// HALBORN

DAMfinance - LMCV part 1

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **August 5th, 2022 - September 6th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	8/31/2022	Pawel Bartunek
0.2	Document Updated	9/07/2022	Pawel Bartunek
0.3	Draft Review	09/07/2022	Kubilay Onur Gungor
0.4	Draft Review	09/07/2022	Gabi Urrutia
1.0	Remediation Plan	10/11/2022	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 August 5th, 2022 and ending on September 6th, 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 four 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:

- CollateralJoinDecimals.sol
- CollateralJoin.sol
- dPrimeJoin.sol
- dPrime.sol
- LMCVProxy.sol
- LMCV.sol
- PSM.sol
- WGLMR.sol

Commit ID: 3391f49ca23e67b2dbb39d35ff7d665dc5769661

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	2	7	6

LIKELIHOOD

IMPACT

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

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - USER FUNDS MAY GET LOCKED	High	SOLVED - 10/10/2022
HAL02 - ADMINS ARE ALLOWED TO BURN USERS DPRIME TOKENS WITHOUT AUTHORIZATION	Medium	FUTURE RELEASE
HAL03 - MULTIPLE INTEGER UNDERFLOWS IN LMCV MODULE	Medium	SOLVED - 10/11/2022
HAL04 - INTEGER UNDERFLOW PSM MODULE	Low	SOLVED - 10/10/2022
HAL05 - CONTRACTS MIGHT LOSE ADMIN FUNCTIONALITY	Low	SOLVED - 10/10/2022
HAL06 - MISSING PAUSE/UNPAUSE FUNCTIONALITY	Low	SOLVED - 10/10/2022
HAL07 - IMPROPER ROLE-BASED ACCESS CONTROL	Low	RISK ACCEPTED
HAL08 - DIVISION BY ZERO IN ISWITHINCREDITLIMIT	Low	SOLVED - 10/10/2022
HAL09 - EDITCOLLATERALLIST BEHAVIOUR MAY BE MISLEADING	Low	SOLVED - 10/10/2022
HAL10 - MISSING ZERO ADDRESS CHECK	Low	SOLVED - 10/10/2022
HAL11 - MISSING EVENTS ON CHANGES	Informational	SOLVED - 10/10/2022
HAL12 - FUNCTIONS COULD BE DECLARED AS EXTERNAL	Informational	SOLVED - 10/10/2022
HAL13 - VARIABLES COULD BE DEFINED AS CONSTANT	Informational	SOLVED - 10/10/2022
HAL14 - COLLATERALLIST SEEMS TO BE UNUSED	Informational	SOLVED - 10/11/2022
HAL15 - MISSING NATSPEC DOCUMENTATION	Informational	FUTURE RELEASE
HAL16 - CHANGE MEMORY TO CALLDATA	Informational	SOLVED - 10/10/2022

FINDINGS & TECH DETAILS

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

Description:

When admin calls thecage function of the CollateralJoin contract, the live flag is set to zero, which means the contract is stopped.

The user can repay the loan, but will not be able to exit (withdraw) assets from the CollateralJoin contract.

Code Location:

cage function:

Listing 1: contracts/CollateralJoin.sol (Line 96)

```
95 function cage() external auth {
96     live = 0;
97     emit Cage();
98 }
```

The exit function, requires contract to be live:

Listing 2: contracts/CollateralJoin.sol (Line 126)

```
125 function exit(address usr, uint256 wad) external {
126     require(live == 1, "CollateralJoin/not-live");
127     lmcv.pullCollateral(collateralName, msg.sender, wad);
128     require(collateralContract.transfer(usr, wad), "CollateralJoin
 ↳ /failed-transfer");
129     emit Exit(usr, wad);
130 }
```

Test scenario:

Hardhat test case:

Listing 3

```
it("HAL-01 User collateral gets locked", async function () {

    // initial collateralJoin balance: 2 users joined with 100 tokens each
    expect(await mockToken.balanceOf(collateralJoin.address)).to.equal(fwad("200"));

    //Total value of collateral: $6000
    //Total loanable amount: $3000
    await userLMCV.loan(collateralBytesList, [fwad("50"), fwad("100"), fwad("200")], fwad("2000"), addr1.address);

    expect(await userLMCV.lockedCollateral(addr1.address, mockTokenBytes)).to.equal(fwad("50"));
    expect(await userLMCV.lockedCollateral(addr1.address, mockToken2Bytes)).to.equal(fwad("100"));
    expect(await userLMCV.lockedCollateral(addr1.address, mockToken3Bytes)).to.equal(fwad("200"));

    expect(await userLMCV.normalizedDebt(addr1.address)).to.equal(fwad("2000"));
    expect(await lmcv.totalDPrime()).to.equal(frad("2000"));
    expect(await lmcv.totalNormalizedDebt()).to.equal(fwad("2000"));

    // lock join contract
    await collateralJoin.cage();

    expect(await await collateralJoin.live()).to.equal(0);

    // repay the loan
    await userLMCV.repay(collateralBytesList, [fwad("50"), fwad("100"), fwad("200")], fwad("2000"), addr1.address);

    // validate the loan was paid - there is no locked collateral, debt nor issued dPrime
    expect(await userLMCV.lockedCollateral(addr1.address, mockTokenBytes)).to.equal(fwad("0"));
    expect(await userLMCV.lockedCollateral(addr1.address, mockToken2Bytes)).to.equal(fwad("0"));
    expect(await userLMCV.lockedCollateral(addr1.address, mockToken3Bytes)).to.equal(fwad("0"));
```

```
30      expect(await userLMCV.normalizedDebt(addr1.address)).to.equal(
↳ fwad("0"));
31      expect(await lmcv.totalDPrime()).to.equal(frad("0"));
32      expect(await lmcv.totalNormalizedDebt()).to.equal(fwad("0"));
33
34      // try to exit from join contract
35      let collatJoinConnect = collateralJoin.connect(addr1);
36      await expect(collatJoinConnect.exit(addr1.address, fwad("100")
↳ ))
37          .to.be.revertedWith("CollateralJoin/not-live");
38
39      // balance does not change
40      expect(await mockToken.balanceOf(collateralJoin.address)).to.
↳ equal(fwad("200"));
41  });
```

Risk Level:

Likelihood - 3
Impact - 5

Recommendation:

Consider allowing users to exit when the contract is stopped (like in CollateralJoinDecimals).

Remediation Plan:

SOLVED: Updated the cage function to allow an admin to re-make the contract. When the administrator resumes the contract, users can withdraw their assets successfully.

Reference: CollateralJoin.sol

3.2 (HAL-02) ADMINS ARE ALLOWED TO BURN USERS DPRIME TOKENS WITHOUT AUTHORIZATION - MEDIUM

Description:

An administrator of the dPrime token can burn users' tokens. Admin burning dPrime might break the LMCV contract as there might be less supply than recorded in the LMCV contract.

Code Location:

Listing 4: contracts/CollateralJoin.sol (Line 159)

```
155 function burn(address from, uint256 value) external {
156     uint256 balance = balanceOf[from];
157     require(balance >= value, "dPrime/insufficient-balance");
158
159     if (from != msg.sender && admins[msg.sender] != 1) {
160         uint256 allowed = allowance[from][msg.sender];
161 [...]
```

Test scenario:

Hardhat test scenario:

Listing 5

```
 1 it("HAL-02 admin burns user dPrime", async function () {
 2     await setupUser(addr1, ["1000", "1000", "1000"]);
 3     await userLMCV.approveMultiple([lmcvProxy.address, dPrimeJoin.
↳ address]);
 4
 5     // lock some collataral of each token
 6     await userLMCVProxy.createLoan(collateralBytesList, [fwad("100
↳ "), fwad("200"), fwad("300")], fwad("1000"));
```

```
 7     expect(await lmcv.totalNormalizedDebt()).to.equal(fwad("1000")
↳ );
 8     expect(await lmcv.normalizedDebt(addr1.address)).to.equal(fwad
↳ ("1000"));
 9
10     expect(await dPrime.balanceOf(addr1.address)).to.eq(fwad("990"
↳ ));
11     expect(await dPrime.totalSupply()).to.eq(fwad("990"));
12
13     dPrime.burn(addr1.address, fwad("500"));
14
15     expect(await lmcv.totalNormalizedDebt()).to.equal(fwad("1000")
↳ );
16     expect(await lmcv.normalizedDebt(addr1.address)).to.equal(fwad
↳ ("1000"));
17     expect(await dPrime.balanceOf(addr1.address)).to.be.eq(fwad("
↳ 490"));
18     expect(await dPrime.totalSupply()).to.eq(fwad("490"));
19 });
```

Risk Level:

Likelihood - 2
Impact - 5

Recommendation:

Administrators should not have the ability to burn user tokens without approval.

Remediation Plan:

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

3.3 (HAL-03) MULTIPLE INTEGER UNDERFLOWS IN LMCV MODULE - MEDIUM

Description:

There are multiple cases in the LMCV contract when subtracting balances without checks. Such behavior may cause the transaction to fail due to arithmetic errors (integer underflow).

Code Location:

The repay function does not perform boundary checks; when the user tries to repay a loan after the interest rate has changed, the transaction may fail with an arithmetic error due to underflow:

Calculating the dPrime amounts and debt:

Listing 6: contracts/LMCV.sol (Lines 406-409)

```
406 dPrime[user]            -= normalizedDebtChange * rateMult;
407 totalDPrime             -= normalizedDebtChange * rateMult;
408 normalizedDebt[user]    -= normalizedDebtChange;
409 totalNormalizedDebt     -= normalizedDebtChange;
410
```

Calculating new collateral amount:

Listing 7: contracts/LMCV.sol (Line 416)

```
415 // Debit locked collateral amount and credit unlocked collateral
 ↳ amount.
416 uint256 newLockedCollateralAmount   = lockedCollateral[user][
 ↳ collateralList[i]]     -= collateralChange[i];
417 uint256 newUnlockedCollateralAmount = unlockedCollateral[user][
 ↳ collateralList[i]]   += collateralChange[i];
```

deflate function calculating protocol deficit:

Listing 8: contracts/LMCV.sol (Lines 522-525)

```
520 function deflate(uint256 rad) external {
521     address u = msg.sender;
522     protocolDeficit[u]      -= rad;
523     totalProtocoldeficit    -= rad;
524     dPrime[u]               -= rad;
525     totalDPrime             -= rad;
526
527     emit Deflate(msg.sender, rad);
528 }
```

pullCollateral function updating unlocked collateral:

Listing 9: contracts/LMCV.sol (Line 275)

```
274 function pullCollateral(bytes32 collat, address user, uint256 wad)
 ↳  external auth {
275     unlockedCollateral[user][collat] -= wad;
276     emit PullCollateral(collat, user, wad);
277 }
```

moveCollateral function updating unlocked collateral:

Listing 10: contracts/LMCV.sol (Line 281)

```
279 function moveCollateral(bytes32 collat, address src, address dst,
 ↳ uint256 wad) external {
280     require(approval(src, msg.sender), "LMCV/collateral move not
 ↳ allowed");
281     unlockedCollateral[src][collat] -= wad;
282     unlockedCollateral[dst][collat] += wad;
283     emit MoveCollateral(collat, src, dst, wad);
284 }
```

moveDPrime function:

Listing 11: contracts/LMCV.sol (Line 292)

```
290 function moveDPrime(address src, address dst, uint256 rad)
 ↳ external {
```

```
291     require(approval(src, msg.sender), "LMCV/dPrime move not
 ↳ allowed");
292     dPrime[src] -= rad;
293     dPrime[dst] += rad;
294     emit MoveDPrime(src, dst, rad);
295 }
```

Test scenarios:

Below are the Hardhat test cases for underflow issues:

Listing 12

```
 1 Issues
 2    HAL-03 Integer underflow in LMCV pullCollateral function - exit
 ↳  twice (38ms)
 3    HAL-03 Integer underflow in LMCV pullCollateral function - exit
 ↳  with more than joined
 4    HAL-03 Integer underflow in LMCV moveCollateral - move more
 ↳ than deposited
 5    HAL-03 Integer underflow in LMCV moveDPrime (62ms)
 6    HAL-03 Integer underflow in LMCV repay - rate updated after
 ↳ loan (98ms)
```

Source code of test cases:

Listing 13

```
 1 it("HAL-03 Integer underflow in LMCV pullCollateral function -
 ↳ exit twice", async function () {
 2
 3     expect(await userLMCV.unlockedCollateral(addr1.address,
 ↳ mockTokenBytes)).to.equal(fwad("100"));
 4
 5     // exit from CollateralJoin
 6     let collatJoinConnect = collateralJoin.connect(addr1);
 7     await collatJoinConnect.exit(addr1.address, fwad("100"));
 8
 9     expect(await userLMCV.unlockedCollateral(addr1.address,
 ↳ mockTokenBytes)).to.equal(fwad("0"));
10
```

```
11      // try to exit for the second time
12      await expect(collatJoinConnect.exit(addr1.address, fwad("100")
↳ ))
13          .to.be.revertedWith("Arithmetic operation underflowed or
↳ overflowed outside of an unchecked block");
14  });
15
16  it("HAL-03 Integer underflow in LMCV pullCollateral function -
↳ exit with more than joined", async function () {
17
18      let collatJoinConnect = collateralJoin.connect(addr1);
19      expect(await userLMCV.unlockedCollateral(addr1.address,
↳ mockTokenBytes)).to.equal(fwad("100"));
20
21      // exit with more than joined
22      await expect(collatJoinConnect.exit(addr1.address, fwad("101")
↳ ))
23          .to.be.revertedWith("Arithmetic operation underflowed or
↳ overflowed outside of an unchecked block");
24
25      expect(await userLMCV.unlockedCollateral(addr1.address,
↳ mockTokenBytes)).to.equal(fwad("100"));
26  });
27
28  it("HAL-03 Integer underflow in LMCV moveCollateral - move more
↳ than deposited", async function () {
29
30      expect(await userLMCV.unlockedCollateral(addr1.address,
↳ mockTokenBytes)).to.equal(fwad("100"));
31      expect(await userLMCV.unlockedCollateral(addr1.address,
↳ mockToken2Bytes)).to.equal(fwad("200"));
32      expect(await userLMCV.unlockedCollateral(addr1.address,
↳ mockToken3Bytes)).to.equal(fwad("300"));
33
34      await expect(userLMCV.moveCollateral(mockTokenBytes, addr1.
↳ address, addr1.address, fwad("200")))
35          .to.be.revertedWith("Arithmetic operation underflowed or
↳ overflowed outside of an unchecked block");
36  });
37
38  it("HAL-03 Integer underflow in LMCV moveDPrime", async function
↳ () {
39
40      await userLMCV.approve(dPrimeJoin.address);
```

```
41      await userLMCV.loan([mockTokenBytes], [fwad("50")], fwad("1000
↳ "), addr1.address);
42      let dPrimeJoinConnect = dPrimeJoin.connect(addr1);
43      await expect(dPrimeJoinConnect.exit(addr1.address, fwad("1001"
↳ )))
44          .to.be.revertedWith("Arithmetic operation underflowed or
↳ overflowed outside of an unchecked block");
45
46      expect(await dPrime.balanceOf(addr1.address)).to.equal(fwad("0
↳ "));
47      expect(await lmcv.totalDPrime()).to.equal(frad("1000"));
48  });
49
50  it("HAL-03 Integer underflow in LMCV repay - rate updated after
↳ loan", async function () {
51
52      // take a loan
53      await userLMCV.loan(collateralBytesList, [fwad("50"), fwad("
↳ 100"), fwad("200")], fwad("2000"), addr1.address);
54      expect(await lmcv.dPrime(addr1.address)).to.equal(frad("2000")
↳ );
55      expect(await userLMCV.normalizedDebt(addr1.address)).to.equal(
↳ fwad("2000"));
56      expect(await userLMCV.dPrime(addr1.address)).to.equal(frad("
↳ 2000"));
57
58      // update rate
59      await lmcv.updateRate(fray(".1"));
60
61      // repay
62      await expect(userLMCV.repay(collateralBytesList, [fwad("50"),
↳ fwad("100"), fwad("200")], fwad("2000"), addr1.address))
63          .to.be.revertedWith("Arithmetic operation underflowed or
↳ overflowed outside of an unchecked block");
64  });
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

Consider adding a validation before calculating the balances to avoid integer underflow.

Remediation Plan:

SOLVED: Added additional require statements to ensure that an underflow does not occur:

LMCV.sol: #275, 282, 294, 409, 420, 530

3.4 (HAL-04) INTEGER UNDERFLOW PSM MODULE - LOW

Description:

PSM contract may revert due to an arithmetic error caused by integer underflow when mintFee is set to a large value.

Code Location:

Listing 14: contracts/PSM.sol (Line 162)

```
158 function createDPrime(address usr, bytes32[] memory collateral,
 ↳ uint256[] memory collatAmount) external {
159     require(collateral.length == 1 && collatAmount.length == 1 &&
 ↳ collateral[0] == collateralName, "PSM/Incorrect setup");
160     uint256 collatAmount18 = collatAmount[0] *
 ↳ to18ConversionFactor; // [wad]
161     uint256 fee = _rmul(collatAmount18, mintFee); // rmul(wad, ray
 ↳ ) = wad
162     uint256 dPrimeAmt = collatAmount18 - fee;
163
164     collateralJoin.join(address(this), collatAmount[0], msg.sender
 ↳ );
165 [...]
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Consider adding a validation of maximum mintFee to avoid integer underflow.

Remediation Plan:

SOLVED: The `require` statement was added to ensure that `mintFee` is less than 100%.

Reference: PSM.sol

3.5 (HAL-05) CONTRACTS MIGHT LOSE ADMIN FUNCTIONALITY - LOW

Description:

The `deny` function is not checking if there are any other active `wards` before setting `wards[usr] = 0`. If the user denies himself, when they are the only `ward`, the contract will lose admin functionality.

Code Location:

`LMCV` module:

Listing 15: contracts/LMCVProxy.sol (Line 97)

```
96 function deny(address usr) external auth {
97     wards[usr] = 0;
98     emit Deny(usr);
99 }
```

`PSM` module:

Listing 16: contracts/PSM.sol (Line 69)

```
68 function deny(address usr) external auth {
69     wards[usr] = 0;
70     emit Deny(usr);
71 }
```

`dPrime` module:

Listing 17: contracts/dPrime.sol (Line 68)

```
67 function deny(address usr) external auth {
68     admins[usr] = 0;
69     emit Deny(usr);
70 }
```

CollateralJoin module:

Listing 18: contracts/CollateralJoin.sol (Line 53)

```
52 function deny(address usr) external auth {
53     wards[usr] = 0;
54     emit Deny(usr);
55 }
```

CollateralJoinDecimals module:

Listing 19: contracts/CollateralJoinDecimals.sol (Line 36)

```
35 function deny(address usr) external auth {
36     wards[usr] = 0;
37     emit Deny(usr);
38 }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Consider adding validation to make sure that there is at least one privileged account left.

Remediation Plan:

SOLVED: The ArchAdmin variable was added to the contract. The address assigned to this field cannot be removed from wards/admins mapping via administrate or deny functions, ensuring 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 admins mapping.

Reference:

- LMCV.sol
- LMCVProxy.sol
- PSM.sol
- dPrime.sol
- CollateralJoin.sol
- CollateralJoinDecimals.sol

3.6 (HAL-06) 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 THOR chain 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).

In case of the contracts in scope, only LMCV and LMCVProxy can be stopped/resumed. Other contracts can only be disabled by the cage function (CollateralJoin and CollateralJoinDecimals) or do not have such possibility at all (PSM, dPrimeJoin).

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 setLive function was added to the contracts. Now all contracts except dPrime and dPrimeJoin can be stopped/resumed in case of an attack.

- CollateralJoin.sol

- CollateralJoinDecimals.sol
- PSM.sol

FINDINGS & TECH DETAILS

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 serious consequences if, for example, a malicious admin decides to take over the platform.

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Halborn recommends that a more granular access control policy is enforced. 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

Remediation Plan:

RISK ACCEPTED: The DAMfinance team accepted the risk of this finding and stated that in this role-based admin structure, only smart contracts would have access to specific roles, and a person-controlled owner address would be able 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) DIVISION BY ZERO IN ISWITHINCREDITLIMIT - LOW

Description:

The isWithinCreditLimit function does not handle leveraged-only collateral properly. When leveraged-only collateral is passed, and the credit value exceeds the collateral value, the transaction fails with a Division or modulo division by zero error.

Code Location:

LMCV module:

Listing 20: contracts/LMCV.sol (Line 603)

```
575 function isWithinCreditLimit(address user, uint256 rate) private
 ↳ view returns (bool) {
576     bytes32[] storage lockedList = lockedCollateralList[user];
577     uint256 creditLimit             = 0; // [rad]
578     uint256 leverTokenCreditLimit   = 0; // [rad]
579     uint256 noLeverageTotal         = 0; // [wad]
580     uint256 leverageTotal           = 0; // [rad]
581     for (uint256 i = 0; i < lockedList.length; i++) {
582         Collateral memory collateralData = CollateralData[
 ↳ lockedList[i]];
583 
584         if(lockedCollateral[user][lockedList[i]] > collateralData.
 ↳ dustLevel){
585             uint256 collateralValue = lockedCollateral[user][
 ↳ lockedList[i]] * collateralData.spotPrice; // wad*ray -> rad
586 
587             if(!collateralData.leveraged){
588                 creditLimit += _rmul(collateralValue,
 ↳ collateralData.creditRatio);
589                 noLeverageTotal += collateralValue / RAY;
590             } else {
591                 leverageTotal += collateralValue;
592                 leverTokenCreditLimit += _rmul(collateralValue,
 ↳ collateralData.creditRatio);
```

```
593             }
594         }
595     }
596
597     // If only leverage tokens exist, just return their credit
 ↳ limit
598     // Keep credit ratio low on levered tokens (60% or lower) to
 ↳ incentivize having non levered collateral in the vault
599     if(noLeverageTotal == 0 && leverageTotal > 0 &&
 ↳ leverTokenCreditLimit >= normalizedDebt[user] * rate){
600         return true;
601     }
602
603     uint256 leverageMultiple = noLeverageTotal == 0 &&
 ↳ leverageTotal == 0 ? RAY : RAY + leverageTotal / noLeverageTotal;
604
605     if (_rmul(creditLimit, leverageMultiple) >= (normalizedDebt[
 ↳ user] * rate)) {
606         return true;
607     }
608     return false;
609 }
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

Consider updating a condition determining the amount of leveraged collateral.

Remediation Plan:

SOLVED: The condition in the isWithinLimit function of the LMCV.sol contract was modified to handle correctly limits for leveraged tokens.

Reference: LMCV.sol

3.9 (HAL-09) EDITCOLLATERALLIST BEHAVIOUR MAY BE MISLEADING - LOW

Description:

The `editCollateralList` function takes three arguments: `bytes32 collateralName, bool accepted, uint256 position`.
When adding collateral, only `collateraName` is used, and `position` is ignored.
When removing collateral, only `position` is used; the function is not validated if a given `collateralName` is located in a specified position.
Moreover, the function does not check if the given collateral is already added to the list.

Code Location:

LMCV module:

Listing 21: contracts/LMCV.sol (Line 238)

```
575 function editCollateralList(bytes32 collateralName, bool accepted,
 ↳  uint256 position) external auth {
576     if(accepted){
577         CollateralList.push(collateralName);
578     }else{
579         deleteElement(CollateralList, position);
580     }
581 }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Consider changing function logic to remove collateral by name, or split it into two separate functions.

Remediation Plan:

SOLVED: The `editCollateralList` function was removed.

Reference: LMCV.sol

3.10 (HAL-10) MISSING ZERO ADDRESS CHECK - LOW

Description:

Code Location:

Following functions are not validating, that given address is different from zero:

- approve function of dPrime.sol
- constructor of CollateralJoin.sol
- constructor of CollateralJoinDecimals.sol
- constructor of PSM.sol

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:

- CollateralJoin.sol
- CollateralJoinDecimals.sol
- PSM.sol

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

Description:

Functions performing important changes on the contract: setLMCV, setDPrimeJoin, setDPrime, and editCollateral are not emitting events to facilitate monitoring of the protocol.

Code Location:

LMCVProxy contract, lines: 75, 80, 85, 106

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

Consider emitting events on the related functions.

Remediation Plan:

SOLVED: New events were added:

- setLMCV
- setDPrimeJoin
- setDPrime
- editCollateral

3.12 (HAL-12) FUNCTIONS COULD BE DECLARED AS EXTERNAL - INFORMATIONAL

Description:

The following functions could be declared as external:

- WGLMR.withdraw(uint256)
- WGLMR.totalSupply()
- WGLMR.approve(address,uint256)
- WGLMR.transfer(address,uint256)

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

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

Remediation Plan:

SOLVED: Function definitions were updated from public to external:

- WGLMR.withdraw(uint256)
- WGLMR.totalSupply()
- WGLMR.approve(address,uint256)
- WGLMR.transfer(address,uint256)

3.13 (HAL-13) VARIABLES COULD BE DEFINED AS CONSTANT - INFORMATIONAL

Description:

The following variables could be defined as constant:

- WGLMR.decimals
- WGLMR.name
- WGLMR.symbol

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Add the constant attributes to state variables that never change.

Remediation Plan:

SOLVED: Variable definitions were updated to constant:

- WGLMR.decimals
- WGLMR.name
- WGLMR.symbol

3.14 (HAL-14) COLLATERALLIST SEEMS TO BE UNUSED - INFORMATIONAL

Description:

The `CollateralList` array seems to be used only in setup tests. There is no use of this array in any contract functions.

Code Location:

LMCV.sol, line 40

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider removing `CollateralList` array and `editCollateralList` function.

Remediation Plan:

SOLVED: The `editCollateralList` function and the `CollateralList` array were removed.

3.15 (HAL-15) 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).

Code Location:

LMCV.sol, LMCVProxy.sol, PSM.sol, WGLMR.sol, dPrimeJoin.sol, dPrime.sol, CollateralJoin.sol, CollateralJoinDecimals.sol

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider adding documentation in Natspec format.

Remediation Plan:

PENDING: Natspec documentation is planned for future releases.

3.16 (HAL-16) CHANGE MEMORY TO CALLDATA - INFORMATIONAL

Description:

It is often more optimal to define parameters as calldata instead of memory for external functions when the parameter is only read.

The following parameters of external functions are stored in memory:
collateralList, collateralChange parameters of loan, repay and liquidate functions in LMCV contract
collaterals and amounts in createLoan and repayLoan functions of LMCVProxy contract

After changing those parameters from memory to calldata, gas usage in unit tests was reduced by around 1000.

Code Location:

LMCV.sol, 316, 389, 462

LMCVProxy.sol 112, 123

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

Consider defining parameter storage as calldata instead of memory in loan, repay, liquidate, createLoan and repayLoan functions.

Remediation Plan:

SOLVED: Function definitions were updated with calldata:

- loan
- repay
- seize
- createLoan
- repayLoan

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 22

```
dPrimeLike is re-used:
        - dPrimeLike (contracts/PSM.sol#31-36)
        - dPrimeLike (contracts/dPrimeJoin.sol#7-10)
LMCVLike is re-used:
        - LMCVLike (contracts/LMCVProxy.sol#24-38)
        - LMCVLike (contracts/PSM.sol#12-29)
        - LMCVLike (contracts/dPrimeJoin.sol#12-15)
CollateralJoinLike is re-used:
        - CollateralJoinLike (contracts/LMCVProxy.sol#14-17)
        - CollateralJoinLike (contracts/PSM.sol#38-44)
dPrimeJoinLike is re-used:
        - dPrimeJoinLike (contracts/LMCVProxy.sol#19-22)
        - dPrimeJoinLike (contracts/PSM.sol#6-10)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#name-reused

dPrimeJoinLike.dPrime().dPrime (contracts/PSM.sol#9) shadows:
        - dPrimeJoinLike.dPrime() (contracts/PSM.sol#9) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#local-variable-shadowing

CollateralJoinDecimals.constructor(address,address,bytes32,address
↳ )._lmcvProxy (contracts/CollateralJoinDecimals.sol#81) lacks a
↳ zero-check on :
```

```
21                 - lmcvProxy = _lmcvProxy (contracts/
↳ CollateralJoinDecimals.sol#89)
22 Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#missing-zero-address-validation
23
24 CollateralJoin.constructor(address,address,bytes32,address).
↳ _lmcvProxy (contracts/CollateralJoin.sol#104) lacks a zero-check
↳ on :
25                 - lmcvProxy = _lmcvProxy (contracts/CollateralJoin
↳ .sol#108)
26 Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#missing-zero-address-validation
27
28 PSM.constructor(address,address,address).treasury_ (contracts/PSM.
↳ sol#111) lacks a zero-check on :
29                 - treasury = treasury_ (contracts/PSM.sol#119)
30 Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#missing-zero-address-validation
31
32 LMCVProxy.createLoan(bytes32[],uint256[],uint256) (contracts/
↳ LMCVProxy.sol#112-121) has external calls inside a loop: require(
↳ bool,string)(ERC20Like(collateralContracts[collaterals[i]]).trans
33 ferFrom(msg.sender,address(this),amounts[i]),LMCVProxy/collateral
↳ transfer failed) (contracts/LMCVProxy.sol#116)
34 LMCVProxy.createLoan(bytes32[],uint256[],uint256) (contracts/
↳ LMCVProxy.sol#112-121) has external calls inside a loop:
↳ CollateralJoinLike(collateralJoins[collaterals[i]]).join(msg.
↳ sender,amoun
35 ts[i]) (contracts/LMCVProxy.sol#117)
36 LMCVProxy.repayLoan(bytes32[],uint256[],uint256) (contracts/
↳ LMCVProxy.sol#123-133) has external calls inside a loop:
↳ CollateralJoinLike(collateralJoins[collaterals[i]]).proxyExit(msg.
↳ sender,a
37 mounts[i]) (contracts/LMCVProxy.sol#131)
38 Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation/#calls-inside-a-loop
39
40 Reentrancy in CollateralJoin.exit(address,uint256) (contracts/
↳ CollateralJoin.sol#125-130):
41         External calls:
42         - lmcv.pullCollateral(collateralName,msg.sender,wad) (
↳ contracts/CollateralJoin.sol#127)
43         - require(bool,string)(collateralContract.transfer(usr,wad
↳ ),CollateralJoin/failed-transfer) (contracts/CollateralJoin.sol
```

```
↳ #128)
        Event emitted after the call(s):
        - Exit(usr,wad) (contracts/CollateralJoin.sol#129)
Reentrancy in CollateralJoin.join(address,uint256) (contracts/
↳ CollateralJoin.sol#118-123):
        External calls:
        - require(bool,string)(collateralContract.transferFrom(msg
↳ .sender,address(this),wad),CollateralJoin/failed-transfer) (
↳ contracts/CollateralJoin.sol#120)
        - lmcv.pushCollateral(collateralName,usr,wad) (contracts/
↳ CollateralJoin.sol#121)
        Event emitted after the call(s):
        - Join(usr,wad) (contracts/CollateralJoin.sol#122)
Reentrancy in CollateralJoin.proxyExit(address,uint256) (contracts
↳ /CollateralJoin.sol#132-137):
        External calls:
        - lmcv.pullCollateral(collateralName,usr,wad) (contracts/
↳ CollateralJoin.sol#134)
        - require(bool,string)(collateralContract.transfer(usr,wad
↳ ),CollateralJoin/failed-transfer) (contracts/CollateralJoin.sol
↳ #135)
        Event emitted after the call(s):
        - Exit(usr,wad) (contracts/CollateralJoin.sol#136)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#reentrancy-vulnerabilities-3

Reentrancy in dPrimeJoin.exit(address,uint256) (contracts/
↳ dPrimeJoin.sol#69-73):
        External calls:
        - lmcv.moveDPrime(msg.sender,address(this),RAY * wad) (
↳ contracts/dPrimeJoin.sol#70)
        - dPrime.mint(usr,wad) (contracts/dPrimeJoin.sol#71)
        Event emitted after the call(s):
        - Exit(usr,wad) (contracts/dPrimeJoin.sol#72)
Reentrancy in dPrimeJoin.join(address,uint256) (contracts/
↳ dPrimeJoin.sol#63-67):
        External calls:
        - lmcv.moveDPrime(address(this),usr,RAY * wad) (contracts/
↳ dPrimeJoin.sol#64)
        - dPrime.burn(msg.sender,wad) (contracts/dPrimeJoin.sol
↳ #65)
        Event emitted after the call(s):
        - Join(usr,wad) (contracts/dPrimeJoin.sol#66)
```

```
Reentrancy in dPrimeJoin.proxyExit(address,uint256) (contracts/dPrimeJoin.sol#75-79):
        External calls:
        - lmcv.moveDPrime(usr,address(this),RAY * wad) (contracts/dPrimeJoin.sol#76)
        - dPrime.mint(usr,wad) (contracts/dPrimeJoin.sol#77)
        Event emitted after the call(s):
        - Exit(usr,wad) (contracts/dPrimeJoin.sol#78)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

dPrime.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (contracts/dPrime.sol#179-203) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= deadline,dPrime/permit-expired) (contracts/dPrime.sol#180)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

LMCV.either(bool,bool) (contracts/LMCV.sol#615-617) uses assembly
        - INLINE ASM (contracts/LMCV.sol#616)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

console._sendLogPayload(bytes) (node_modules/hardhat/console.sol#7-14) uses assembly
        - INLINE ASM (node_modules/hardhat/console.sol#10-13)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Different versions of Solidity are used:
        - Version used: ['^0.8.0', '^0.8.7']
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

```

```
Different versions of Solidity are used:
        - Version used: ['0.8.7', '>=0.4.22<0.9.0']
        - 0.8.7 (contracts/LMCVProxy.sol#3)
        - 0.8.7 (contracts/PSM.sol#2)
        - 0.8.7 (contracts/dPrimeJoin.sol#3)
        - >=0.4.22<0.9.0 (node_modules/hardhat/console.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-
 ↳ Documentation#different-pragma-directives-are-used

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/
 ↳ ERC20/ERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/
 ↳ ERC20/IERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/
 ↳ ERC20/extensions/IERC20Metadata.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/
 ↳ Context.sol#4) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-
 ↳ Documentation#incorrect-versions-of-solidity

Pragma version>=0.4.22<0.9.0 (node_modules/hardhat/console.sol#2)
 ↳ is too complex
Reference: https://github.com/crytic/slither/wiki/Detector-
 ↳ Documentation#incorrect-versions-of-solidity

dPrimeJoin (contracts/dPrimeJoin.sol#17-80) should inherit from
 ↳ CollateralJoinLike (contracts/LMCVProxy.sol#14-17)
Reference: https://github.com/crytic/slither/wiki/Detector-
 ↳ Documentation#missing-inheritance

Parameter CollateralJoinDecimals.join(address,uint256,address).
 ↳ _msgSender (contracts/CollateralJoinDecimals.sol#96) is not in
 ↳ mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-
 ↳ Documentation#conformance-to-solidity-naming-conventions

Contract dPrime (contracts/dPrime.sol#7-204) is not in CapWords
Function dPrime.DOMAIN_SEPARATOR() (contracts/dPrime.sol#57-59) is
 ↳  not in mixedCase
Constant dPrime.version (contracts/dPrime.sol#13) is not in
 ↳ UPPER_CASE_WITH_UNDERSCORES
Variable dPrime._DOMAIN_SEPARATOR (contracts/dPrime.sol#29) is not
 ↳  in mixedCase
```

```
128 Reference: https://github.com/crytic/slither/wiki/Detector-
 ↳ Documentation#conformance-to-solidity-naming-conventions
129
130 Parameter LMCV.setTreasury(address)._treasury (contracts/LMCV.sol
 ↳ #195) is not in mixedCase
131 Parameter LMCV.editLeverageStatus(bytes32,bool)._leveraged (
 ↳ contracts/LMCV.sol#229) is not in mixedCase
132 Parameter LMCV.editAcceptedCollateralType(bytes32,uint256,uint256,
 ↳ uint256,uint256,bool)._lockedAmountLimit (contracts/LMCV.sol#248)
 ↳ is not in mixedCase
133 Parameter LMCV.editAcceptedCollateralType(bytes32,uint256,uint256,
 ↳ uint256,uint256,bool)._dustLevel (contracts/LMCV.sol#249) is not
 ↳ in mixedCase
134 Parameter LMCV.editAcceptedCollateralType(bytes32,uint256,uint256,
 ↳ uint256,uint256,bool)._creditRatio (contracts/LMCV.sol#250) is not
 ↳  in mixedCase
135 Parameter LMCV.editAcceptedCollateralType(bytes32,uint256,uint256,
 ↳ uint256,uint256,bool)._liqBonusMult (contracts/LMCV.sol#251) is
 ↳ not in mixedCase
136 Parameter LMCV.editAcceptedCollateralType(bytes32,uint256,uint256,
 ↳ uint256,uint256,bool)._leveraged (contracts/LMCV.sol#252) is not
 ↳ in mixedCase
137 Variable LMCV.PSMAddresses (contracts/LMCV.sol#24) is not in
 ↳ mixedCase
138 Variable LMCV.CollateralList (contracts/LMCV.sol#40) is not in
 ↳ mixedCase
139 Variable LMCV.CollateralData (contracts/LMCV.sol#41) is not in
 ↳ mixedCase
140 Variable LMCV.ProtocolDebtCeiling (contracts/LMCV.sol#60) is not
 ↳ in mixedCase
141 Variable LMCV.MintFee (contracts/LMCV.sol#61) is not in mixedCase
142 Variable LMCV.AccumulatedRate (contracts/LMCV.sol#62) is not in
 ↳ mixedCase
143 Variable LMCV.Treasury (contracts/LMCV.sol#69) is not in mixedCase
144 Reference: https://github.com/crytic/slither/wiki/Detector-
 ↳ Documentation#conformance-to-solidity-naming-conventions
145
146 Contract dPrimeJoinLike (contracts/LMCVProxy.sol#19-22) is not in
 ↳ CapWords
147 Parameter LMCVProxy.setLMCV(address)._lmcv (contracts/LMCVProxy.
 ↳ sol#75) is not in mixedCase
148 Parameter LMCVProxy.setDPrimeJoin(address)._dPrimeJoin (contracts/
 ↳ LMCVProxy.sol#80) is not in mixedCase
```

```
Parameter LMCVProxy.setDPrime(address)._dPrime (contracts/LMCVProxy.sol#85) is not in mixedCase
Contract dPrimeLike (contracts/PSM.sol#31-36) is not in CapWords
Contract dPrimeJoin (contracts/dPrimeJoin.sol#17-80) is not in CapWords
Contract console (node_modules/hardhat/console.sol#4-1532) is not in CapWords
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Reentrancy in WGLMR.withdraw(uint256) (contracts/WGLMR.sol#38-43):
        External calls:
        - address(msg.sender).transfer(wad) (contracts/WGLMR.sol#41)
        Event emitted after the call(s):
        - Withdrawal(msg.sender,wad) (contracts/WGLMR.sol#42)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4

console.slitherConstructorConstantVariables() (node_modules/hardhat/console.sol#4-1532) uses literals with too many digits:
        - CONSOLE_ADDRESS = address(0x000000000000000000636F6e736F6c652e6c6f67) (node_modules/hardhat/console.sol#5)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

WGLMR.decimals (contracts/WGLMR.sol#21) should be constant
WGLMR.name (contracts/WGLMR.sol#19) should be constant
WGLMR.symbol (contracts/WGLMR.sol#20) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant

name() should be declared external:
        - ERC20.name() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#62-64)
symbol() should be declared external:
        - ERC20.symbol() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#70-72)
decimals() should be declared external:
        - ERC20.decimals() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#87-89)
totalSupply() should be declared external:
```

```
178         - ERC20.totalSupply() (node_modules/@openzeppelin/
 ↳ contracts/token/ERC20/ERC20.sol#94-96)
179 balanceOf(address) should be declared external:
180         - ERC20.balanceOf(address) (node_modules/@openzeppelin/
 ↳ contracts/token/ERC20/ERC20.sol#101-103)
181 transfer(address,uint256) should be declared external:
182         - ERC20.transfer(address,uint256) (node_modules/
 ↳ @openzeppelin/contracts/token/ERC20/ERC20.sol#113-117)
183 approve(address,uint256) should be declared external:
184         - ERC20.approve(address,uint256) (node_modules/
 ↳ @openzeppelin/contracts/token/ERC20/ERC20.sol#136-140)
185 transferFrom(address,address,uint256) should be declared external:
186         - ERC20.transferFrom(address,address,uint256) (
 ↳ node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol
 ↳ #158-167)
187 increaseAllowance(address,uint256) should be declared external:
188         - ERC20.increaseAllowance(address,uint256) (node_modules/
 ↳ @openzeppelin/contracts/token/ERC20/ERC20.sol#181-185)
189 decreaseAllowance(address,uint256) should be declared external:
190         - ERC20.decreaseAllowance(address,uint256) (node_modules/
 ↳ @openzeppelin/contracts/token/ERC20/ERC20.sol#201-210)
191 Reference: https://github.com/crytic/slither/wiki/Detector-
 ↳ Documentation#public-function-that-could-be-declared-external
192
193 withdraw(uint256) should be declared external:
194         - WGLMR.withdraw(uint256) (contracts/WGLMR.sol#38-43)
195 totalSupply() should be declared external:
196         - WGLMR.totalSupply() (contracts/WGLMR.sol#45-47)
197 approve(address,uint256) should be declared external:
198         - WGLMR.approve(address,uint256) (contracts/WGLMR.sol
 ↳ #49-53)
199 transfer(address,uint256) should be declared external:
200         - WGLMR.transfer(address,uint256) (contracts/WGLMR.sol
 ↳ #55-57)
201 Reference: https://github.com/crytic/slither/wiki/Detector-
 ↳ Documentation#public-function-that-could-be-declared-external
202 . analyzed (31 contracts with 78 detectors), 82 result(s) found
```

Slither correctly flagged that:

- some state variables could be declared as constant.
- some functions can be defined as external
- usage of external calls inside a loop
- 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