DAMfinance - LMCV part 2

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **September 8th, 2022 - September 29th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY 6
CONTACTS 7
1 EXECUTIVE OVERVIEW 8
1.1 INTRODUCTION 9
1.2 AUDIT SUMMARY 9
1.3 TEST APPROACH & METHODOLOGY 9
RISK METHODOLOGY 10
1.4 SCOPE 12
2 ASSESSMENT SUMMARY & FINDINGS OVERVIEW 13
3 FINDINGS & TECH DETAILS 15
3.1 (HAL-01) USER FUNDS MAY GET LOCKED IN JOIN CONTRACTS - HIGH 16
Description 16
Code Location 16
Test scenario 17
Risk Level 18
Recommendation 18
Remediation Plan 18
3.2 (HAL-02) USERS MAY NOT BE ABLE TO UNSTAKE OR CLAIM REWARDS WHEN stakedAmountLimit IS DECREASED - MEDIUM 19
Description 19
Code Location 19
Test scenario 20
Risk Level 22
Recommendation 22
Remediation Plan 23

3.3 (HAL-03) INTEGER UNDERFLOWS IN STAKINGVAULT MODULE - MEDIUM 24
Description 24
Code Location 24
Test scenario 25
Risk Level 27
Recommendation 27
Remediation Plan 28
3.4 (HAL-04) DATA RETURNED FROM CHAINLINK IS NOT VALIDATED - MEDIUM 29
Description 29
Code Location 29
Risk Level 29
Recommendation 30
Remediation Plan 30
3.5 (HAL-05) ADMINISTRATORS ARE ALLOWED TO BURN USERS DDPRIME TOKENS WITHOUT AUTHORIZATION - MEDIUM 31
Description 31
Code Location 31
Test scenario 32
Risk Level 32
Recommendation 32
Remediation Plan 33
3.6 (HAL-06) CONTRACTS MIGHT LOSE ADMINISTRATOR FUNCTIONALITY - LOW 34
Description 34
Code Location 34
Risk Level 34

Recommendation 34
Remediation Plan 35
3.7 (HAL-07) IMPROPER ROLE-BASED ACCESS CONTROL - LOW 36
Description 36
Risk Level 36
Recommendation 36
Remediation Plan 37
3.8 (HAL-08) MISSING PAUSE/UNPAUSE FUNCTIONALITY - LOW 38
Description 38
Code Location 38
Risk Level 39
Recommendation 39
Remediation Plan 39
3.9 (HAL-09) MISSING ZERO ADDRESS CHECKS - LOW 40
Description 40
Code Location 40
Risk Level 42
Recommendation 43
Remediation Plan 43
3.10 (HAL-10) MISSING DATA VALIDATION - LOW 44
Description 44
Code Location 44
Risk Level 45
Recommendation 45
Remediation Plan 45
3.11 (HAL-11) MISSING EVENTS ON CHANGES - INFORMATIONAL 46
Description 46

Code Location 46
Risk Level 46
Recommendation 46
Remediation Plan 46
3.12 (HAL-12) REQUIRE MESSAGE NOT UPDATED - INFORMATIONAL 48
Description 48
Code Location 48
Risk Level 48
Recommendation 48
Remediation Plan 48
3.13 (HAL-13) FUNCTION COULD BE DEFINED AS EXTERNAL - INFORMATIONAL 50
Description 50
Code Location 50
Risk Level 50
Recommendation 50
Remediation Plan 50
3.14 (HAL-14) MISSING NATSPEC DOCUMENTATION - INFORMATIONAL 51
Description 51
Risk Level 51
Recommendation 51
Remediation Plan 51
3.15 (HAL-15) VARIABLES COULD BE DEFINED AS IMMUTABLE - INFORMATIONAL 52
Description 52
Code Location 52
Risk Level 52

Recommendation 52

Remediation Plan 53

3.16 (HAL-16) CHANGING PUBLIC CONSTANT VARIABLES TO PRIVATE CAN SAVE GAS - INFORMATIONAL 54

Description 54

Code Location 54

Risk Level 54

Recommendation 55

Remediation Plan 55

4 AUTOMATED TESTING 56

4.1 STATIC ANALYSIS REPORT 57

Description 57

Slither results 57

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	09/29/2022	Pawel Bartunek
0.2	Document amendments	09/30/2022	Pawel Bartunek
0.3	Document Update	10/03/2022	Kubilay Onur Gungor
0.4	Document Update	10/03/2022	Gabi Urrutia
1.0	Remediation Plan	10/10/2022	Pawel Bartunek
1.1	Remediation Plan Review	10/13/2022	Kubilay Onur Gungor
1.2	Remediation Plan Review	10/13/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Kubilay Onur Gungor	Halborn	Kubilay.Gungor@halborn.com
Pawel Bartunek	Halborn	Pawel.Bartunek@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

DAMfinance engaged Halborn to conduct a security audit on their smart contracts beginning on September 8th, 2022 and ending on September 29th, 2022. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided three weeks for the engagement and assigned a full-time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn identified some security risks that were mostly addressed by the DAMfinance team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the bridge code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose
- Smart contract manual code review and walk-through
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes
- Manual testing by custom scripts
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Local deployment (Hardhat, Remix IDE, Brownie)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.

1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

IN-SCOPE:
The security assessment was scoped to the following smart contracts:

new LMCV contracts:

- lmcv/AuctionHouse.sol
- lmcv/ChainlinkClient.sol
- lmcv/Liquidator.sol
- lmcv/OSM.sol
- lmcv/PriceUpdater.sol
- lmcv/PSM.sol
- lmcv/RatesUpdater.sol

Staking:

- staking/RewardJoin.sol
- staking/StakeJoin.sol
- staking/StakingVault.sol
- staking/ddPrime.sol
- staking/ddPrimeJoin.sol

Commit ID: f5861988508a5bb291a3a7f2863693cd9762dee7
Pull request: 25

Remediation plan:

Pull Request: 30
Branch: secondRoundAuditFixes
Commit ID 9798fb6f03aab96d8702116e6bef394b2e501d59

OUT-OF-SCOPE:
Other smart contracts in the repository, external libraries and economical attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	1	4	5	6

LIKELIHOOD

IMPACT

	(HAL-02) (HAL-05)	(HAL-01)		
	(HAL-04)			
(HAL-06)	(HAL-07) (HAL-08)	(HAL-03)		
	(HAL-10)	(HAL-09)		
(HAL-11) (HAL-12) (HAL-13) (HAL-14) (HAL-15) (HAL-16)				

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 - USER FUNDS MAY GET LOCKED IN JOIN CONTRACTS	High	SOLVED - 10/10/2022
HAL-02- USERS MAY NOT BE ABLE TO UNSTAKE OR CLAIM REWARDS WHEN stakedAmountLimit IS DECREASED	Medium	SOLVED - 10/11/2022
HAL-03 - INTEGER UNDERFLOWS IN STAKINGVAULT MODULE	Medium	SOLVED - 10/11/2022
HAL-04 - DATA RETURNED FROM CHAINLINK IS NOT VALIDATED	Medium	SOLVED - 10/10/2022
HAL-05 - ADMINISTRATORS ARE ALLOWED TO BURN USERS DDPRIME TOKENS WITHOUT AUTHORIZATION	Medium	FUTURE RELEASE
HAL-06 - CONTRACTS MIGHT LOSE ADMINISTRATOR FUNCTIONALITY	Low	SOLVED - 10/10/2022
HAL-07 - IMPROPER ROLE-BASED ACCESS CONTROL	Low	RISK ACCEPTED
HAL-08 - MISSING PAUSE/UNPAUSE FUNCTIONALITY	Low	SOLVED - 10/11/2022
HAL-09 - MISSING ZERO ADDRESS CHECK	Low	SOLVED - 10/11/2022
HAL-10 - MISSING DATA VALIDATION	Low	PARTIALLY SOLVED - 10/11/2022
HAL-11 - MISSING EVENTS ON CHANGES	Informational	SOLVED - 10/11/2022
HAL-12 - REQUIRE MESSAGES NOT UPDATED	Informational	SOLVED - 10/11/2022
HAL-13 - FUNCTION COULD BE DEFINED AS EXTERNAL	Informational	SOLVED - 10/11/2022
HAL-14 - MISSING NATSPEC DOCUMENTATION	Informational	FUTURE RELEASE
HAL-15 - VARIABLES COULD BE DEFINED AS IMMUTABLE	Informational	SOLVED - 10/11/2022
HAL-16 - CHANGING PUBLIC CONSTANT VARIABLES TO PRIVATE CAN SAVE GAS	Informational	SOLVED - 10/11/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) USER FUNDS MAY GET LOCKED IN JOIN CONTRACTS - HIGH

Description:

The RewardJoin and StakeJoin contracts have possibility to be stopped/disabled by the administrator using cage function.

When the administrator calls the cage function, user funds will get locked as the users will not be able to call the exit function successfully and get the staked and rewards tokens.

The contracts also do not implement a possibility to start the contract again (setting the live flag to 1), so all funds will get locked inside the contract.

Code Location:

cage function:

Listing 1: contracts/staking/RewardJoin.sol (Line 71)

```
70 function cage() external auth {
71     live = 0;
72     emit Cage();
73 }
```

staking/RewardJoin.sol

Listing 2: contracts/staking/RewardJoin.sol (Line 100)

```
 99 function exit(address usr, uint256 wad) external {
100     require(live == 1, "CollateralJoin/not-live");
101     stakingVault.pullRewards(collateralName, msg.sender, wad);
102     require(collateralContract.transfer(usr, wad), "RewardJoin/
 ↳ failed-transfer");
103     emit Exit(usr, wad);
104 }
```

staking/StakeJoin.sol

Listing 3: contracts/staking/StakeJoin.sol (Line 98)

```
97 function exit(address usr, uint256 wad) external {
98     require(live == 1, "CollateralJoin/not-live");
99     stakingVault.pullStakingToken(msg.sender, wad);
100    require(collateralContract.transfer(usr, wad), "CollateralJoin
↳ /failed-transfer");
101    emit Exit(usr, wad);
102 }
```

Test scenario:

Example Hardhat test cases:

Listing 4

```
1  it("HAL-01 User staked tokens are locked (StakeJoin)", async
↳ function () {
2      // User stakes tokens
3      let userStakeJoin = stakeJoin.connect(addr1);
4      await userStakeJoin.join(addr1.address, fwad("1000"));
5
6      // Contract is stopped
7      await stakeJoin.cage();
8
9      // User can't exit, collateral is locked in the contract
10     await expect(userStakeJoin.exit(addr1.address, fwad("500")))
11         .to.be.revertedWith("CollateralJoin/not-live");
12 });
```

Listing 5

```
1  it("HAL-01 User rewards are locked (RewardsJoin)", async function
↳ () {
2      // User stakes tokens
3      let userStakeJoin = stakeJoin.connect(addr1);
4      await userStakeJoin.join(addr1.address, fwad("1000"));
5
6      // User stakes tokens in the vault
7      userSV = stakingVault.connect(addr1);
```

```
 8     await userSV.stake(fwad("800"), addr1.address);
 9
10     // Rewards are added
11     await fooJoin.join(fwad("1000"));
12
13     // User stakes 0 to claim rewards
14     await userSV.stake("0", addr1.address);
15
16     // Contract is stopped
17     await fooJoin.cage();
18
19     // User tries to exit rewards join contract
20     let userFooJoin1 = fooJoin.connect(addr1);
21     await expect(userFooJoin1.exit(addr1.address, "
↳ 10000000000000000000"))
22         .to.be.revertedWith("CollateralJoin/not-live");
23 });
```

Risk Level:

Likelihood - 3
Impact - 5

Recommendation:

Consider allowing users to call `exit` when the contract is not live, or add a possibility to make the contract live again.

Remediation Plan:

SOLVED: The `cage` function in the `RewardJoin` and `StakeJoin` contracts was updated, adding a possibility to make the contract live again.

Reference: `RewardJoin.sol` and `StakeJoin.sol`

3.2 (HAL-02) USERS MAY NOT BE ABLE TO UNSTAKE OR CLAIM REWARDS WHEN stakedAmountLimit IS DECREASED - MEDIUM

Description:

The StakingVault contract defines a maximum amount allowed to stake in the stakedAmountLimit variable. An administrator can modify this value. However, the setStakedAmountLimit function does not perform any validation on its parameter.

It is possible to decrease stakedAmountLimit to a value lower than the currently staked amount. In such a situation, some users may not be able to unstake or claim their rewards because of the require statement in the stake function, enforcing that stakedAmount must be lower or equal to the limit.

Example scenario:

- stakedAmountLimit is set to 5000
- User 1 stakes 4000 tokens
- User 2 stakes 1000 tokens
- Administrator decreases stakedAmountLimit to 3000
- User 2 can't unstake/claim rewards as transaction gets reverted due to require statement

In the above scenario, User 2 won't be able to unstake or claim the rewards until User 1 decreases his staked amount.

Code Location:

setStakedAmountLimit setter:

Listing 6: contracts/staking/StakingVault.sol (Line 158)

```
157 function setStakedAmountLimit(uint256 wad) external auth {
158     stakedAmountLimit = wad;
159     emit StakedAmountLimit(wad);
160 }
```

Unstaking may not be possible, because of the require statement in stake function, enforcing that current stakedAmount + new staked amount is lower than the limit:

Listing 7: contracts/staking/StakingVault.sol (Line 240)

```
231 function stake(int256 wad, address user) external stakeAlive { //
 ↳ [wad]
232     require(approval(user, msg.sender), "StakingVault/Owner must
 ↳ consent");
233     require(getOwnedDDPrime(user) >= lockedStakeable[user] *
 ↳ stakedMintRatio, "StakingVault/Need to own ddPRIME to cover locked
 ↳  amount");
234
235     //1. Add locked tokens
236     uint256 prevStakedAmount    = lockedStakeable[user]; //[wad]
237     unlockedStakeable[user]     = _sub(unlockedStakeable[user],
 ↳ wad);
238     lockedStakeable[user]       = _add(lockedStakeable[user], wad)
 ↳ ;
239     stakedAmount                = _add(stakedAmount, wad);
240     require(stakedAmount <= stakedAmountLimit, "StakingVault/
 ↳ Cannot be over staked token limit");
241 [...]
```

Test scenario:

Hardhat test case:

Listing 8

```
 1 it("HAL-02 User may not be able to unstake/claim rewards when
 ↳ stakedAmountLimit is decreased", async function () {
 2
```

```
3     //User 1 and 2 add 10000 stakeable coin with join contract
4     await userStakeJoin.join(addr1.address, fwad("10000"));
5     await userStakeJoin2.join(addr2.address, fwad("10000"));
6
7     // User1 stakes 4000 of tokens (stakedAmount< limit)
8     await userSV.stake(fwad("4000"), addr1.address);
9
10    // assert balances
11    expect(await stakingVault.lockedStakeable(addr1.address)).to.
↳ equal(fwad("4000"));
12    expect(await stakingVault.unlockedStakeable(addr1.address)).to
↳ .equal(fwad("6000"));
13    expect(await stakingVault.ddPrime(addr1.address)).to.equal(
↳ frad("4000"));
14
15    // User2 stakes 1000 of tokens  (stakedAmount == limit)
16    await userSV2.stake(fwad("1000"), addr2.address);
17
18    // assert balances
19    expect(await stakingVault.lockedStakeable(addr2.address)).to.
↳ equal(fwad("1000"));
20    expect(await stakingVault.unlockedStakeable(addr2.address)).to
↳ .equal(fwad("9000"));
21    expect(await stakingVault.ddPrime(addr2.address)).to.equal(
↳ frad("1000"));
22
23    // Foo Rewards get added
24    await fooJoin.join(fwad("50"));
25
26    // limit is changed to 3000
27    await stakingVault.setStakedAmountLimit(fwad("3000"));
28
29    // user 2 stakes 0 to claim rewards - transaction fails
30    await expect(userSV2.stake(fwad("0"), addr2.address))
31        .to.be.revertedWith("StakingVault/Cannot be over staked
↳ token limit");
32
33    // user 2 tries to unstake -1000 tokens to claim rewards -
↳ transaction fails
34    await expect(userSV2.stake(fwad("-1000"), addr2.address))
35        .to.be.revertedWith("StakingVault/Cannot be over staked
↳ token limit");
36
```

```
37      // User 1 stakes 0 to claim rewards - transaction fails,
↳ without reducing staked tokens
38      await expect(userSV.stake(fwad("0"), addr1.address))
39          .to.be.revertedWith("StakingVault/Cannot be over staked
↳ token limit");
40
41      // User 1 reduces staked tokens to claim rewards
42      await userSV.stake(fwad("-2000"), addr1.address);
43
44      // assert reward balances
45      expect(await stakingVault.rewardDebt(addr1.address, fooBytes))
↳ .to.equal(fwad("20"));
46      expect(await stakingVault.rewardDebt(addr2.address, fooBytes))
↳ .to.equal(0);
47      expect(await stakingVault.withdrawableRewards(addr1.address,
↳ fooBytes)).to.equal(fwad("40"));
48      expect(await stakingVault.withdrawableRewards(addr2.address,
↳ fooBytes)).to.equal(0);
49
50      // Now user 2 is able to claim rewards/unstake
51      await userSV2.stake(fwad("0"), addr2.address);
52      expect(await stakingVault.rewardDebt(addr2.address, fooBytes))
↳ .to.equal(fwad("10"));
53      expect(await stakingVault.withdrawableRewards(addr2.address,
↳ fooBytes)).to.equal(fwad("10"));
54  });
```

Risk Level:

Likelihood - 2
Impact - 5

Recommendation:

One of the solutions might be to add a check in the `setStakedAmountLimit` function and not allow decreasing a limit below the currently staked amount.

Remediation Plan:

SOLVED: The `require` statement in the `stake` function was modified to always allow a negative amount (withdraw request) - in this way, the user will always be able to withdraw the amount staked at `0`. When the user tries to withdraw more than staked, the transaction is reverted in the `_add` function.

Reference: StakingVault.sol

3.3 (HAL-03) INTEGER UNDERFLOWS IN STAKINGVAULT MODULE - MEDIUM

Description:

There are multiple instances in the StakingVault contract when subtracting balances is done without checks. Such behavior may cause the transaction to fail due to arithmetic errors (integer underflow), for example, when the user balance is lower than the requested withdrawal.

Code Location:

The pullStakingToken function is not verifying balance before subtraction in staking/StakingVault.sol:

Listing 9: contracts/staking/StakingVault.sol (Line 201)

```
200 function pullStakingToken(address user, uint256 wad) external auth
 ↳  {
201     unlockedStakeable[user] -= wad;
202     emit PullStakingToken(user, wad);
203 }
```

The pullRewards function is not checking if user has enough tokens to move in staking/StakingVault.sol:

Listing 10: staking/StakingVault.sol (Line 222)

```
220 function pullRewards(bytes32 rewardToken, address usr, uint256 wad
 ↳ ) external auth {
221     RewardTokenData storage tokenData = RewardData[rewardToken];
222     withdrawableRewards[usr][rewardToken]   -= wad;
223     tokenData.totalRewardAmount             -= wad;
224     emit PullRewards(rewardToken, usr, wad);
225 }
```

The liquidationWithdraw does not check if liquidator has enough ddPrime

in staking/StakingVault.sol:

Listing 11: staking/StakingVault.sol (Line 263)

```
254 function liquidationWithdraw(address liquidator, address
 ↳ liquidated, uint256 rad) external {
255     require(approval(liquidator, msg.sender), "StakingVault/Owner
 ↳ must consent");
256
257     //1. Check that liquidated does not own ddPrime they claim to
258     require(getOwnedDDPrime(liquidated) <= lockedStakeable[
 ↳ liquidated] * stakedMintRatio - rad, "StakingVault/Account must
 ↳ not have ownership of tokens");
259     uint256 liquidatedAmount         = rad / stakedMintRatio; //
 ↳ rad / ray = wad
260     uint256 prevStakedAmount         = lockedStakeable[liquidated
 ↳ ]; //[wad]
261
262     //2. Take ddPrime from liquidator's account to repay
263     ddPrime[liquidator]              -= rad;
264     totalDDPrime                     -= rad;
265 [...]
```

Test scenario:

Example Hardhat test cases:

Listing 12

```
 1 it("HAL-03 Integer underflow - withdraw more than staked", async
 ↳ function () {
 2     // User tries to exit from stake join contract without staking
 ↳  anything before
 3     let userStakeJoin = stakeJoin.connect(addr1);
 4     await expect(userStakeJoin.exit(addr1.address, fwad("500")))
 5         .to.be.revertedWith("Arithmetic operation underflowed or
 ↳ overflowed outside of an unchecked block");
 6 });
```

Listing 13

```
it("HAL-03 Integer underflow - withdraw more than withdrawable (RewardJoin)", async function () {
    // User joins stake
    let userStakeJoin = stakeJoin.connect(addr1);
    await userStakeJoin.join(addr1.address, fwad("1000"));

    // User stakes tokens in the vault
    userSV = stakingVault.connect(addr1);
    await userSV.stake(fwad("800"), addr1.address);

    // Rewards are added
    await fooJoin.join(fwad("1000"));

    // User stakes 0 to claim rewards
    await userSV.stake("0", addr1.address);
    expect(await stakingVault.withdrawableRewards(addr1.address, fooBytes)).to.equal(fwad("1000"));

    // User tries to exit reward join contract with more tokens than withdrawable
    let userFooJoin1 = fooJoin.connect(addr1);
    await expect(userFooJoin1.exit(addr1.address, fwad("10000")))
        .to.be.revertedWith("Arithmetic operation underflowed or overflowed outside of an unchecked block");
});
```

Listing 14

```
it("HAL-03 Integer underflow - Liquidator doesn't hold enough ddPrime", async function () {
    // Sanity checks
    expect(await lmcv.lockedCollateral(addr4.address, ddPrimeBytes)).to.equal(fwad("1000"));
    expect(await lmcv.lockedCollateral(addr4.address, blorpBytes)).to.equal(fwad("1000"));
    expect(await ddPrime.balanceOf(addr4.address)).to.equal(0);
    expect(await stakingVault.ddPrime(addr4.address)).to.equal(0);
    expect(await stakingVault.getOwnedDDPrime(addr4.address)).to.equal(frad("1000"));
    expect(await lmcv.normalizedDebt(addr4.address)).to.equal(fwad("100"));

```

```
10      // User 4 stakes 1000 to claim rewards
11      await userStakeJoin4.join(addr4.address, fwad("1000"));
12      await userSV4.stake(fwad("1000"), addr4.address);
13
14      // Assert rewards
15      expect(await stakingVault.withdrawableRewards(addr4.address,
↳ fooBytes)).to.equal("0");
16      expect(await stakingVault.rewardDebt(addr4.address, fooBytes))
↳ .to.equal(fwad("20"));
17
18      // Foo Rewards get added
19      await fooJoin.join(fwad("20"));
20
21      // User 4 stakes 0 to claim rewards
22      await userSV4.stake(fwad("0"), addr4.address);
23
24      // User 1 gets liquidated, ddPrime is transferred to user 3 (
↳ bypasses auction functionality)
25      await lmcv.seize([ddPrimeBytes], [fwad("1000")], fwad("100"),
↳ addr4.address, addr3.address, owner.address);
26
27      // User 3 calls liquidationWithdraw not having enough ddPrime
28      await expect(userSV3.liquidationWithdraw(addr3.address, addr4.
↳ address, frad("1000")))
29          .to.be.revertedWith("Arithmetic operation underflowed or
↳ overflowed outside of an unchecked block");
30  });
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

Consider adding a validation before calculating the balances to avoid integer underflow and return an appropriate error message to the user.

Remediation Plan:

SOLVED: Additional require statements were added, to ensure underflow does not occur:

StakingVault.sol: #216, 238, 280

3.4 (HAL-04) DATA RETURNED FROM CHAINLINK IS NOT VALIDATED - MEDIUM

Description:

The getLatestPrice function in the contract ChainlinkClient.sol fetches the asset price from a Chainlink aggregator using the latestRoundData function. However, there are no checks on roundID nor timeStamp, which may result in stale prices.

If there is a problem with Chainlink starting a new round and finding consensus on the new value for the oracle (e.g., Chainlink nodes abandon the oracle, chain congestion, vulnerability/attacks on the Chainlink system), consumers of this contract may continue using outdated stale data (if oracles are unable to submit no new round is started).

Code Location:

Listing 15: contracts/lmcv/ChainlinkClient.sol (Line 26)

```
18 function getLatestPrice() public view returns (int256) {
19     (
20         /*uint80 roundID*/,
21         int256 price,
22         /*uint startedAt*/,
23         /*uint timeStamp*/,
24         /*uint80 answeredInRound*/
25     ) = priceFeed.latestRoundData();
26     return price;
27 }
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

Consider adding checks on the return data such as `price > 0`, `timestamp != 0` and `answeredInRound >= roundID`. Add a proper revert message if the price is stale or the round is incomplete.

Remediation Plan:

SOLVED: Validation of data returned from Chainlink was added.

Reference: ChainlinkClient.sol

3.5 (HAL-05) ADMINISTRATORS ARE ALLOWED TO BURN USERS DDPRIME TOKENS WITHOUT AUTHORIZATION - MEDIUM

Description:

An administrator of the ddPrime token can burn users' tokens. Such action might break the StakingVault contract, as it keeps its copy of ddPrime balances.

Code Location:

ddPrime.sol

Listing 16: staking/ddPrime.sol (Line 159)

```
155 function burn(address from, uint256 value) external {
156     uint256 balance = balanceOf[from];
157     require(balance >= value, "ddPrime/insufficient-balance");
158
159     if (from != msg.sender && admins[msg.sender] != 1) {
160         uint256 allowed = allowance[from][msg.sender];
161         if (allowed != type(uint256).max) {
162             require(allowed >= value, "ddPrime/insufficient-
 ↳ allowance");
163
164             unchecked {
165                 allowance[from][msg.sender] = allowed - value;
166             }
167         }
168     }
169 [...]
```

Test scenario:

Hardhat test scenario:

Listing 17

```
it("HAL-05 Administrator burns users ddPrime", async function () {

    //User 1 joins and stakes 1000 tokens
    await userStakeJoin.join(addr4.address, fwad("10000"));
    await userSV4.stake(fwad("1000"), addr4.address);

    // Approve and exit with ddPrime
    await userSV4.approve(ddPrimeJoin.address);
    userDDPrimeJoin = ddPrimeJoin.connect(addr4);
    await userDDPrimeJoin.exit(addr4.address, fwad("1000"));

    // Assert balance
    expect(await stakingVault.ddPrime(addr4.address)).to.equal(0);
    expect(await ddPrime.balanceOf(addr4.address)).to.equal(fwad("
↳ 1000"));

    // Administrator burns ddPrime
    await ddPrime.burn(addr4.address, fwad("500"));

    // Assert balance
    expect(await ddPrime.balanceOf(addr4.address)).to.equal(fwad("
↳ 500"));
});
```

Risk Level:

Likelihood - 2
Impact - 5

Recommendation:

Administrator should not have an ability to burn user tokens, without approval.

Remediation Plan:

PENDING: The DAMfinance team stated that they plan to implement the recommended fix with the governance module in the future.

3.6 (HAL-06) CONTRACTS MIGHT LOSE ADMINISTRATOR FUNCTIONALITY - LOW

Description:

The `deny` function is not checking if there are any other active `wards` before setting `wards[usr] = 0`. The contract will lose administrator functionality when the only `ward` user calls this function.

Code Location:

OSM.sol, #32

RatesUpdater.sol, #30

Liquidator.sol, #68

AuctionHouse.sol, #26

staking/ddPrime.sol, #68

staking/RewardJoin.sol, #34

staking/StakeJoin.sol, #32

staking/StakingVault.sol, #102

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Consider adding validation to ensure at least one privileged account is left.

Remediation Plan:

SOLVED: The ArchAdmin variable was added to the contract. The address assigned to this field cannot be removed from the wards/admins mapping via the administrate or deny functions, ensuring that there is at least one administrator on the contract. To update this address, a new ArchAdmin must be set; then, the address can be removed from the admin mapping.

- OSM.sol
- RatesUpdater.sol
- Liquidator.sol
- AuctionHouse.sol
- staking/ddPrime.sol
- staking/RewardJoin.sol
- staking/StakeJoin.sol
- staking/StakingVault.sol

3.7 (HAL-07) IMPROPER ROLE-BASED ACCESS CONTROL - LOW

Description:

The smart contracts in scope do not implement granular access control. All the privileged functionality is assigned to one role. This could lead to severe consequences if, for example, such an account gets compromised or a malicious administrator decides to take over the platform.

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Consider a more robust RBAC (Role Based Access Control) that allows for finer-grained control as to which functionalities users and contracts have permissions, rather than a blanket ‘admin’ access to all.

For instance, the following user roles could be set:

- protocolAdmin - responsible for setting loans, fees, debt ceiling, etc.
- collateralAdmin - used for managing collateral-related functions
- keepers/oracle - used for updating prices/rates
- owner/admin - used for most sensitive actions like adding/removing admins

A secure multisig would be best utilized as an administrator, with all permissions and the ability to add or remove permissions to other addresses.

Remediation Plan:

RISK ACCEPTED: The DAM finance team accepted the risk of this finding. In this role-based admin structure, only smart contracts would have access to specific roles, and a person-controlled owner address would have the ability to set all of these roles. Since a smart contract can only call functions it has interfaces for and admin access to in this setup, and an owner-level admin hack would have the ability to set itself as any other level admin, this does not seem like a useful check.

3.8 (HAL-08) MISSING PAUSE/UNPAUSE FUNCTIONALITY - LOW

Description:

In case a hack occurs, or an exploit is discovered, the team should be able to pause functionality until the necessary changes are made to the system.

To use a THORchain example again, the team behind the THORchain noticed an attack was going to occur well before the system transferred funds to the hacker. However, they were unable to shut the system down fast enough (According to the incident report).

Following contracts: Liquidator, AuctionHouse, RewardsJoin, and StakeJoin implement a cage function which the administrator may use to stop the contract. However, there is no possibility of putting the contract into a live state again.

The StakingVault contract has a stakeLive flag, which the stakeAlive modifier uses to allow access to the stake function. During an audit, it was discovered that the contract couldn't be paused/stopped. The stakeLive flag is set to 1 in the constructor, and there is no method to change it later.

Code Location:

lmcv/Liquidator.sol, #125

lmcv/AuctionHouse.sol, #92

stake/RewardsJoin.sol, #71

stake/StakeJoin.sol, #69

stake/StakingVault.sol, #93

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Pause functionality on the contract would have helped secure the funds quickly in an emergency.

Remediation Plan:

SOLVED: The `cage` function was modified, and the `setStakeAlive` function was added to the contracts. Now, contracts can be stopped/resumed in case of an attack.

- lmcv/Liquidator.sol
- lmcv/AuctionHouse.sol
- stake/RewardJoin.sol
- stake/StakeJoin.sol
- stake/StakingVault.sol

3.9 (HAL-09) MISSING ZERO ADDRESS CHECKS - LOW

Description:

Contracts in-scope are missing address validation in constructors and setter functions. It is possible to configure the 0x0 address, which will cause issues during execution.

Code Location:

lmcv/ChainlinkClient.sol

Listing 18: lmcv/ChainlinkClient.sol (Line 12)

```
11 constructor(address priceFeedAddress) {
12     priceFeed = AggregatorV3Interface(priceFeedAddress);
13 }
```

lmcv/OSM.sol

Listing 19: lmcv/OSM.sol (Line 93)

```
92 function changeOracleAddress(address _oracleAddress) external auth
↳  {
93     oracleAddress = _oracleAddress;
94 }
```

lmcv/PriceUpdater.sol

Listing 20: lmcv/PriceUpdater.sol (Line 66)

```
64 constructor(address vat_) {
65     wards[msg.sender] = 1;
66     lmcv = LMCVLike(vat_);
67     live = 1;
68 }
```

lmcv/PriceUpdater.sol

Listing 21: lmcv/PriceUpdater.sol (Line 78)

```
77 function updateSource(bytes32 collateral, address _osm) external
↳ auth {
78     osms[collateral] = OSMLike(_osm);
79 }
```

lmcv/RatesUpdater.sol

Listing 22: lmcv/RatesUpdater.sol (Line 51)

```
49 constructor(address lmcvAddress) {
50     wards[msg.sender]   = 1;
51     lmcv                = LMCVLike(lmcvAddress);
52     stabilityRate       = ONE;
53     lastAccrual         = block.timestamp;
54 }
```

lmcv/Liquidator.sol

Listing 23: lmcv/Liquidator.sol (Line 151)

```
149 function setAuctionHouse(address addr) external auth {
150     lmcv.disapprove(address(auctionHouse));
151     auctionHouse = AuctionHouseLike(addr);
152     lmcv.approve(addr);
153 }
```

staking/RewardJoin.sol

Listing 24: staking/RewardJoin.sol (Lines 82,84)

```
79 constructor(address stakingVault_, bytes32 collateralName_,
↳ address collateralContract_) {
80     wards[msg.sender] = 1;
81     live = 1;
82     stakingVault = StakingVaultLike(stakingVault_);
83     collateralName = collateralName_;
84     collateralContract = CollateralLike(collateralContract_);
```

```
85      emit Rely(msg.sender);
86  }
```

staking/StakeJoin.sol

Listing 25: staking/StakeJoin.sol (Lines 80,82)

```
77  constructor(address stakingVault_, bytes32 collateralName_,
↳ address collateralContract_) {
78      wards[msg.sender] = 1;
79      live = 1;
80      stakingVault = StakingVaultLike(stakingVault_);
81      collateralName = collateralName_;
82      collateralContract = CollateralLike(collateralContract_);
83      emit Rely(msg.sender);
84  }
```

staking/StakingVault.sol

Listing 26: staking/StakingVault.sol (Line 91)

```
87  constructor(bytes32 _ddPRIMEBytes, address _ddPRIMEContract,
↳ address _lmcv) {
88      require(_ddPRIMEContract != address(0), "StakingVault/
↳ dPrimeContract address cannot be zero");
89      require(_lmcv != address(0), "StakingVault/LMCV address cannot
↳  be zero");
90      ddPRIMEBytes        = _ddPRIMEBytes;        // bytes32 of
↳ ddPRIME in LMCV for lookup in locked collateral list
91      ddPRIMEContract     = _ddPRIMEContract;     // Address of
↳ ddPRIME for balance lookup
92      lmcv                = _lmcv;
93      stakeLive           = 1;
94      admins[msg.sender]  = 1;
95  }
```

Risk Level:

Likelihood - 3

Impact - 2

Recommendation:

Halborn recommends that validation is added to the setter functions throughout all the smart contracts. At a minimum, the DAMfinance team should ensure that these values cannot be set to zero.

Remediation Plan:

SOLVED: Zero-address checks were added:

- ChainlinkClient.sol: #12
- OSM.sol: #82, 117
- PriceUpdater.sol: #36, 86, 101
- RatesUpdater.sol: #32, 75
- Liquidator.sol: #64, 175
- RewardJoin.sol: #30, 88
- StakeJoin.sol: #28, 86
- StakingVault.sol: #90, 91, 105,

3.10 (HAL-10) MISSING DATA VALIDATION - LOW

Description:

Configuration values in the contracts are not validated; it is possible to set large and/or invalid values, causing contracts to fail.

Code Location:

OSM.sol, #99

The pokeTimeout has no maximum value and can be set to a very large value, making calling poke impossible in a reasonable time.

RatesUpdater.sol, #103

The value of stabilityRate is not validated. When stability rate is set to an invalid value (not per the second rate), for example, percentage: 1.05, calls to accrueInterest will revert during rate calculation in the _rpow function.

AuctionHouse.sol, #96, 100, 104, 108

Bid/auction expirity do not have maximum value.
The minimumBidIncrease and minimumBidDecrease do not have max/min values. Setting minimumBidDecrease >= 1.0 will cause users to be able to bid any value.

Liquidator.sol, #129, 133

The lotSize variable can be set to 0, causing the liquidate function to fail each time because of the require statement:

Listing 27: lmcv/Liquidator.sol (Line 180)

```
178 function liquidate(address user) external {
179     require(live == 1, "Liquidator/Not live");
180     require(liquidationPenalty != 0 && lotSize != 0, "Liquidator/
 ↳ Not set up");
181     require(!lmcv.isWithinCreditLimit(user, lmcv.AccumulatedRate()
 ↳ ), "Liquidator/Vault within credit limit");
182 [...]
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Consider adding validation to avoid setting too large or invalid values.

Remediation Plan:

PARTIALLY SOLVED: The lotSize validation was added in the Liquidator.sol contract.
However, the \client team decided to leave other variables without validation.

3.11 (HAL-11) MISSING EVENTS ON CHANGES - INFORMATIONAL

Description:

Functions performing important changes on tchangePokeTimeout,start,stop,void,updateSource,cage,changeStabilityRate‘ are not emitting events to facilitate monitoring of the protocol.

Code Location:

lmcv/RatesUpdater.sol, #102

lmcv/PriceUpdater.sol, #78, 84

lmcv/OSM.sol, #86, 87, 92, 101, 112

lmcv/AuctionHouse.sol, #92, 96, 100, 104, 108

lmcv/Liquidator.sol, #125, 129, 133, 139, 149

staking/StakingVault.sol, #101, 106, 110

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider adding events for each function performing important changes of the contract configuration.

Remediation Plan:

SOLVED: New events were added, except approval events for StakingVault:

- RatesUpdater.sol: #132
- PriceUpdater.sol: #103, 111
- OSM.sol: #110, 111, 119, 128, 142
- AuctionHouse.sol: #107, 112, 117, 122, 127
- Liquidator.sol: #146, 151, 157, 171, 179
- StakingVault.sol, #169

3.12 (HAL-12) REQUIRE MESSAGE NOT UPDATED - INFORMATIONAL

Description:

Some error messages in require statements were not updated after copying from MakerDAO Jug and other LMCV contracts.

Code Location:

RatesUpdater.sol, #31

staking/StakingVault.sol, #78, 83

staking/RewardJoin.sol, #62, 93, 100

staking/StakeJoin.sol, #60, 91, 98

staking/ddPrimeJoin.sol, #40

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Update the require statement to follow conventions from other DAM functions and contracts.

Remediation Plan:

SOLVED: Messages in require statements were updated:

- RatesUpdater.sol: #48

- StakingVault.sol: #80, 85
- RewardJoin.sol: #70, 103, 110
- StakeJoin.sol: #68, 102, 110
- ddPrimeJoin.sol: #40

3.13 (HAL-13) FUNCTION COULD BE DEFINED AS EXTERNAL - INFORMATIONAL

Description:

The `getLatestPrice()` function could be declared as `external`.

Code Location:

ChainlinkClient.sol, #18

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Use the `external` attribute for the functions never called from the contract.

Remediation Plan:

SOLVED: Function definition was updated from `public` to `external`: ChainlinkClient.sol

3.14 (HAL-14) MISSING NATSPEC DOCUMENTATION - INFORMATIONAL

Description:

Solidity contracts can use a special form of comments to provide rich documentation for functions, return variables, and more. This special form is named the Ethereum Natural Language Specification Format (NatSpec).

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider adding documentation in Natspec format.

Remediation Plan:

PENDING: Natspec documentation is planned for future releases.

3.15 (HAL-15) VARIABLES COULD BE DEFINED AS IMMUTABLE - INFORMATIONAL

Description:

Values of immutable variables can be set inside the constructor, but cannot be modified afterward.
The PriceUpdater and RateUpdater contracts contain the lmcv variable, which is set only in the constructor and never updated. The such variable could be defined as immutable similar to other LMCV contracts.

Code Location:

PriceUpdater:

Listing 28: lmcv/PriceUpdater.sol (Line 48)

```
48 LMCVLike     public lmcv;   // CDP Engine
```

RatesUpdater:

Listing 29: lmcv/RatesUpdater.sol (Line 39)

```
39 LMCVLike     public lmcv;            // LMCV contract
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider defining variables as immutable.

Remediation Plan:

SOLVED: Variable definitions were updated:

- PriceUpdater.sol
- RatesUpdater.sol

3.16 (HAL-16) CHANGING PUBLIC CONSTANT VARIABLES TO PRIVATE CAN SAVE GAS - INFORMATIONAL

Description:

Some constants are public and have a getter function. It is unlikely for these values to be read from the outside. Therefore, it is not necessary to make them public.

Code Location:

LMCVProxy

Listing 30: lmcv/LMCVProxy.sol (Line 46)

```
46 uint256 constant RAY = 10 ** 27;
```

OSM

Listing 31: lmcv/OSM.sol (Line 40)

```
40 uint256     constant    ONE_HOUR        = 3600;
```

StakingVault

Listing 32: staking/StakingVault.sol (Line 112)

```
112 uint256 constant RAY = 10 ** 27;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider defining constants as private.

Remediation Plan:

SOLVED: Constant definitions were updated:

- LMCVProxy.sol
- OSM.sol
- StakingVault.sol

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contract in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contract in the repository and was able to compile it correctly into its ABI and binary format, Slither was run against the contract. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Slither results:

Listing 33

```
OSM.prev(uint256) (contracts/lmcv/OSM.sol#176-179) uses a weak
↳ PRNG: "ts - (ts % pokeTimeout) (contracts/lmcv/OSM.sol#178)"
RatesUpdater._rpow(uint256,uint256,uint256) (contracts/lmcv/
↳ RatesUpdater.sol#64-86) uses a weak PRNG: "
↳ switch_expr_2184_53_20__rpow_asm_0 = n % 2 (contracts/lmcv/
↳ RatesUpdater.sol#68)"
RatesUpdater._rpow(uint256,uint256,uint256) (contracts/lmcv/
↳ RatesUpdater.sol#64-86) uses a weak PRNG: "n % 2 (contracts/lmcv/
↳ RatesUpdater.sol#76-82)"
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#weak-PRNG

StakingVaultLike is re-used:
        - StakingVaultLike (contracts/staking/RewardJoin.sol
↳ #10-14)
        - StakingVaultLike (contracts/staking/StakeJoin.sol#10-13)
        - StakingVaultLike (contracts/staking/ddPrimeJoin.sol
↳ #12-14)
LMCVLike is re-used:
        - LMCVLike (contracts/lmcv/AuctionHouse.sol#7-10)
        - LMCVLike (contracts/lmcv/Liquidator.sol#7-34)
        - LMCVLike (contracts/lmcv/PSM.sol#12-29)
        - LMCVLike (contracts/lmcv/PriceUpdater.sol#18-20)
        - LMCVLike (contracts/lmcv/RatesUpdater.sol#7-10)
```

```
        - LMCVLike (contracts/staking/StakingVault.sol#11-14)
dPrimeJoinLike is re-used:
        - dPrimeJoinLike (contracts/lmcv/PSM.sol#6-10)
CollateralLike is re-used:
        - CollateralLike (contracts/staking/RewardJoin.sol#5-8)
        - CollateralLike (contracts/staking/StakeJoin.sol#5-8)
CollateralJoinLike is re-used:
        - CollateralJoinLike (contracts/lmcv/PSM.sol#38-44)
dPrimeLike is re-used:
        - dPrimeLike (contracts/lmcv/PSM.sol#31-36)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#name-reused

Liquidator.liquidate(address) (contracts/lmcv/Liquidator.sol
↳ #178-253) performs a multiplication on the result of a division:
        -debtHaircut = min(normalizedDebt,lotSize * RAY /
↳ stabilityRate * RAY / liquidationPenalty) (contracts/lmcv/
↳ Liquidator.sol#192)
RatesUpdater._rpow(uint256,uint256,uint256) (contracts/lmcv/
↳ RatesUpdater.sol#64-86) performs a multiplication on the result of
↳  a division:
        -x = xxRound__rpow_asm_0 / b (contracts/lmcv/RatesUpdater.
↳ sol#75)
        -zx__rpow_asm_0 = z * x (contracts/lmcv/RatesUpdater.sol
↳ #77)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#divide-before-multiply

RatesUpdater._rpow(uint256,uint256,uint256) (contracts/lmcv/
↳ RatesUpdater.sol#64-86) uses a dangerous strict equality:
        - switch_expr_2113_41_20__rpow_asm_0 == 0 (contracts/lmcv/
↳ RatesUpdater.sol#66)
RatesUpdater._rpow(uint256,uint256,uint256) (contracts/lmcv/
↳ RatesUpdater.sol#64-86) uses a dangerous strict equality:
        - switch_expr_2184_53_20__rpow_asm_0 == 0 (contracts/lmcv/
↳ RatesUpdater.sol#68)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#dangerous-strict-equalities

Reentrancy in AuctionHouse.converge(uint256,uint256) (contracts/
↳ lmcv/AuctionHouse.sol#207-236):
        External calls:
        - lmcv.moveDPrime(msg.sender,auctions[id].currentWinner,
↳ auctions[id].debtBid) (contracts/lmcv/AuctionHouse.sol#233)
```

```
        State variables written after the call(s):
        - auctions[id].currentWinner = msg.sender (contracts/lmcv/
↳ AuctionHouse.sol#234)
Reentrancy in OSM.poke() (contracts/lmcv/OSM.sol#133-142):
        External calls:
        - (wut,ok) = OracleLike(oracleAddress).peek() (contracts/
↳ lmcv/OSM.sol#135)
        State variables written after the call(s):
        - zzz = prev(block.timestamp) (contracts/lmcv/OSM.sol#139)
Reentrancy in AuctionHouse.raise(uint256,uint256) (contracts/lmcv/
↳ AuctionHouse.sol#169-197):
        External calls:
        - lmcv.moveDPrime(msg.sender,auctions[id].currentWinner,
↳ auctions[id].debtBid) (contracts/lmcv/AuctionHouse.sol#190)
        State variables written after the call(s):
        - auctions[id].currentWinner = msg.sender (contracts/lmcv/
↳ AuctionHouse.sol#191)
Reentrancy in AuctionHouse.raise(uint256,uint256) (contracts/lmcv/
↳ AuctionHouse.sol#169-197):
        External calls:
        - lmcv.moveDPrime(msg.sender,auctions[id].currentWinner,
↳ auctions[id].debtBid) (contracts/lmcv/AuctionHouse.sol#190)
        - lmcv.moveDPrime(msg.sender,auctions[id].treasury,bid -
↳ auctions[id].debtBid) (contracts/lmcv/AuctionHouse.sol#193)
        State variables written after the call(s):
        - auctions[id].debtBid = bid (contracts/lmcv/AuctionHouse.
↳ sol#195)
        - auctions[id].bidExpiry = uint256(block.timestamp) +
↳ bidExpiry (contracts/lmcv/AuctionHouse.sol#196)
Reentrancy in Liquidator.setAuctionHouse(address) (contracts/lmcv/
↳ Liquidator.sol#149-153):
        External calls:
        - lmcv.disapprove(address(auctionHouse)) (contracts/lmcv/
↳ Liquidator.sol#150)
        State variables written after the call(s):
        - auctionHouse = AuctionHouseLike(addr) (contracts/lmcv/
↳ Liquidator.sol#151)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#reentrancy-vulnerabilities-1

OSM.changeOracleAddress(address)._oracleAddress (contracts/lmcv/
↳ OSM.sol#93) lacks a zero-check on :
                - oracleAddress = _oracleAddress (contracts/lmcv/
↳ OSM.sol#95)
```

```
PSM.constructor(address,address,address).treasury_ (contracts/lmcv/PSM.sol#111) lacks a zero-check on :
                - treasury = treasury_ (contracts/lmcv/PSM.sol#119)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

AuctionHouse.start(address,address,uint256,bytes32[],uint256[],uint256,uint256) (contracts/lmcv/AuctionHouse.sol#131-159) has external calls inside a loop: lmcv.moveCollateral(lotList[i],msg.sender,address(this),lotValues[i]) (contracts/lmcv/AuctionHouse.sol#157)
AuctionHouse.converge(uint256,uint256) (contracts/lmcv/AuctionHouse.sol#207-236) has external calls inside a loop: lmcv.moveCollateral(auctions[id].lotList[i],address(this),auctions[id].liqui
dated,portionToReturn) (contracts/lmcv/AuctionHouse.sol#224)
AuctionHouse.end(uint256) (contracts/lmcv/AuctionHouse.sol#243-256) has external calls inside a loop: lmcv.moveCollateral(auctions[id].lotList[i],address(this),auctions[id].currentWinner,rema
iningCollateral) (contracts/lmcv/AuctionHouse.sol#253)
Liquidator.liquidate(address) (contracts/lmcv/Liquidator.sol#178-253) has external calls inside a loop: amount = lmcv.lockedCollateral(user,collateralList[i]) (contracts/lmcv/Liquidator.sol#2
09)
Liquidator.liquidate(address) (contracts/lmcv/Liquidator.sol#178-253) has external calls inside a loop: (spotPrice) = lmcv.CollateralData(collateralList[i]) (contracts/lmcv/Liquidator.sol#214
)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

Reentrancy in OSM.poke() (contracts/lmcv/OSM.sol#133-142):
        External calls:
        - (wut,ok) = OracleLike(oracleAddress).peek() (contracts/lmcv/OSM.sol#135)
        State variables written after the call(s):
        - cur = nxt (contracts/lmcv/OSM.sol#137)
        - nxt = Data(wut,1) (contracts/lmcv/OSM.sol#138)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
```

```
95
96 Reentrancy in RewardJoin.exit(address,uint256) (contracts/staking/
↳ RewardJoin.sol#99-104):
97         External calls:
98         - stakingVault.pullRewards(collateralName,msg.sender,wad)
↳ (contracts/staking/RewardJoin.sol#101)
99         - require(bool,string)(collateralContract.transfer(usr,wad
↳ ),RewardJoin/failed-transfer) (contracts/staking/RewardJoin.sol
↳ #102)
100        Event emitted after the call(s):
101        - Exit(usr,wad) (contracts/staking/RewardJoin.sol#103)
102 Reentrancy in StakeJoin.exit(address,uint256) (contracts/staking/
↳ StakeJoin.sol#97-102):
103        External calls:
104        - stakingVault.pullStakingToken(msg.sender,wad) (contracts
↳ /staking/StakeJoin.sol#99)
105        - require(bool,string)(collateralContract.transfer(usr,wad
↳ ),CollateralJoin/failed-transfer) (contracts/staking/StakeJoin.sol
↳ #100)
106        Event emitted after the call(s):
107        - Exit(usr,wad) (contracts/staking/StakeJoin.sol#101)
108 Reentrancy in ddPrimeJoin.exit(address,uint256) (contracts/staking
↳ /ddPrimeJoin.sol#56-60):
109        External calls:
110        - stakingVault.moveDDPrime(msg.sender,address(this),RAY *
↳ wad) (contracts/staking/ddPrimeJoin.sol#57)
111        - ddPrime.mint(usr,wad) (contracts/staking/ddPrimeJoin.sol
↳ #58)
112        Event emitted after the call(s):
113        - Exit(usr,wad) (contracts/staking/ddPrimeJoin.sol#59)
114 Reentrancy in RewardJoin.join(uint256) (contracts/staking/
↳ RewardJoin.sol#92-97):
115        External calls:
116        - require(bool,string)(collateralContract.transferFrom(msg
↳ .sender,address(this),wad),RewardJoin/failed-transfer) (contracts/
↳ staking/RewardJoin.sol#94)
117        - stakingVault.pushRewards(collateralName,wad) (contracts/
↳ staking/RewardJoin.sol#95)
118        Event emitted after the call(s):
119        - Join(wad) (contracts/staking/RewardJoin.sol#96)
120 Reentrancy in StakeJoin.join(address,uint256) (contracts/staking/
↳ StakeJoin.sol#90-95):
121        External calls:
```

```
122         - require(bool,string)(collateralContract.transferFrom(msg
 ↳ .sender,address(this),wad),CollateralJoin/failed-transfer) (
 ↳ contracts/staking/StakeJoin.sol#92)
123         - stakingVault.pushStakingToken(usr,wad) (contracts/
 ↳ staking/StakeJoin.sol#93)
124         Event emitted after the call(s):
125         - Join(usr,wad) (contracts/staking/StakeJoin.sol#94)
126 Reentrancy in ddPrimeJoin.join(address,uint256) (contracts/staking
 ↳ /ddPrimeJoin.sol#50-54):
127         External calls:
128         - stakingVault.moveDDPrime(address(this),usr,RAY * wad) (
 ↳ contracts/staking/ddPrimeJoin.sol#51)
129         - ddPrime.burn(msg.sender,wad) (contracts/staking/
 ↳ ddPrimeJoin.sol#52)
130         Event emitted after the call(s):
131         - Join(usr,wad) (contracts/staking/ddPrimeJoin.sol#53)
132 Reentrancy in Liquidator.liquidate(address) (contracts/lmcv/
 ↳ Liquidator.sol#178-253):
133         External calls:
134         - lmcv.seize(collateralList,collateralHaircuts,debtHaircut
 ↳ ,user,address(this),lmcv.Treasury()) (contracts/lmcv/Liquidator.
 ↳ sol#219)
135         - id = auctionHouse.start(user,lmcv.Treasury(),
 ↳ askingAmount,collateralList,collateralHaircuts,0,minimumBid) (
 ↳ contracts/lmcv/Liquidator.sol#242-250)
136         Event emitted after the call(s):
137         - Liquidated(collateralList,collateralHaircuts,user,
 ↳ debtHaircut,askingAmount,id) (contracts/lmcv/Liquidator.sol#252)
138 Reentrancy in OSM.poke() (contracts/lmcv/OSM.sol#133-142):
139         External calls:
140         - (wut,ok) = OracleLike(oracleAddress).peek() (contracts/
 ↳ lmcv/OSM.sol#135)
141         Event emitted after the call(s):
142         - LogValue(cur.val) (contracts/lmcv/OSM.sol#140)
143 Reentrancy in PriceUpdater.updatePrice(bytes32) (contracts/lmcv/
 ↳ PriceUpdater.sol#95-101):
144         External calls:
145         - (val,has) = osms[collateral].peek() (contracts/lmcv/
 ↳ PriceUpdater.sol#97)
146         - lmcv.updateSpotPrice(collateral,spot) (contracts/lmcv/
 ↳ PriceUpdater.sol#99)
147         Event emitted after the call(s):
148         - PriceUpdate(collateral,spot) (contracts/lmcv/
 ↳ PriceUpdater.sol#100)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

AuctionHouse.raise(uint256,uint256) (contracts/lmcv/AuctionHouse.sol#169-197) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(auctions[id].bidExpiry > block.timestamp || auctions[id].bidExpiry == 0,AuctionHouse/Bid expiry reached) (contracts/lmcv/AuctionHouse.sol#173)
        - require(bool,string)(auctions[id].auctionExpiry > block.timestamp,AuctionHouse/Auction ended) (contracts/lmcv/AuctionHouse.sol#175)
AuctionHouse.converge(uint256,uint256) (contracts/lmcv/AuctionHouse.sol#207-236) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(auctions[id].bidExpiry > block.timestamp || auctions[id].bidExpiry == 0,AuctionHouse/Bid expiry reached) (contracts/lmcv/AuctionHouse.sol#211)
        - require(bool,string)(auctions[id].auctionExpiry > block.timestamp,AuctionHouse/Auction ended) (contracts/lmcv/AuctionHouse.sol#213)
AuctionHouse.end(uint256) (contracts/lmcv/AuctionHouse.sol#243-256) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(auctions[id].bidExpiry != 0 && (auctions[id].bidExpiry < block.timestamp || auctions[id].auctionExpiry < block.timestamp),AuctionHouse/Auction not finished) (co
ntracts/lmcv/AuctionHouse.sol#245-248)
AuctionHouse.restart(uint256) (contracts/lmcv/AuctionHouse.sol#261-266) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(auctions[id].auctionExpiry < block.timestamp,AuctionHouse/Auction not finished) (contracts/lmcv/AuctionHouse.sol#262)
OSM.pass() (contracts/lmcv/OSM.sol#123-125) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp >= zzz + pokeTimeout (contracts/lmcv/OSM.sol#124)
RatesUpdater._rpow(uint256,uint256,uint256) (contracts/lmcv/RatesUpdater.sol#64-86) uses timestamp for comparisons
        Dangerous comparisons:
```

```
171         - switch_expr_2113_41_20__rpow_asm_0 == 0 (contracts/lmcv/
 ↳ RatesUpdater.sol#66)
172         - switch_expr_2184_53_20__rpow_asm_0 == 0 (contracts/lmcv/
 ↳ RatesUpdater.sol#68)
173 RatesUpdater.accrueInterest() (contracts/lmcv/RatesUpdater.sol
 ↳ #114-121) uses timestamp for comparisons
174         Dangerous comparisons:
175         - require(bool,string)(block.timestamp >= lastAccrual,
 ↳ RatesUpdater/invalid block.timestamp) (contracts/lmcv/RatesUpdater
 ↳ .sol#115)
176 ddPrime.permit(address,address,uint256,uint256,uint8,bytes32,
 ↳ bytes32) (contracts/staking/ddPrime.sol#179-203) uses timestamp
 ↳ for comparisons
177         Dangerous comparisons:
178         - require(bool,string)(block.timestamp <= deadline,ddPrime
 ↳ /permit-expired) (contracts/staking/ddPrime.sol#180)
179 Reference: https://github.com/crytic/slither/wiki/Detector-
 ↳ Documentation#block-timestamp
180
181 RatesUpdater._rpow(uint256,uint256,uint256) (contracts/lmcv/
 ↳ RatesUpdater.sol#64-86) uses assembly
182         - INLINE ASM (contracts/lmcv/RatesUpdater.sol#65-85)
183 StakingVault.either(bool,bool) (contracts/staking/StakingVault.sol
 ↳ #307-309) uses assembly
184         - INLINE ASM (contracts/staking/StakingVault.sol#308)
185 Reference: https://github.com/crytic/slither/wiki/Detector-
 ↳ Documentation#assembly-usage
186
187 Different versions of Solidity are used:
188         - Version used: ['0.8.7', '>=0.4.22<0.9.0', '>=0.8.7',
 ↳ '^0.8.0', '^0.8.7']
189         - 0.8.7 (contracts/lmcv/AuctionHouse.sol#3)
190         - >=0.8.7 (contracts/lmcv/ChainlinkClient.sol#3)
191         - 0.8.7 (contracts/lmcv/Liquidator.sol#3)
192         - >=0.8.7 (contracts/lmcv/OSM.sol#18)
193         - 0.8.7 (contracts/lmcv/PSM.sol#2)
194         - ^0.8.7 (contracts/lmcv/PriceUpdater.sol#16)
195         - 0.8.7 (contracts/lmcv/RatesUpdater.sol#3)
196         - 0.8.7 (contracts/staking/RewardJoin.sol#3)
197         - 0.8.7 (contracts/staking/StakeJoin.sol#3)
198         - 0.8.7 (contracts/staking/StakingVault.sol#9)
199         - 0.8.7 (contracts/staking/ddPrime.sol#5)
200         - 0.8.7 (contracts/staking/ddPrimeJoin.sol#3)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

AuctionHouse (contracts/lmcv/AuctionHouse.sol#12-278) should inherit from AuctionHouseLike (contracts/lmcv/Liquidator.sol#36-46)
OSM (contracts/lmcv/OSM.sol#24-181) should inherit from OracleLike (contracts/lmcv/OSM.sol#20-22)
StakingVault (contracts/staking/StakingVault.sol#20-335) should inherit from StakingVaultLike (contracts/staking/StakeJoin.sol#10-13)
ddPrime (contracts/staking/ddPrime.sol#7-204) should inherit from dPrimeLike (contracts/lmcv/dPrimeJoin.sol#7-10)
ddPrime (contracts/staking/ddPrime.sol#7-204) should inherit from CollateralLike (contracts/staking/RewardJoin.sol#5-8)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-inheritance

Parameter Liquidator.setMinimumAskingPriceVariables(uint256,uint256,uint256)._collateralFactor (contracts/lmcv/Liquidator.sol#140) is not in mixedCase
Parameter Liquidator.setMinimumAskingPriceVariables(uint256,uint256,uint256)._debtFactor (contracts/lmcv/Liquidator.sol#141) is not in mixedCase
Parameter Liquidator.setMinimumAskingPriceVariables(uint256,uint256,uint256)._debtGrossUpFactor (contracts/lmcv/Liquidator.sol#142) is not in mixedCase
Parameter OSM.changeOracleAddress(address)._oracleAddress (contracts/lmcv/OSM.sol#93) is not in mixedCase
Parameter OSM.changePokeTimeout(uint256)._pokeTimeout (contracts/lmcv/OSM.sol#101) is not in mixedCase
Parameter PriceUpdater.updateSource(bytes32,address)._osm (contracts/lmcv/PriceUpdater.sol#77) is not in mixedCase
Parameter PriceUpdater.cage(uint256)._live (contracts/lmcv/PriceUpdater.sol#84) is not in mixedCase
Parameter RatesUpdater.changeStabilityRate(uint256)._stabilityRate (contracts/lmcv/RatesUpdater.sol#103) is not in mixedCase
Contract ddPRIMELike (contracts/staking/StakingVault.sol#16-18) is not in CapWords
Parameter StakingVault.bytes32ToString(bytes32)._bytes32 (contracts/staking/StakingVault.sol#322) is not in mixedCase
Variable StakingVault.RewardTokenList (contracts/staking/StakingVault.sol#35) is not in mixedCase
```

```
221 Variable StakingVault.RewardData (contracts/staking/StakingVault.
 ↳ sol#36) is not in mixedCase
222 Contract ddPrime (contracts/staking/ddPrime.sol#7-204) is not in
 ↳ CapWords
223 Function ddPrime.DOMAIN_SEPARATOR() (contracts/staking/ddPrime.sol
 ↳ #57-59) is not in mixedCase
224 Constant ddPrime.version (contracts/staking/ddPrime.sol#13) is not
 ↳  in UPPER_CASE_WITH_UNDERSCORES
225 Variable ddPrime._DOMAIN_SEPARATOR (contracts/staking/ddPrime.sol
 ↳ #29) is not in mixedCase
226 Contract ddPrimeLike (contracts/staking/ddPrimeJoin.sol#7-10) is
 ↳ not in CapWords
227 Contract ddPrimeJoin (contracts/staking/ddPrimeJoin.sol#16-61) is
 ↳ not in CapWords
228 Reference: https://github.com/crytic/slither/wiki/Detector-
 ↳ Documentation#conformance-to-solidity-naming-conventions
229
230 getLatestPrice() should be declared external:
231         - ChainlinkClient.getLatestPrice() (contracts/lmcv/
 ↳ ChainlinkClient.sol#19-29)
232 bytes32ToString(bytes32) should be declared external:
233         - StakingVault.bytes32ToString(bytes32) (contracts/staking
 ↳ /StakingVault.sol#322-332)
234 Reference: https://github.com/crytic/slither/wiki/Detector-
 ↳ Documentation#public-function-that-could-be-declared-external
```

Slither correctly flagged that:
- some functions can be defined as external
- missing zero address checks

Those issues are included in the findings section of the report.

No major issues found by Slither.

THANK YOU FOR CHOOSING

// HALBORN