilers are case sensitive.

Recommendation: Update the file name to `Types.sol`