Debt DAO - P2P Loan

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **July 25th, 2022 - August 12th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	07/27/2022	Grzegorz Trawinski
0.2	Document Update	08/11/2022	Grzegorz Trawinski
0.3	Draft Review	08/14/2022	Kubilay Onur Gungor
0.4	Draft Review	08/16/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Kubilay Onur Gungor	Halborn	kubilay.gungor@halborn.com
Grzegorz Trawinski	Halborn	Grzegorz.Trawinski@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Debt DAO engaged Halborn to conduct a security audit on their smart contracts beginning on July 25th, 2022 and ending on August 12th, 2022. The security assessment was scoped to the smart contracts provided in the GitHub repository debtdao/smart-contracts, ffb66b4. After July 27th, 2022 changed to debtdao/smart-contracts, 955be0c.

1.2 AUDIT SUMMARY

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

The purpose of this audit is to:

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

In summary, Halborn identified several security risks that should be addressed by the Debt DAO team.

1.3 TEST APPROACH & METHODOLOGY

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

- Research into architecture and purpose
- Smart contract manual code review and walkthrough
- Graphing out functionality and contract logic/connectivity/functions. (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes
- Manual testing by custom scripts
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Brownie, Remix IDE, Visual Studio Code)

RISK METHODOLOGY:

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

RISK SCALE - LIKELIHOOD

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

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.

3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

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

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

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

1.4 SCOPE

IN-SCOPE:

The security assessment was scoped to the following debtdao/smart-contracts, 955be0c:

- /utils/MutualUpgrade.sol
- /utils/LoanLib.sol
- /modules/credit/EscrowedLoan.sol
- /modules/credit/BasicEscrowedLoan.sol
- /modules/credit/SpigotedLoan.sol
- /modules/credit/BaseLoan.sol
- /modules/credit/LineOfCredit.sol
- /modules/interest-rate/InterestRateCredit.sol
- /modules/oracle/Oracle.sol
- /modules/escrow/Escrow.sol
- /modules/spigot/Spigot.sol
- /interfaces/ILoan.sol
- /interfaces/IInterestRateCredit.sol
- /interfaces/ITermLoan.sol
- /interfaces/ISpigotedLoan.sol
- /interfaces/IEscrow.sol
- /interfaces/IOracle.sol
- /interfaces/IInterestRateTerm.sol
- /interfaces/ILineOfCredit.sol

Commit ID: 955be0c0652009604383d2fb257c76a2f4e54cb9

OUT-OF-SCOPE:

Other smart contracts in the repository, external libraries and economical attacks.

EXECUTIVE OVERVIEW

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
7	8	6	5	6

LIKELIHOOD

IMPACT

		(HAL-09)		(HAL-01) (HAL-02) (HAL-03) (HAL-04) (HAL-05) (HAL-06) (HAL-07)
(HAL-22)	(HAL-17)	(HAL-20)	(HAL-11) (HAL-13) (HAL-14) (HAL-15)	(HAL-12)
(HAL-24) (HAL-25)		(HAL-16) (HAL-21)	(HAL-19)	(HAL-08) (HAL-10)
	(HAL-26)	(HAL-23)	(HAL-18)	
(HAL-27) (HAL-28) (HAL-29) (HAL-30) (HAL-31) (HAL-32)				

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 - LENDER LIQUIDITY LOCKOUT POSSIBLE VIA DEPOSITANDCLOSE FUNCTION	Critical	SOLVED - 07/26/2022
HAL-02 - DEBT PAY OFF IMPOSSIBLE DUE TO INTEGER UNDERFLOW	Critical	-
HAL-03 - LENDER CAN WITHDRAW INTEREST MULTIPLE TIMES	Critical	-
HAL-04 - ORACLE PRICE AFFECTS THE POSSIBILITY OF DEBT REPAYMENT	Critical	-
HAL-05 - WITHDRAWING ALL LIQUIDITY BEFORE BORROWING CAN DEADLOCK CONTRACT	Critical	-
HAL-06 - SWEEP FUNCTION DOES NOT WORK FOR ARBITER	Critical	-
HAL-07 - COLLATERAL TOKENS LOCKOUT IN ESCROW	Critical	-
HAL-08 - GETOUTSTANDINGDEBT FUNCTION RETURNS UNDERSTATED VALUE	High	-
HAL-09 - BORROWING FROM NON-FIRST POSITION CAN DEADLOCK CONTRACT	High	-
HAL-10 - UPDATEOWNERSPLIT FUNCTION CAN BE ABUSED BY LENDER OR BORROWER	High	-
HAL-11 - UNUSED REVENUE TOKENS LOCKOUT WHILE LOAN IS ACTIVE	High	-
HAL-12 - PAYING OFF DEBT WITH SPIGOT EARNING IN ETHER IS NOT POSSIBLE	High	-
HAL-13 - DOUBLE UPDATE OF UNUSEDTOKENS COLLECTION POSSIBLE	High	-
HAL-14 UNEXPECTED LIQUIDATABLE STATUS IN NEW ESCROWEDLOAN	High	-
HAL-15 CANNOT LIQUIDATE LIQUIDATABLE SECUREDLOAN DUE TO COLLATERAL RATIO CHECK	High	-

HAL-16 - CREDIT CAN BE CLOSED WITHOUT PAYING INTEREST FROM UNUSED FUNDS	Medium	-
HAL-17 - CLOSE FUNCTION CAN BE FRONT-RUN BY LENDER	Medium	-
HAL-18 - UNUSED CREDIT TOKENS LOCKOUT UNTIL NEW REVENUE	Medium	-
HAL-19 - BORROWER CAN CLAIM REVENUE WHILE LOAN IS LIQUIDATABLE	Medium	-
HAL-20 - MINIMUMCOLLATERALRATIO LACKS INPUT VALIDATION	Medium	-
HAL-21 - REVENUE CONTRACT OWNERSHIP LOCKOUT POSSIBLE IN REMOVESPIGOT	Medium	-
HAL-22 - MALICIOUS ARBITER CAN ALLOW OWNERSHIP TRANSFER FUNCTION TO OPERATOR	Low	-
HAL-23 - UPDATEWHITELISTFUNCTION EVENT IS ALWAYS EMITTED WITH TRUE VALUE	Low	-
HAL-24 - BORROWER CAN MINIMIZE DRAWN INTEREST ACCRUING	Low	-
HAL-25 - REMOVESPIGOT DOES NOT CHECK CONTRACT'S BALANCE	Low	-
HAL-26 - INCREASECREDIT FUNCTION LACKS CALL TO SORTINTOQ	Low	-
HAL-27 - GAS OVER-CONSUMPTION IN LOOPS	Informational	-
HAL-28 - UNNEEDED INITIALIZATION OF UINT256 VARIABLES TO 0	Informational	-
HAL-29 - ASSERTIONS LACK MESSAGES	Informational	-
HAL-30 - DEFAULTREVENUESPLIT LACKS INPUT VALIDATION	Informational	-
HAL-31 - UNUSED CODE	Informational	-

HAL-32 - LACK OF CHECK EFFECTS INTERACTIONS PATTERN OR REENTRENCY GUARD	Informational	-

FINDINGS & TECH DETAILS

3.1 (HAL-01) LENDER LIQUIDITY LOCKOUT POSSIBLE VIA DEPOSITANDCLOSE FUNCTION - CRITICAL

Description:

In the LineOfCredit contract, the borrower has two possibilities to pay off the debt: by calling depositAndRepay() and then close() functions, or by calling a single depositAndClose() function.

The assessment revealed that the depositAndClose() does not transfer funds back to the lender, yet it deletes the debt record (using the internal _close function). As a result, the lender's liquidity is locked in the contract.

Listing 1: LineOfCredit.sol (Line 193)

```
function depositAndClose()
    whileBorrowing
    onlyBorrower
    override external
    returns(bool)
  {
    bytes32 id = positionIds[0];
    _accrueInterest(id);

    uint256 totalOwed = debts[id].principal + debts[id].interestAccrued;

    // borrower deposits remaining balance not already repaid and held in contract
    bool success = IERC20(debts[id].token).transferFrom(
      msg.sender,
      address(this),
      totalOwed
    );
    require(success, 'Loan: deposit failed');
    // clear the debt
    _repay(id, totalOwed);

193     require(_close(id));
194     return true;
195   }
```

Listing 2: LineOfCredit.sol (Line 328)

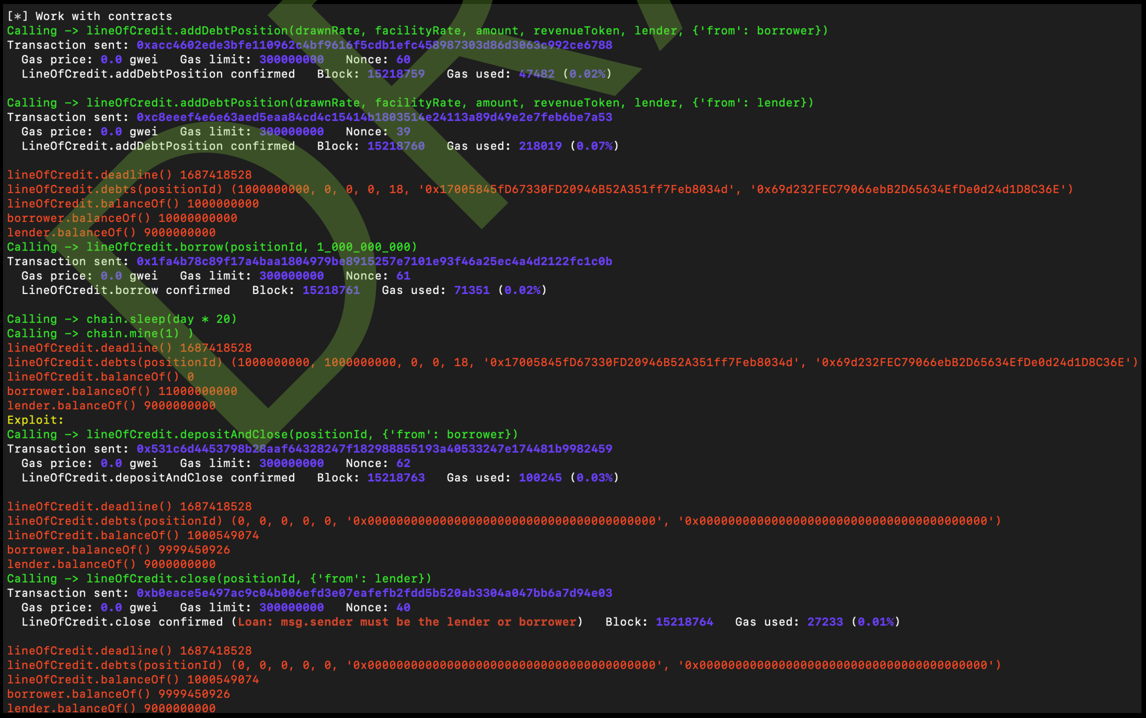
```
318   function close(bytes32 positionId) override external returns(bool) {
319     DebtPosition memory debt = debts[positionId];
320     require(
321       msg.sender == debt.lender ||
322       msg.sender == borrower,
323       "Loan: msg.sender must be the lender or borrower"
324     );
325
326     // return the lender's deposit
327     if(debt.deposit > 0) {
328       require(IERC20(debt.token).transfer(debt.lender, debt.deposit + debt.interestRepaid));
329     }
330
331     require(_close(positionId));
332
333     return true;
334   }
```

Listing 3: LineOfCredit.sol (Line 470)

```
464   function _close(bytes32 positionId) virtual internal returns(bool) {
465     require(
466       debts[positionId].principal + debts[positionId].interestAccrued == 0,
467       'Loan: close failed. debt owed'
468     );
469
470     delete debts[positionId]; // yay gas refunds!!!
471
472     // remove from active list
473     positionIds = LoanLib.removePosition(positionIds, positionId);
474
475     // brick loan contract if all positions closed
476     if(positionIds.length == 0) {
    loanStatus = LoanLib.STATUS.REPAID;
  }

  emit CloseDebtPosition(positionId);

  return true;
}
```

Proof of Concept:

1. All necessary contracts are deployed and initialized: RevenueToken, SimpleOracle, LoanLib, LineOfCredit.
2. As borrower and lender add debt position.
3. As borrower, borrow all deposit.
4. As borrower call depositAndClose to pay the debt.
5. Observe that lender did not receive the liquidity.
6. As the lender attempt to call the close function. Observe that it reverts with the error (Loan: msg.sender must be the lender or borrower).



```
[*] Work with contracts
Calling -> lineOfCredit.addDebtPosition(drawnRate, facilityRate, amount, revenueToken, lender, {'from': borrower})
Transaction sent: 0xacc4602ede3bfe110962c4bf9616f5cdb1efc458987303d86d3063c992ce6788
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 60
  LineOfCredit.addDebtPosition confirmed   Block: 15218759   Gas used: 47482 (0.02%)

Calling -> lineOfCredit.addDebtPosition(drawnRate, facilityRate, amount, revenueToken, lender, {'from': lender})
Transaction sent: 0xc8eeef4e6e63aed5eaa84cd4c15414b1803514e24113a89d49e2e7feb6be7a53
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 39
  LineOfCredit.addDebtPosition confirmed   Block: 15218760   Gas used: 218019 (0.07%)

lineOfCredit.deadline() 1687418528
lineOfCredit.debts(positionId) (1000000000, 0, 0, 0, 18, '0x17005845fD67330FD20946B52A351ff7Feb8034d', '0x69d232FEC79066ebB2D65634EfDe0d24d1D8C36E')
lineOfCredit.balanceOf() 1000000000
borrower.balanceOf() 10000000000
lender.balanceOf() 9000000000
Calling -> lineOfCredit.borrow(positionId, 1_000_000_000)
Transaction sent: 0x1fa4b78c89f17a4baa1804979be8915257e7101e93f46a25ec4a4d2122fc1c0b
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 61
  LineOfCredit.borrow confirmed   Block: 15218761   Gas used: 71351 (0.02%)

Calling -> chain.sleep(day * 20)
Calling -> chain.mine(1) )
lineOfCredit.deadline() 1687418528
lineOfCredit.debts(positionId) (1000000000, 1000000000, 0, 0, 18, '0x17005845fD67330FD20946B52A351ff7Feb8034d', '0x69d232FEC79066ebB2D65634EfDe0d24d1D8C36E')
lineOfCredit.balanceOf() 0
borrower.balanceOf() 11000000000
lender.balanceOf() 9000000000
Exploit:
Calling -> lineOfCredit.depositAndClose(positionId, {'from': borrower})
Transaction sent: 0x531c6d4453798b28aaf64328247f182988855193a40533247e174481b9982459
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 62
  LineOfCredit.depositAndClose confirmed   Block: 15218763   Gas used: 100245 (0.03%)

lineOfCredit.deadline() 1687418528
lineOfCredit.debts(positionId) (0, 0, 0, 0, 0, '0x0000000000000000000000000000000000000000', '0x0000000000000000000000000000000000000000')
lineOfCredit.balanceOf() 1000549074
borrower.balanceOf() 9999450926
lender.balanceOf() 9000000000
Calling -> lineOfCredit.close(positionId, {'from': lender})
Transaction sent: 0xb0eace5e497ac9c04b006efd3e07eafefb2fdd5b520ab3304a047bb6a7d94e03
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 40
  LineOfCredit.close confirmed (Loan: msg.sender must be the lender or borrower)   Block: 15218764   Gas used: 27233 (0.01%)

lineOfCredit.deadline() 1687418528
lineOfCredit.debts(positionId) (0, 0, 0, 0, 0, '0x0000000000000000000000000000000000000000', '0x0000000000000000000000000000000000000000')
lineOfCredit.balanceOf() 1000549074
borrower.balanceOf() 9999450926
lender.balanceOf() 9000000000
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

It is recommended to return lender’s liquidity upon calling the depositAndClose() function.

Remediation Plan:

SOLVED: The Debt DAO solved this issue in commit d739f19d646a2d192aae1e8f56f11e90bbc75 the transfer of lender’s liquidity now happens in internal _close() function which is called by both depositAndClose() and close() functions.

3.2 (HAL-02) DEBT PAY OFF IMPOSSIBLE DUE TO INTEGER UNDERFLOW - CRITICAL

Description:

In the LineOfCredit contract, the borrower has two possibilities to pay off the debt: by calling depositAndRepay() and then close() functions or by calling a single depositAndClose() function.

The assessment revealed that both functions depositAndClose() and depositAndRepay() revert due to integer overflow. The error occurs due to principalUsd parameter subtraction done in _repay function. The principalUsd parameter is supposed to have a non-zero value; however, due to condition check in _createCredit where principal parameter is always 0, the principalUsd parameter is not updated.

Listing 4: LineOfCredit.sol (Line 197)

```
176 function addCredit(
177         uint128 drate,
178         uint128 frate,
179         uint256 amount,
180         address token,
181         address lender
182     )
183         external
184         virtual
185         override
186         whileActive
187         mutualConsent(lender, borrower)
188         returns (bytes32)
189     {
190         bool success = IERC20(token).transferFrom(
191             lender,
192             address(this),
193             amount
194         );
195         require(success, "Loan: no tokens to lend");
196 
197         bytes32 id = _createCredit(lender, token, amount, 0);
```

```

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

    return id;
  }
```

Listing 5: LineOfCredit.sol (Lines 546-549)

```
 function _createCredit(
    address lender,
    address token,
    uint256 amount,
    uint256 principal
  ) internal returns (bytes32 id) {
    id = LoanLib.computePositionId(address(this), lender, token);

    // MUST not double add position. otherwise we can not _close()
    require(
      credits[id].lender == address(0),
      "Loan: position exists"
    );

    (bool passed, bytes memory result) = token.call(
      abi.encodeWithSignature("decimals()")
    );
    uint8 decimals = !passed ? 18 : abi.decode(result, (uint8));

    uint256 value = LoanLib.getValuation(oracle, token, amount, decimals);
    require(value > 0 , "Loan: token cannot be valued");

    credits[id] = Credit({
      lender: lender,
      token: token,
      decimals: decimals,
      deposit: amount,
      principal: principal,
      interestAccrued: 0,
      interestRepaid: 0
    });

```

```
542        ids.push(id); // add lender to end of repayment queue
543
544        emit AddCredit(lender, token, amount, 0);
545
546        if(principal > 0) {
547            principalUsd += value;
548            emit Borrow(id, principal, value);
549        }
550
551        return id;
552    }
```

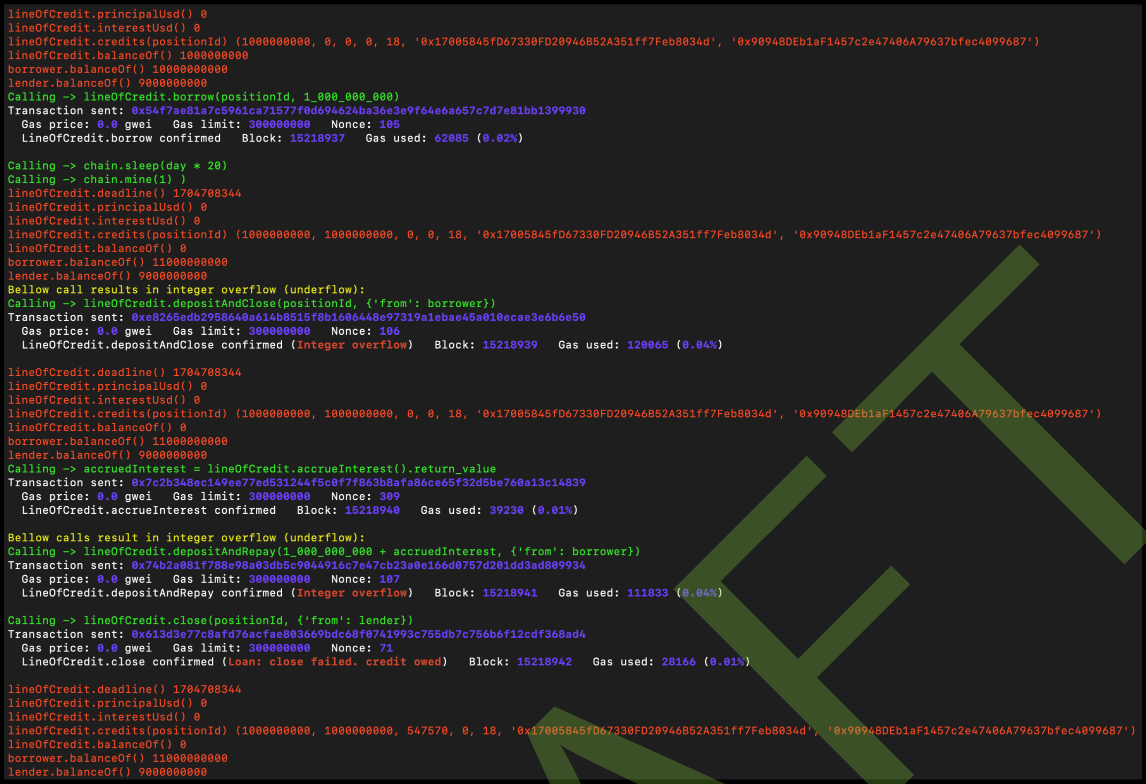
Listing 6: LineOfCredit.sol (Line 587)

```
562  function _repay(bytes32 id, uint256 amount)
563        internal
564        returns (bool)
565    {
566        Credit memory credit = credits[id];
567        int price = oracle.getLatestAnswer(credit.token);
568
569        if (amount <= credit.interestAccrued) {
570            credit.interestAccrued -= amount;
571            uint256 val = LoanLib.calculateValue(price, amount, credit.decimals);
572            interestUsd -= val;
573
574            credit.interestRepaid += amount;
575            emit RepayInterest(id, amount, val);
576        } else {
577            uint256 principalPayment = amount - credit.interestAccrued;
578
579            uint256 iVal = LoanLib.calculateValue(price, credit.interestAccrued, credit.decimals);
580            uint256 pVal = LoanLib.calculateValue(price, principalPayment, credit.decimals);
581
582            emit RepayInterest(id, credit.interestAccrued, iVal);
583            emit RepayPrincipal(id, principalPayment, pVal);
584
585            // update global credit denominated in usd
586            interestUsd -= iVal;
587            principalUsd -= pVal;
```

```

            // update individual credit position denominated in token
            credit.principal -= principalPayment;
            credit.interestRepaid += credit.interestAccrued;
            credit.interestAccrued = 0;

            // if credit fully repaid then remove lender from repayment queue
            if (credit.principal == 0) ids = LoanLib.stepQ(ids);
        }

        credits[id] = credit;

        return true;
    }
```

Proof of Concept:

1. All necessary contracts are deployed and initialized: RevenueToken, SimpleOracle, LoanLib, LineOfCredit.
2. As the borrower and lender, add credit position,
3. As the borrower, borrow() all deposits.
4. As the borrower, attempt to call depositAndClose to pay the debt.
5. Observe that transaction reverts due to integer overflow.
6. As the borrower, attempt to call depositAndRepay to pay the debt.
7. Observe that transaction reverts due to integer overflow.



```
lineOfCredit.principalUsd() 0
lineOfCredit.interestUsd() 0
lineOfCredit.credits(positionId) (1000000000, 0, 0, 0, 18, '0x17005845fD67330FD20946B52A351ff7Feb8034d', '0x90948DEb1aF1457c2e47406A79637bfec4099687')
lineOfCredit.balanceOf() 1000000000
borrower.balanceOf() 10000000000
lender.balanceOf() 9000000000
Calling -> lineOfCredit.borrow(positionId, 1_000_000_000)
Transaction sent: 0x54f7ae81a7c5961ca71577f0d694624ba36e3e9f64e6a657c7d7e81bb1399930
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 105
  LineOfCredit.borrow confirmed   Block: 15218937   Gas used: 62085 (0.02%)

Calling -> chain.sleep(day * 20)
Calling -> chain.mine(1) )
lineOfCredit.deadline() 1704708344
lineOfCredit.principalUsd() 0
lineOfCredit.interestUsd() 0
lineOfCredit.credits(positionId) (1000000000, 1000000000, 0, 0, 18, '0x17005845fD67330FD20946B52A351ff7Feb8034d', '0x90948DEb1aF1457c2e47406A79637bfec4099687')
lineOfCredit.balanceOf() 0
borrower.balanceOf() 11000000000
lender.balanceOf() 9000000000
Bellow call results in integer overflow (underflow):
Calling -> lineOfCredit.depositAndClose(positionId, {'from': borrower})
Transaction sent: 0xe8265edb2958640a614b8515f8b1606448e97319a1ebae45a010ecae3e6b6e50
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 106
  LineOfCredit.depositAndClose confirmed (Integer overflow)   Block: 15218939   Gas used: 120065 (0.04%)

lineOfCredit.deadline() 1704708344
lineOfCredit.principalUsd() 0
lineOfCredit.interestUsd() 0
lineOfCredit.credits(positionId) (1000000000, 1000000000, 0, 0, 18, '0x17005845fD67330FD20946B52A351ff7Feb8034d', '0x90948DEb1aF1457c2e47406A79637bfec4099687')
lineOfCredit.balanceOf() 0
borrower.balanceOf() 11000000000
lender.balanceOf() 9000000000
Calling -> accruedInterest = lineOfCredit.accrueInterest().return_value
Transaction sent: 0x7c2b348ec149ee77ed531244f5c0f7f863b8afa86ce65f32d5be760a13c14839
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 309
  LineOfCredit.accrueInterest confirmed   Block: 15218940   Gas used: 39230 (0.01%)

Bellow calls result in integer overflow (underflow):
Calling -> lineOfCredit.depositAndRepay(1_000_000_000 + accruedInterest, {'from': borrower})
Transaction sent: 0x74b2a081f788e98a03db5c9044916c7e47cb23a0e166d0757d201dd3ad809934
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 107
  LineOfCredit.depositAndRepay confirmed (Integer overflow)   Block: 15218941   Gas used: 111833 (0.04%)

Calling -> lineOfCredit.close(positionId, {'from': lender})
Transaction sent: 0x613d3e77c8afd76acfae803669bdc68f0741993c755db7c756b6f12cdf368ad4
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 71
  LineOfCredit.close confirmed (Loan: close failed. credit owed)   Block: 15218942   Gas used: 28166 (0.01%)

lineOfCredit.deadline() 1704708344
lineOfCredit.principalUsd() 0
lineOfCredit.interestUsd() 0
lineOfCredit.credits(positionId) (1000000000, 1000000000, 547570, 0, 18, '0x17005845fD67330FD20946B52A351ff7Feb8034d', '0x90948DEb1aF1457c2e47406A79637bfec4099687')
lineOfCredit.balanceOf() 0
borrower.balanceOf() 11000000000
lender.balanceOf() 9000000000
```

Risk Level:

Likelihood - 5

Impact - 5

Recommendation:

It is recommended to review and adjust the calculations related to the principalUsd parameter.

3.3 (HAL-03) LENDER CAN WITHDRAW INTEREST MULTIPLE TIMES - CRITICAL

Description:

In the LineOfCredit contract, a lender has the possibility to withdraw accrued interest via withdrawInterest() function. The function does not record the fact of withdrawal; thus, the function can be called multiple times until the contract has a positive token balance.

As a result, in the case of multiple lenders recorded in the contract, one lender can extract liquidity from other lenders.
Alternatively, a lender can pull unborrowed deposits and force borrowers to pay off higher debt than expected, or force default.

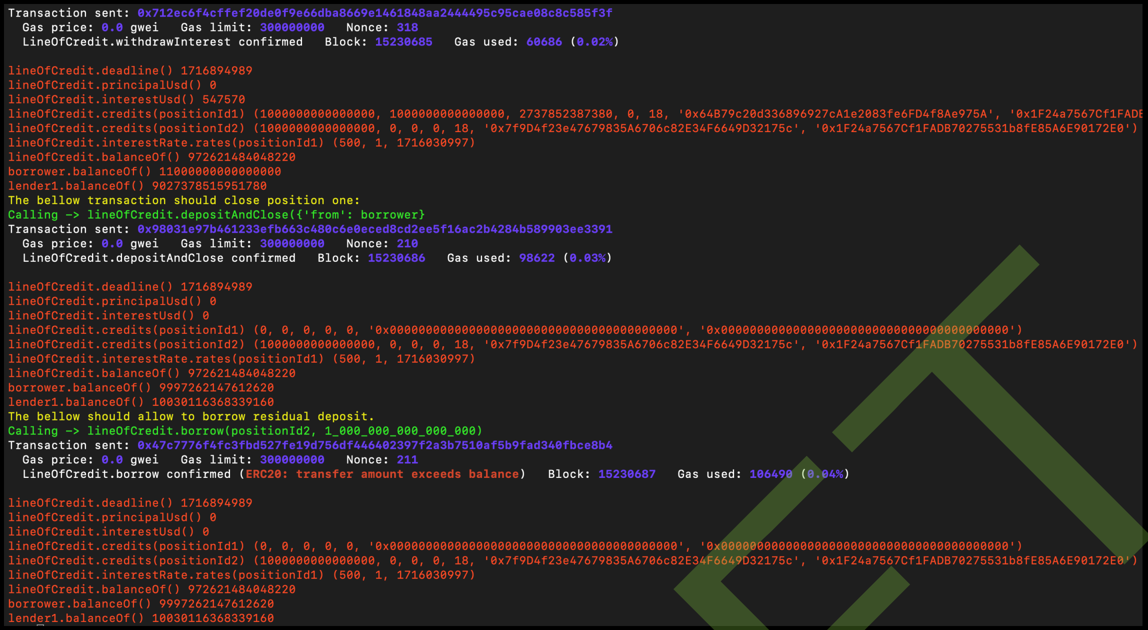
Listing 7: LineOfCredit.sol (Line 587)

```
    function withdrawInterest(bytes32 id)
        external
        override
        returns (uint256)
    {
        require(
            msg.sender == credits[id].lender,
            "Loan: only lender can withdraw"
        );

        _accrueInterest(id);

        uint256 amount = credits[id].interestAccrued;

        bool success = IERC20(credits[id].token).transfer(
            credits[id].lender,
            amount
        );
        require(success, "Loan: withdraw failed");

        emit WithdrawProfit(id, amount);

        return amount;
    }
```

Proof of Concept:

Scenario 1 - steal other lenders' liquidity

1. All necessary contracts are deployed and initialized: RevenueToken, SimpleOracle, LoanLib, LineOfCredit.
2. As borrower and lender1, add credit position for 10_000_000_000_000_000 tokens and drawn rate set to 3000. This action registers the first credit position.
3. As the borrower and the lender2, add credit position for 10_000_000_000_000_000 tokens and drawn rate set to 3000. This action registers the second credit position.
4. As borrower, borrow 10_000_000_000_000_000 tokens from the first position.
5. Forward blockchain time for 20 days.
6. As the lender1, call withdrawInterest for the first credit position ten times.
7. As the borrower, call depositAndClose to pay off the debt and close the first position.
8. As the borrower, attempt to borrow 10_000_000_000_000_000 tokens from the second position.
9. Observe that transaction reverts due to ERC20: transfer amount exceeds balance error. Note that LineOfCredit balance is below 10_000_000_000_000_000.



```
Transaction sent: 0x712ec6f4cffef20de0f9e66dba8669e1461848aa2444495c95cae08c8c585f3f
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 318
  LineOfCredit.withdrawInterest confirmed   Block: 15230685   Gas used: 60686 (0.02%)

lineOfCredit.deadline() 1716894989
lineOfCredit.principalUsd() 0
lineOfCredit.interestUsd() 547570
lineOfCredit.credits(positionId1) (1000000000000000, 1000000000000000, 2737852387380, 0, 18, '0x64B79c20d336896927cA1e2083fe6FD4f8Ae975A', '0x1F24a7567Cf1FADB
lineOfCredit.credits(positionId2) (1000000000000000, 0, 0, 0, 18, '0x7f9D4f23e47679835A6706c82E34F6649D32175c', '0x1F24a7567Cf1FADB70275531b8fE85A6E90172E0')
lineOfCredit.interestRate.rates(positionId1) (500, 1, 1716030997)
lineOfCredit.balanceOf() 972621484048220
borrower.balanceOf() 11000000000000000
lender1.balanceOf() 9027378515951780
The bellow transaction should close position one:
Calling -> lineOfCredit.depositAndClose({'from': borrower}
Transaction sent: 0x98031e97b461233efb663c480c6e0eced8cd2ee5f16ac2b4284b589903ee3391
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 210
  LineOfCredit.depositAndClose confirmed   Block: 15230686   Gas used: 98622 (0.03%)

lineOfCredit.deadline() 1716894989
lineOfCredit.principalUsd() 0
lineOfCredit.interestUsd() 0
lineOfCredit.credits(positionId1) (0, 0, 0, 0, 0, '0x0000000000000000000000000000000000000000', '0x0000000000000000000000000000000000000000')
lineOfCredit.credits(positionId2) (1000000000000000, 0, 0, 0, 18, '0x7f9D4f23e47679835A6706c82E34F6649D32175c', '0x1F24a7567Cf1FADB70275531b8fE85A6E90172E0')
lineOfCredit.interestRate.rates(positionId1) (500, 1, 1716030997)
lineOfCredit.balanceOf() 972621484048220
borrower.balanceOf() 9997262147612620
lender1.balanceOf() 10030116368339160
The bellow should allow to borrow residual deposit.
Calling -> lineOfCredit.borrow(positionId2, 1_000_000_000_000_000)
Transaction sent: 0x47c7776f4fc3fbd527fe19d756df446402397f2a3b7510af5b9fad340fbce8b4
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 211
  LineOfCredit.borrow confirmed (ERC20: transfer amount exceeds balance)   Block: 15230687   Gas used: 106490 (0.04%)

lineOfCredit.deadline() 1716894989
lineOfCredit.principalUsd() 0
lineOfCredit.interestUsd() 0
lineOfCredit.credits(positionId1) (0, 0, 0, 0, 0, '0x0000000000000000000000000000000000000000', '0x0000000000000000000000000000000000000000')
lineOfCredit.credits(positionId2) (1000000000000000, 0, 0, 0, 18, '0x7f9D4f23e47679835A6706c82E34F6649D32175c', '0x1F24a7567Cf1FADB70275531b8fE85A6E90172E0')
lineOfCredit.interestRate.rates(positionId1) (500, 1, 1716030997)
lineOfCredit.balanceOf() 972621484048220
borrower.balanceOf() 9997262147612620
lender1.balanceOf() 10030116368339160
```

Scenario 2 - steal borrower liquidity

1. All necessary contracts are deployed and initialized: RevenueToken, SimpleOracle, LoanLib, LineOfCredit.

2. As borrower and lender1, add credit position for 10_000_000_000_000_000 tokens and drawn rate set to 3000.

4. As borrower, borrow 5_000_000_000_000_000 tokens.

5. Forward blockchain time for 20 days.

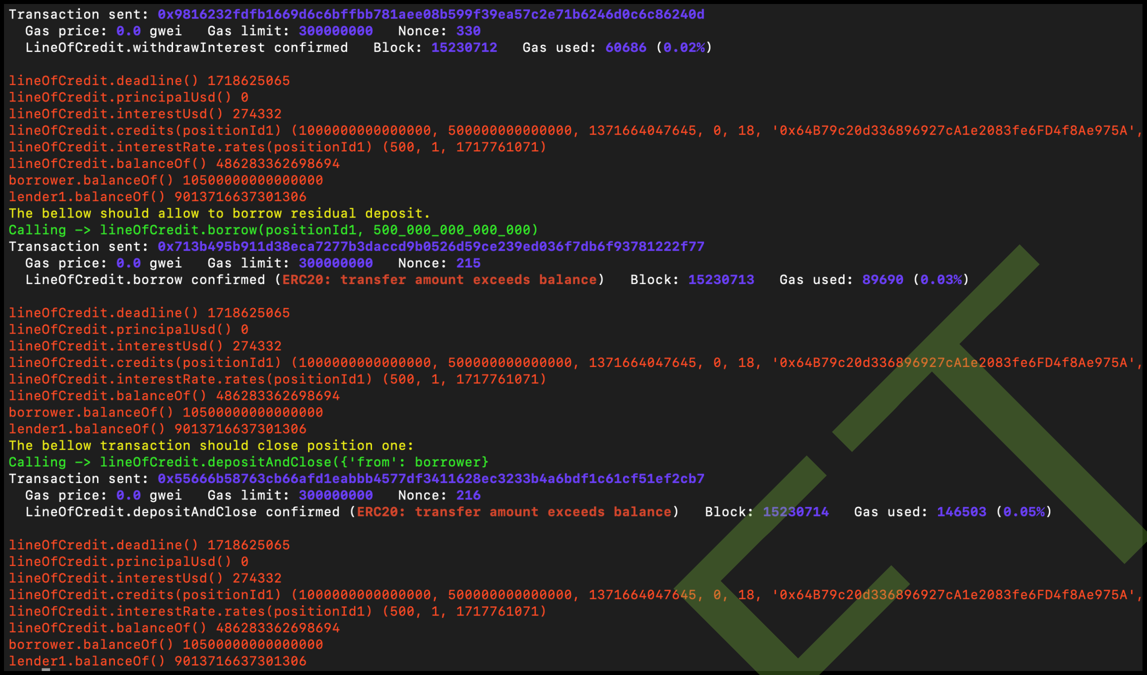
6. As the lender1, call withdrawInterest ten times.

7. As the borrower, attempt to borrow 5_000_000_000_000_000 residual tokens.

8. Observe that transaction reverts due to ERC20: transfer amount exceeds balance error. Note that LineOfCredit balance is below 5_000_000_000_000_000.

9. As the borrower, attempt to call depositAndClose to pay off the debt.

10. Observe that transaction reverts due to ERC20: transfer amount exceeds balance error.



```
Transaction sent: 0x9816232fdfb1669d6c6bffbb781aee08b599f39ea57c2e71b6246d0c6c86240d
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 330
  LineOfCredit.withdrawInterest confirmed   Block: 15230712   Gas used: 60686 (0.02%)

lineOfCredit.deadline() 1718625065
lineOfCredit.principalUsd() 0
lineOfCredit.interestUsd() 274332
lineOfCredit.credits(positionId1) (1000000000000000, 500000000000000, 1371664047645, 0, 18, '0x64B79c20d336896927cA1e2083fe6FD4f8Ae975A',
lineOfCredit.interestRate.rates(positionId1) (500, 1, 1717761071)
lineOfCredit.balanceOf() 486283362698694
borrower.balanceOf() 10500000000000000
lender1.balanceOf() 9013716637301306
The bellow should allow to borrow residual deposit.
Calling -> lineOfCredit.borrow(positionId1, 500_000_000_000_000)
Transaction sent: 0x713b495b911d38eca7277b3daccd9b0526d59ce239ed036f7db6f93781222f77
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 215
  LineOfCredit.borrow confirmed (ERC20: transfer amount exceeds balance)   Block: 15230713   Gas used: 89690 (0.03%)

lineOfCredit.deadline() 1718625065
lineOfCredit.principalUsd() 0
lineOfCredit.interestUsd() 274332
lineOfCredit.credits(positionId1) (1000000000000000, 500000000000000, 1371664047645, 0, 18, '0x64B79c20d336896927cA1e2083fe6FD4f8Ae975A',
lineOfCredit.interestRate.rates(positionId1) (500, 1, 1717761071)
lineOfCredit.balanceOf() 486283362698694
borrower.balanceOf() 10500000000000000
lender1.balanceOf() 9013716637301306
The bellow transaction should close position one:
Calling -> lineOfCredit.depositAndClose({'from': borrower}
Transaction sent: 0x55666b58763cb66afd1eabbb4577df3411628ec3233b4a6bdf1c61cf51ef2cb7
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 216
  LineOfCredit.depositAndClose confirmed (ERC20: transfer amount exceeds balance)   Block: 15230714   Gas used: 146503 (0.05%)

lineOfCredit.deadline() 1718625065
lineOfCredit.principalUsd() 0
lineOfCredit.interestUsd() 274332
lineOfCredit.credits(positionId1) (1000000000000000, 500000000000000, 1371664047645, 0, 18, '0x64B79c20d336896927cA1e2083fe6FD4f8Ae975A',
lineOfCredit.interestRate.rates(positionId1) (500, 1, 1717761071)
lineOfCredit.balanceOf() 486283362698694
borrower.balanceOf() 10500000000000000
lender1.balanceOf() 9013716637301306
```

Risk Level:

Likelihood - 5

Impact - 5

Recommendation:

It is recommended to limit the withdrawal up to the amount of accrued interest so far and update related storage-parameters to prevent subsequent withdrawals.

3.4 (HAL-04) ORACLE PRICE AFFECTS THE POSSIBILITY OF DEBT REPAYMENT - CRITICAL

Description:

The LineOfCredit contract tracks unpaid interest valued in USD by interestUsd parameter. This parameter is updated with addition in _accrueInterest() internal function and subtraction in _repay() internal function. The _accrueInterest() is used by accrueInterest(), setRates(), increaseCredit(), borrow(), withdraw(), and withdrawInterest() functions among the others. The _repay() is used by depositAndRepay() and depositAndClose() functions.

Listing 8: LineOfCredit.sol (Line 631)

```
609     function _accrueInterest(bytes32 id)
610         internal
611         returns (uint256 accruedToken, uint256 accruedValue)
612     {
613         Credit memory credit = credits[id];
614         // get token demoninated interest accrued
615         accruedToken = interestRate.accrueInterest(
616             id,
617             credit.principal,
618             credit.deposit
619         );
620 
621         // update credits balance
622         credit.interestAccrued += accruedToken;
623 
624         // get USD value of interest accrued
625         accruedValue = LoanLib.getValuation(
626             oracle,
627             credit.token,
628             accruedToken,
629             credit.decimals
630         );
631         interestUsd += accruedValue;
632 
633        emit InterestAccrued(id, accruedToken, accruedValue);
634
635        credits[id] = credit; // save updates to intterestAccrued
636
637        return (accruedToken, accruedValue);
638    }
```

Listing 9: LineOfCredit.sol (Lines 572,586)

```
562 function _repay(bytes32 id, uint256 amount)
563        internal
564        returns (bool)
565    {
566        Credit memory credit = credits[id];
567        int price = oracle.getLatestAnswer(credit.token);
568
569        if (amount <= credit.interestAccrued) {
570            credit.interestAccrued -= amount;
571            uint256 val = LoanLib.calculateValue(price, amount,
  ↳ credit.decimals);
572            interestUsd -= val;
573
574            credit.interestRepaid += amount;
575            emit RepayInterest(id, amount, val);
576        } else {
577            uint256 principalPayment = amount - credit.
  ↳ interestAccrued;
578
579            uint256 iVal = LoanLib.calculateValue(price, credit.
  ↳ interestAccrued, credit.decimals);
580            uint256 pVal = LoanLib.calculateValue(price,
  ↳ principalPayment, credit.decimals);
581
582            emit RepayInterest(id, credit.interestAccrued, iVal);
583            emit RepayPrincipal(id, principalPayment, pVal);
584
585            // update global credit denominated in usd
586            interestUsd -= iVal;
587            principalUsd -= pVal;
588
589            // update individual credit position denominated in
  ↳ token
590            credit.principal -= principalPayment;
591            credit.interestRepaid += credit.interestAccrued;
```

```
            credit.interestAccrued = 0;

            // if credit fully repaid then remove lender from
 ↳ repayment queue
            if (credit.principal == 0) ids = LoanLib.stepQ(ids);
        }

        credits[id] = credit;

        return true;
    }
```

The value of interestUsd parameter is strongly affected by the price returned by Oracle. Thus, if the Oracle returns a higher value than previously, an integer underflow occurs in _repay() function making debt repayment impossible. To exploit this vulnerability, _accrueInterest() must be called prior to _repay() to update interestUsd parameter.

Listing 10: LoanLib.sol (Lines 40,55)

```
36
37      /**
38       * @notice         - Gets total valuation for amount of tokens
 ↳  using given oracle.
39       * @dev            - Assumes oracles all return answers in USD
 ↳  with 1e8 decimals
40                         - Does not check if price < 0. HAndled in
 ↳ Oracle or Loan
41       * @param oracle   - oracle contract specified by loan getting
 ↳  valuation
42       * @param token    - token to value on oracle
43       * @param amount   - token amount
44       * @param decimals - token decimals
45       * @return         - total value in usd of all tokens
46       */
47      function getValuation(
48        IOracle oracle,
49        address token,
50        uint256 amount,
51        uint8 decimals
52      )
53        external
```

```
        returns(uint256)
    {
        return _calculateValue(oracle.getLatestAnswer(token), amount, decimals);
    }
```

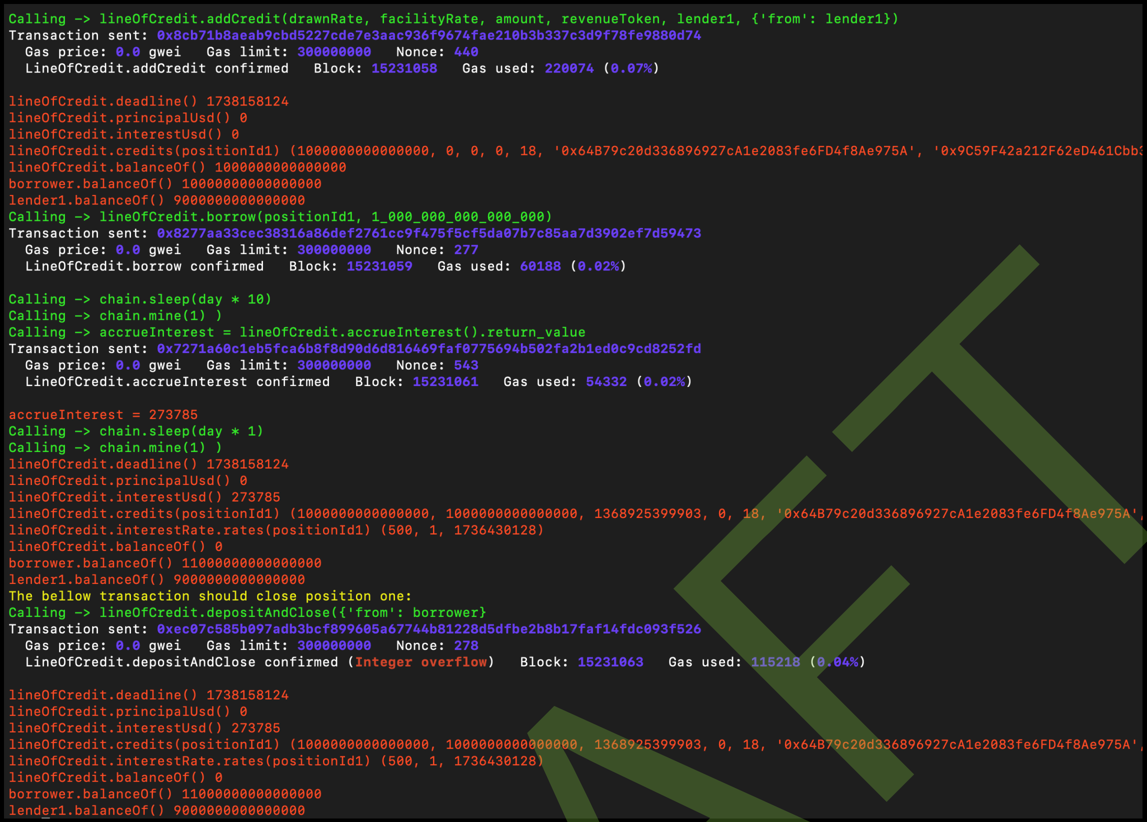
Proof of Concept:

The codebase uses SimpleOracle mock for testing. Based on this contract, the ChangingOracle was prepared that mimics the price increase after ten days.

Listing 11: ChangingOracle.sol (Lines 15,38,49)

```
 1 pragma solidity 0.8.9;
 2 
 3 import { IOracle } from "../interfaces/IOracle.sol";
 4 import { LoanLib } from "../utils/LoanLib.sol";
 5 
 6 contract ChangingOracle is IOracle {
 7 
 8     mapping(address => int) prices;
 9 
10     uint256 public immutable creationTime;
11 
12     constructor(address _supportedToken1, address _supportedToken2) {
13         prices[_supportedToken1] = 1000 * 1e8; // 1000 USD
14         prices[_supportedToken2] = 2000 * 1e8; // 2000 USD
15         creationTime = block.timestamp;
16     }
17 
18     function init() external returns(bool) {
19         return true;
20     }
21 
22     function changePrice(address token, int newPrice) external {
23         prices[token] = newPrice;
24     }
25 
26     function getLatestAnswer(address token) external returns(int256) {
        // mimic eip4626
        // (bool success, bytes memory result) = token.call(abi.encodeWithSignature("asset()"));
        // if(success && result.length > 0) {
        //     // get the underlying token value (if ERC4626)
        //     // NB: Share token to underlying ratio might not be 1:1
        //     token = abi.decode(result, (address));
        // }
        require(prices[token] != 0, "SimpleOracle: unsupported token");

        //simulate price change
        uint256 difference = block.timestamp - creationTime;
        if (difference > 900000) //900000 = 10 days and 10 hours
            return prices[token] * 10001 / 10000;
        return prices[token];
    }

    function healthcheck() external returns (LoanLib.STATUS status) {
        return LoanLib.STATUS.ACTIVE;
    }

    function loan() external returns (address) {
        return address(0);
    }
}
```

1. All necessary contracts are deployed and initialized: RevenueToken, ChangingOracle, LoanLib, LineOfCredit.
2. As the borrower and lender1, add credit position for 1_000_000_000_000_000 tokens.
3. As the borrower, borrow 1_000_000_000_000_000 tokens.
4. Forward blockchain time for 10 days.
5. Call accrueInterest function. Note that the interestUsd parameter value is updated.
6. Forward blockchain time for 1 day. Note that after 11 days, the ChangingOracle will return higher results.
7. As the borrower, attempt to call depositAndClose.
8. Observe that transaction reverts due to integer overflow.



```
Calling -> lineOfCredit.addCredit(drawnRate, facilityRate, amount, revenueToken, lender1, {'from': lender1})
Transaction sent: 0x8cb71b8aeab9cbd5227cde7e3aac936f9674fae210b3b337c3d9f78fe9880d74
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 440
  LineOfCredit.addCredit confirmed   Block: 15231058   Gas used: 220074 (0.07%)

lineOfCredit.deadline() 1738158124
lineOfCredit.principalUsd() 0
lineOfCredit.interestUsd() 0
lineOfCredit.credits(positionId1) (1000000000000000, 0, 0, 0, 18, '0x64B79c20d336896927cA1e2083fe6FD4f8Ae975A', '0x9C59F42a212F62eD461Cbb3
lineOfCredit.balanceOf() 1000000000000000
borrower.balanceOf() 10000000000000000
lender1.balanceOf() 9000000000000000
Calling -> lineOfCredit.borrow(positionId1, 1_000_000_000_000_000)
Transaction sent: 0x8277aa33cec38316a86def2761cc9f475f5cf5da07b7c85aa7d3902ef7d59473
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 277
  LineOfCredit.borrow confirmed   Block: 15231059   Gas used: 60188 (0.02%)

Calling -> chain.sleep(day * 10)
Calling -> chain.mine(1) )
Calling -> accrueInterest = lineOfCredit.accrueInterest().return_value
Transaction sent: 0x7271a60c1eb5fca6b8f8d90d6d816469faf0775694b502fa2b1ed0c9cd8252fd
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 543
  LineOfCredit.accrueInterest confirmed   Block: 15231061   Gas used: 54332 (0.02%)

accrueInterest = 273785
Calling -> chain.sleep(day * 1)
Calling -> chain.mine(1) )
lineOfCredit.deadline() 1738158124
lineOfCredit.principalUsd() 0
lineOfCredit.interestUsd() 273785
lineOfCredit.credits(positionId1) (1000000000000000, 1000000000000000, 1368925399903, 0, 18, '0x64B79c20d336896927cA1e2083fe6FD4f8Ae975A',
lineOfCredit.interestRate.rates(positionId1) (500, 1, 1736430128)
lineOfCredit.balanceOf() 0
borrower.balanceOf() 11000000000000000
lender1.balanceOf() 9000000000000000
The bellow transaction should close position one:
Calling -> lineOfCredit.depositAndClose({'from': borrower}
Transaction sent: 0xec07c585b097adb3bcf899605a67744b81228d5dfbe2b8b17faf14fdc093f526
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 278
  LineOfCredit.depositAndClose confirmed (Integer overflow)   Block: 15231063   Gas used: 115218 (0.04%)

lineOfCredit.deadline() 1738158124
lineOfCredit.principalUsd() 0
lineOfCredit.interestUsd() 273785
lineOfCredit.credits(positionId1) (1000000000000000, 1000000000000000, 1368925399903, 0, 18, '0x64B79c20d336896927cA1e2083fe6FD4f8Ae975A',
lineOfCredit.interestRate.rates(positionId1) (500, 1, 1736430128)
lineOfCredit.balanceOf() 0
borrower.balanceOf() 11000000000000000
lender1.balanceOf() 9000000000000000
```

Risk Level:

Likelihood - 5

Impact - 5

Recommendation:

It is recommended to review and adjust the calculations related to the interestUsd parameter.

3.5 (HAL-05) WITHDRAWING ALL LIQUIDITY BEFORE BORROWING CAN DEADLOCK CONTRACT - CRITICAL

Description:

In the LineOfCredit contract, the lender has the possibility to withdraw all unborrowed deposit previously provided for loan through withdraw() function.

The assessment revealed that withdrawal of all deposits before the borrower borrows any amount, can deadlock the contract. The withdraw() function calls _accrueInterest() functions, so a small amount of facility interest is accrued. Eventually, the borrower can't pay off the debt, close the credit, or release the spigots.
The whileBorrowing() modifier checks if any principal is borrowed; however, it does not check if any interest is accrued. The whileBorrowing() modifier is used both in LineOfCredit and SpigotedLoan contracts in depositAndClose(), depositAndRepay(), claimAndRepay() and claimAndTrade() functions.

Listing 12: LineOfCredit.sol

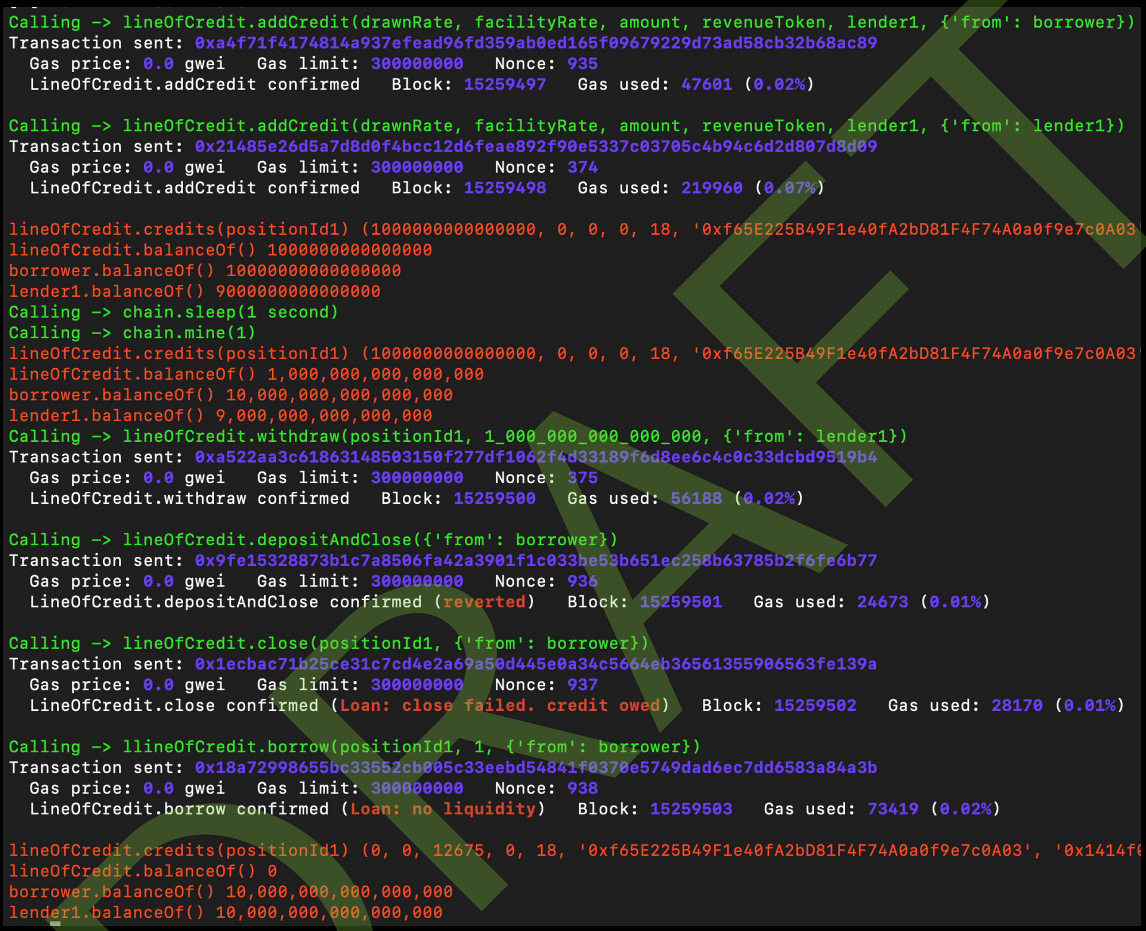
```
    modifier whileBorrowing() {
        require(ids.length > 0 && credits[ids[0]].principal > 0);
        _;
    }
```

Proof of Concept:

1. All necessary contracts are deployed and initialized: RevenueToken, SimpleOracle, LoanLib, LineOfCredit.
2. As borrower and lender1, add credit position for 1_000_000_000_000_000 tokens.
3. As the lender, withdraw() all deposits.
4. As the borrower, attempt to depositAndClose. Observe that the

transaction reverted.

5. As the borrower, attempt to close credit. Observe that the transaction reverted with Loan: close failed. credit owed error.
6. As the borrower, attempt to borrow deposit. Observe that the transaction reverted with Loan: no liquidity error.



```
Calling -> lineOfCredit.addCredit(drawnRate, facilityRate, amount, revenueToken, lender1, {'from': borrower})
Transaction sent: 0xa4f71f4174814a937efead96fd359ab0ed165f09679229d73ad58cb32b68ac89
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 935
  LineOfCredit.addCredit confirmed   Block: 15259497   Gas used: 47601 (0.02%)

Calling -> lineOfCredit.addCredit(drawnRate, facilityRate, amount, revenueToken, lender1, {'from': lender1})
Transaction sent: 0x21485e26d5a7d8d0f4bcc12d6feae892f90e5337c03705c4b94c6d2d807d8d09
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 374
  LineOfCredit.addCredit confirmed   Block: 15259498   Gas used: 219960 (0.07%)

lineOfCredit.credits(positionId1) (1000000000000000, 0, 0, 0, 18, '0xf65E225B49F1e40fA2bD81F4F74A0a0f9e7c0A03'
lineOfCredit.balanceOf() 1000000000000000
borrower.balanceOf() 10000000000000000
lender1.balanceOf() 9000000000000000
Calling -> chain.sleep(1 second)
Calling -> chain.mine(1)
lineOfCredit.credits(positionId1) (1000000000000000, 0, 0, 0, 18, '0xf65E225B49F1e40fA2bD81F4F74A0a0f9e7c0A03'
lineOfCredit.balanceOf() 1,000,000,000,000,000
borrower.balanceOf() 10,000,000,000,000,000
lender1.balanceOf() 9,000,000,000,000,000
Calling -> lineOfCredit.withdraw(positionId1, 1_000_000_000_000_000, {'from': lender1})
Transaction sent: 0xa522aa3c61863148503150f277df1062f4d33189f6d8ee6c4c0c33dcbd9519b4
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 375
  LineOfCredit.withdraw confirmed   Block: 15259500   Gas used: 56188 (0.02%)

Calling -> lineOfCredit.depositAndClose({'from': borrower})
Transaction sent: 0x9fe15328873b1c7a8506fa42a3901f1c033be53b651ec258b63785b2f6fe6b77
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 936
  LineOfCredit.depositAndClose confirmed (reverted)   Block: 15259501   Gas used: 24673 (0.01%)

Calling -> lineOfCredit.close(positionId1, {'from': borrower})
Transaction sent: 0x1ecbac71b25ce31c7cd4e2a69a50d445e0a34c5664eb36561355906563fe139a
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 937
  LineOfCredit.close confirmed (Loan: close failed. credit owed)   Block: 15259502   Gas used: 28170 (0.01%)

Calling -> llineOfCredit.borrow(positionId1, 1, {'from': borrower})
Transaction sent: 0x18a72998655bc33552cb005c33eebd54841f0370e5749dad6ec7dd6583a84a3b
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 938
  LineOfCredit.borrow confirmed (Loan: no liquidity)   Block: 15259503   Gas used: 73419 (0.02%)

lineOfCredit.credits(positionId1) (0, 0, 12675, 0, 18, '0xf65E225B49F1e40fA2bD81F4F74A0a0f9e7c0A03', '0x1414f
lineOfCredit.balanceOf() 0
borrower.balanceOf() 10,000,000,000,000,000
lender1.balanceOf() 10,000,000,000,000,000
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

It is recommended to adjust whileBorrowing() modifier to verify if both interestAccrued and principal parameters are above 0.

3.6 (HAL-06) SWEEP FUNCTION DOES NOT WORK FOR ARBITER - CRITICAL

Description:

The SpigotedLoan contract implements a fallback mechanism to withdraw all unused funds from spigots in case of the borrower default. The sweep() function can be called to send all unused funds (based on unusedTokens collection) to the arbiter when the loan has defaulted and the status is set to INSOLVENT. However, the INSOLVENT status is never assigned to the loan in the solution, whereas the loan can have LIQUIDATABLE status assigned e.g. in healthcheck() function when the debt deadline has passed.

Listing 13: SpigotedLoan.sol (Lines 261,270)

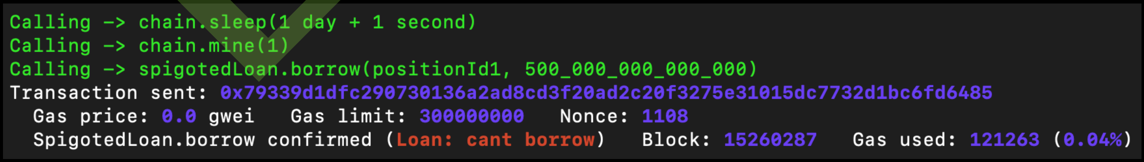
```
261    * @notice - sends unused tokens to borrower if repaid or
 ↳ arbiter if liquidatable
262             -  doesnt send tokens out if loan is unpaid but
 ↳ healthy
263    * @dev    - callable by anyone
264    * @param token - token to take out
265   */
266     function sweep(address token) external returns (uint256) {
267         if (loanStatus == LoanLib.STATUS.REPAID) {
268             return _sweep(borrower, token);
269         }
270         if (loanStatus == LoanLib.STATUS.INSOLVENT) {
271             return _sweep(arbiter, token);
272         }
273 
274         return 0;
275     }
276 
277     function _sweep(address to, address token) internal returns (
 ↳ uint256 x) {
278         x = unusedTokens[token];
279         if (token == address(0)) {
280             payable(to).transfer(x);
281         } else {
282             require(IERC20(token).transfer(to, x));
283         }
```

```
        delete unusedTokens[token];
    }
```

As a result, all unused revenue and credit tokens stored in SpigotedLoan (unusedTokens collection) are locked in the contract. The credit token can be transferred to the lender using claimAndRepay() function, unless the spigot is still owned by the SpigotedLoan contract, and it is providing new revenue. On the other hand, the revenue token is locked permanently.

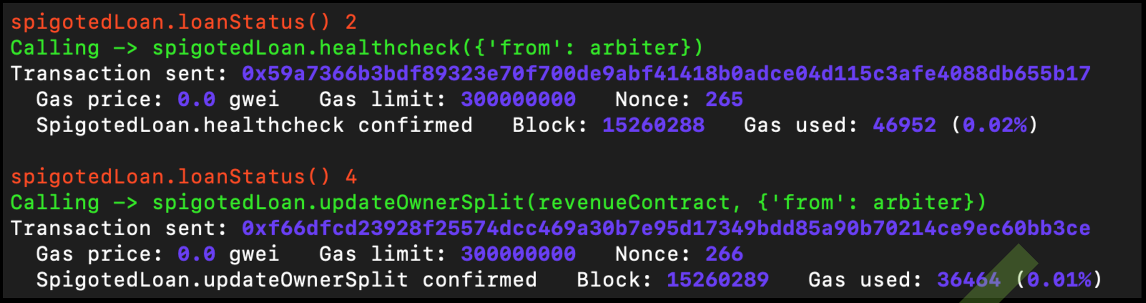
Proof of Concept:

1. All necessary contracts are deployed and initialized: CreditToken, RevenueToken, SimpleOracle, LoanLib, SpigotedLoan, SimpleRevenueContract. Set the ttl parameter to 1 day.
2. As the borrower and lender1, add credit position for 10_000_000_000_000_000 tokens.
3. As the borrower, borrow() half of the deposit - 5_000_000_000_000_000.
4. As th borrower and arbiter, add new spigot with addSpigot() function and RevenueContract as input.
5. As the borrower, transfer the ownership of RevenueContract contract to the SpigotController.
6. Mint 500_000_000_000_000 revenue tokens to the RevenueContract contract to simulate token gain.
7. Forward blockchain time for 1 day and 1 second.
8. As the borrower, attempt to borrow() the remaining deposit. Observe that the transaction reverted with Loan: can't borrow error.



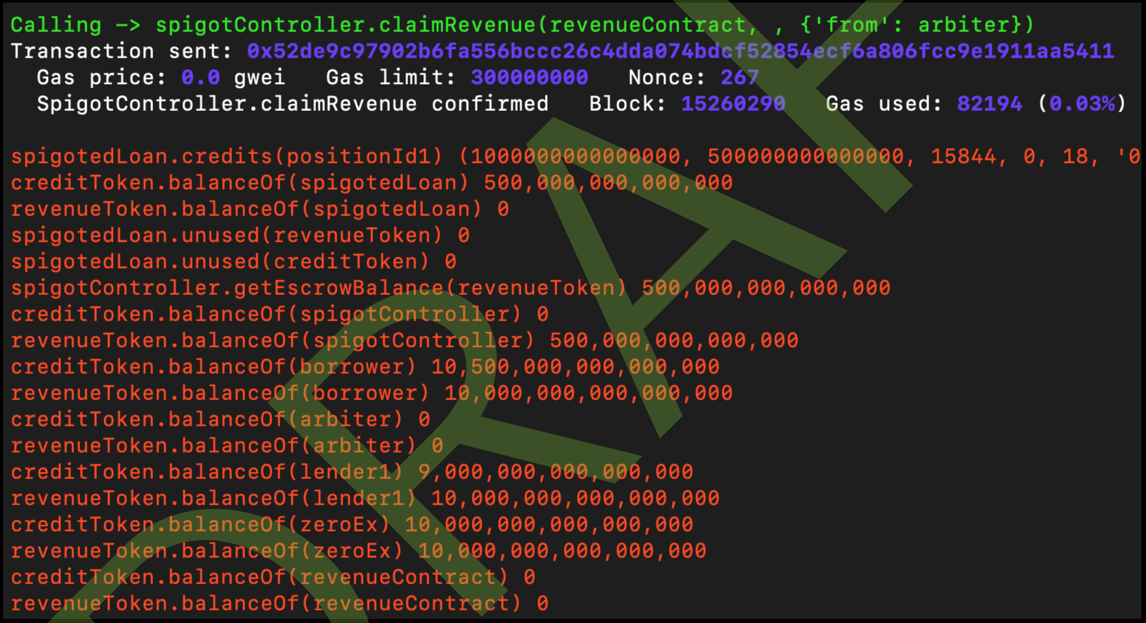
```
Calling -> chain.sleep(1 day + 1 second)
Calling -> chain.mine(1)
Calling -> spigotedLoan.borrow(positionId1, 500_000_000_000_000)
Transaction sent: 0x79339d1dfc290730136a2ad8cd3f20ad2c20f3275e31015dc7732d1bc6fd6485
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 1108
  SpigotedLoan.borrow confirmed (Loan: cant borrow)   Block: 15260287   Gas used: 121263 (0.04%)
```

7. As the arbiter, call healthcheck() function. Note that the loan’s status changed from ACTIVE to LIQUIDATABLE.
8. As the arbiter, call updateOwnerSplit so 100% of revenue will go to the SpigotController contract.



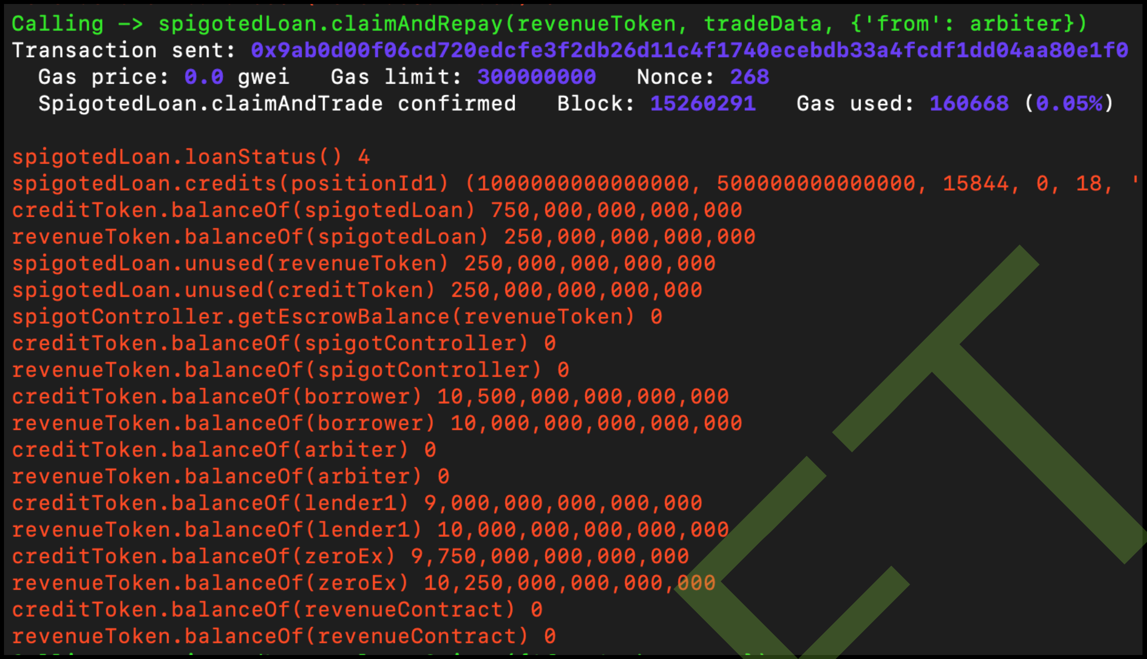
```
spigotedLoan.loanStatus() 2
Calling -> spigotedLoan.healthcheck({'from': arbiter})
Transaction sent: 0x59a7366b3bdf89323e70f700de9abf41418b0adce04d115c3afe4088db655b17
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 265
  SpigotedLoan.healthcheck confirmed   Block: 15260288   Gas used: 46952 (0.02%)

spigotedLoan.loanStatus() 4
Calling -> spigotedLoan.updateOwnerSplit(revenueContract, {'from': arbiter})
Transaction sent: 0xf66dfcd23928f25574dcc469a30b7e95d17349bdd85a90b70214ce9ec60bb3ce
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 266
  SpigotedLoan.updateOwnerSplit confirmed   Block: 15260289   Gas used: 36464 (0.01%)
```

9. As the arbiter, call claimRevenue() in SpigotController contract to claim revenue tokens. Note that 100% of tokens (550_000_000_000_000) are transferred to the SpigotController.



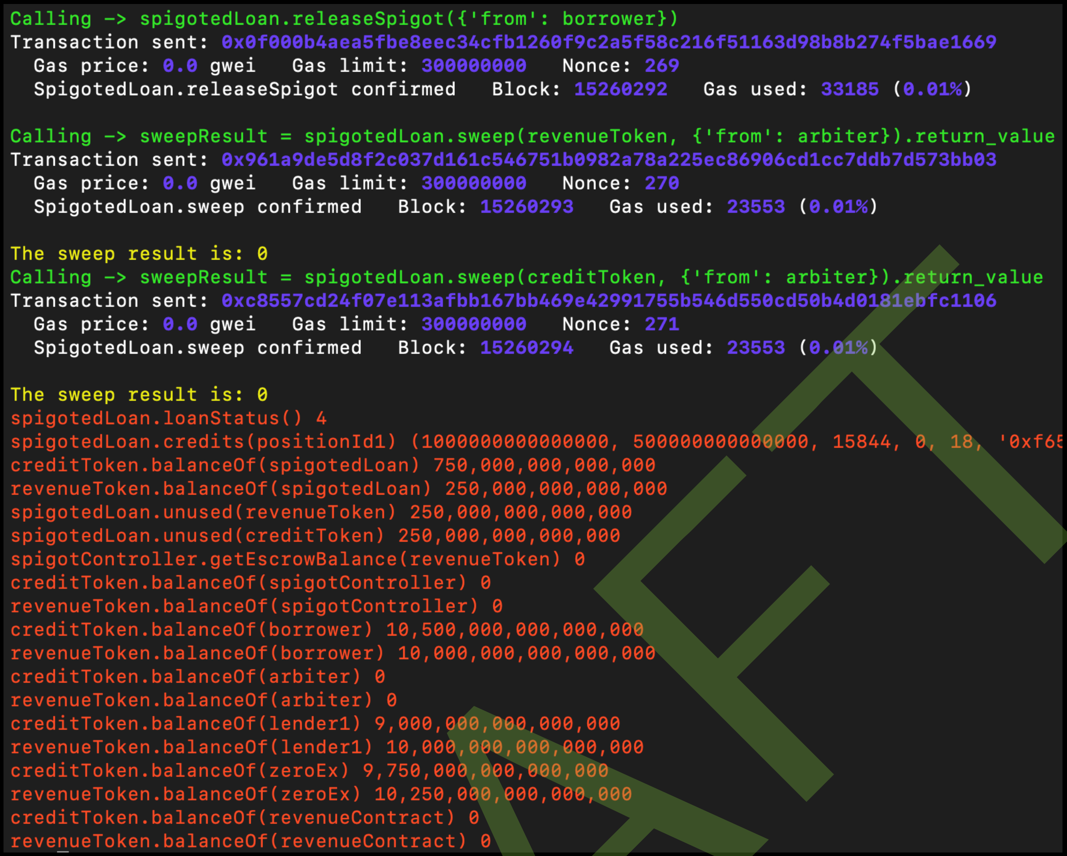
```
Calling -> spigotController.claimRevenue(revenueContract, , {'from': arbiter})
Transaction sent: 0x52de9c97902b6fa556bccc26c4dda074bdcf52854ecf6a806fcc9e1911aa5411
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 267
  SpigotController.claimRevenue confirmed   Block: 15260290   Gas used: 82194 (0.03%)

spigotedLoan.credits(positionId1) (1000000000000000, 500000000000000, 15844, 0, 18, '0
creditToken.balanceOf(spigotedLoan) 500,000,000,000,000
revenueToken.balanceOf(spigotedLoan) 0
spigotedLoan.unused(revenueToken) 0
spigotedLoan.unused(creditToken) 0
spigotController.getEscrowBalance(revenueToken) 500,000,000,000,000
creditToken.balanceOf(spigotController) 0
revenueToken.balanceOf(spigotController) 500,000,000,000,000
creditToken.balanceOf(borrower) 10,500,000,000,000,000
revenueToken.balanceOf(borrower) 10,000,000,000,000,000
creditToken.balanceOf(arbiter) 0
revenueToken.balanceOf(arbiter) 0
creditToken.balanceOf(lender1) 9,000,000,000,000,000
revenueToken.balanceOf(lender1) 10,000,000,000,000,000
creditToken.balanceOf(zeroEx) 10,000,000,000,000,000
revenueToken.balanceOf(zeroEx) 10,000,000,000,000,000
creditToken.balanceOf(revenueContract) 0
revenueToken.balanceOf(revenueContract) 0
```

10. As the arbiter, call claimAndRepay() in SpigotedLoan contract to trade escrowed revenue tokens and pay off part of the debt. As an input, trade exchange data provide 250_000_000_000_000 revenue tokens that should be exchanged for credit tokens.
11. Observe the SpigotedLoan balances. Note that the contract has 750,000,000,000,000 credit tokens and 250,000,000,000,000 revenue tokens. Also, 250,000,000,000,000 credit tokens and 250,000,000,000,000 revenue tokens are stored in unusedTokens collection.



```
Calling -> spigotedLoan.claimAndRepay(revenueToken, tradeData, {'from': arbiter})
Transaction sent: 0x9ab0d00f06cd720edcfe3f2db26d11c4f1740ecebdb33a4fcdf1dd04aa80e1f0
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 268
  SpigotedLoan.claimAndTrade confirmed   Block: 15260291   Gas used: 160668 (0.05%)

spigotedLoan.loanStatus() 4
spigotedLoan.credits(positionId1) (1000000000000000, 500000000000000, 15844, 0, 18, '
creditToken.balanceOf(spigotedLoan) 750,000,000,000,000
revenueToken.balanceOf(spigotedLoan) 250,000,000,000,000
spigotedLoan.unused(revenueToken) 250,000,000,000,000
spigotedLoan.unused(creditToken) 250,000,000,000,000
spigotController.getEscrowBalance(revenueToken) 0
creditToken.balanceOf(spigotController) 0
revenueToken.balanceOf(spigotController) 0
creditToken.balanceOf(borrower) 10,500,000,000,000,000
revenueToken.balanceOf(borrower) 10,000,000,000,000,000
creditToken.balanceOf(arbiter) 0
revenueToken.balanceOf(arbiter) 0
creditToken.balanceOf(lender1) 9,000,000,000,000,000
revenueToken.balanceOf(lender1) 10,000,000,000,000,000
creditToken.balanceOf(zeroEx) 9,750,000,000,000,000
revenueToken.balanceOf(zeroEx) 10,250,000,000,000,000
creditToken.balanceOf(revenueContract) 0
revenueToken.balanceOf(revenueContract) 0
```

12. As arbiter, call sweep() function with revenue token address as an input. Note that the function returns a 0 value.
13. As arbiter, call sweep() function with credit token address as an input. Note that the function returns a 0 value.
14. Observe that SpigotedLoan balances remain unchanged.



```
Calling -> spigotedLoan.releaseSpigot({'from': borrower})
Transaction sent: 0x0f000b4aea5fbe8eec34cfb1260f9c2a5f58c216f51163d98b8b274f5bae1669
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 269
  SpigotedLoan.releaseSpigot confirmed   Block: 15260292   Gas used: 33185 (0.01%)

Calling -> sweepResult = spigotedLoan.sweep(revenueToken, {'from': arbiter}).return_value
Transaction sent: 0x961a9de5d8f2c037d161c546751b0982a78a225ec86906cd1cc7ddb7d573bb03
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 270
  SpigotedLoan.sweep confirmed   Block: 15260293   Gas used: 23553 (0.01%)

The sweep result is: 0
Calling -> sweepResult = spigotedLoan.sweep(creditToken, {'from': arbiter}).return_value
Transaction sent: 0xc8557cd24f07e113afbb167bb469e42991755b546d550cd50b4d0181ebfc1106
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 271
  SpigotedLoan.sweep confirmed   Block: 15260294   Gas used: 23553 (0.01%)

The sweep result is: 0
spigotedLoan.loanStatus() 4
spigotedLoan.credits(positionId1) (1000000000000000, 500000000000000, 15844, 0, 18, '0xf65
creditToken.balanceOf(spigotedLoan) 750,000,000,000,000
revenueToken.balanceOf(spigotedLoan) 250,000,000,000,000
spigotedLoan.unused(revenueToken) 250,000,000,000,000
spigotedLoan.unused(creditToken) 250,000,000,000,000
spigotController.getEscrowBalance(revenueToken) 0
creditToken.balanceOf(spigotController) 0
revenueToken.balanceOf(spigotController) 0
creditToken.balanceOf(borrower) 10,500,000,000,000,000
revenueToken.balanceOf(borrower) 10,000,000,000,000,000
creditToken.balanceOf(arbiter) 0
revenueToken.balanceOf(arbiter) 0
creditToken.balanceOf(lender1) 9,000,000,000,000,000
revenueToken.balanceOf(lender1) 10,000,000,000,000,000
creditToken.balanceOf(zeroEx) 9,750,000,000,000,000
revenueToken.balanceOf(zeroEx) 10,250,000,000,000,000
creditToken.balanceOf(revenueContract) 0
revenueToken.balanceOf(revenueContract) 0
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

It is recommended that sweep() function checks loan status against LIQUIDATABLE value instead of INSOLVENT.

3.7 (HAL-07) COLLATERAL TOKENS LOCKOUT IN ESCROW - CRITICAL

Description:

In the Escrow contract, the releaseCollateral() function allows the borrower to withdraw collateral with the assumption that the remaining collateral is still above the minimum threshold. When the debt is paid off, the borrower should be allowed to withdraw all remaining collateral. However, the assessment revealed that withdraw of the remaining collateral is not possible.

Listing 14: Escrow.sol (Lines 217-220)

```
203     function releaseCollateral(
204         uint256 amount,
205         address token,
206         address to
207     ) external returns (uint256) {
208         require(amount > 0, "Escrow: amount is 0");
209         require(msg.sender == borrower, "Escrow: only borrower can
  ↳ call");
210         require(
211             deposited[token].amount >= amount,
212             "Escrow: insufficient balance"
213         );
214         deposited[token].amount -= amount;
215         require(IERC20(token).transfer(to, amount));
216         uint256 cratio = _getLatestCollateralRatio();
217         require(
218             cratio >= minimumCollateralRatio,
219             "Escrow: cannot release collateral if cratio becomes
  ↳ lower than the minimum"
220         );
221         emit RemoveCollateral(token, amount);
222 
223         return cratio;
224     }
```

When the last part of the collateral is released, the releaseCollateral

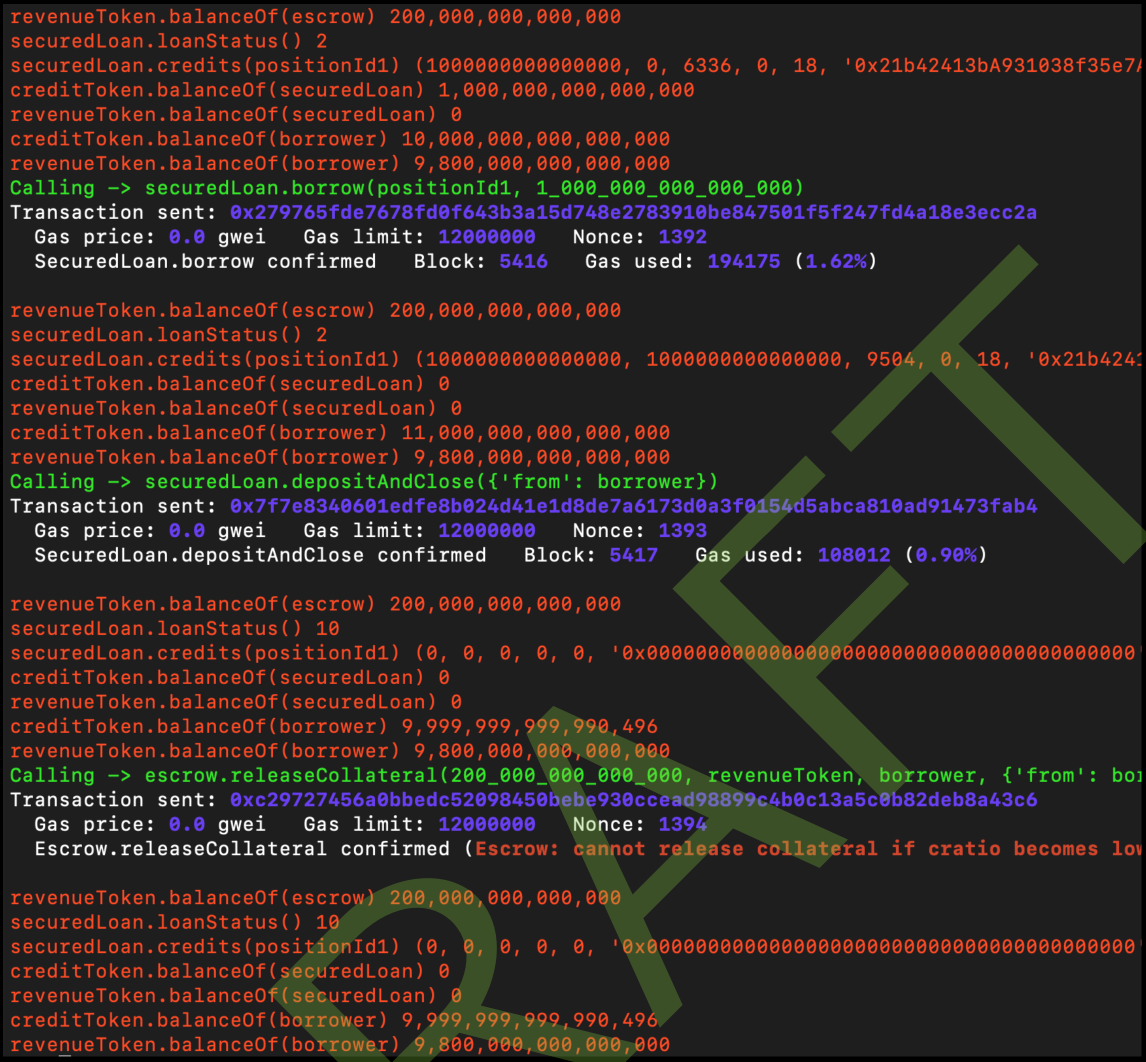
() function updates the deposited[token].amount with 0 value. Then the _getLatestCollateralRatio() returns 0 as collateralValue is also 0. Therefore, it is not possible to pass the assertion cratio >= minimumCollateralRatio as it is always false.

Listing 15: Escrow.sol (Line 52)

```
43     /**
44      * @notice updates the cratio according to the collateral value vs loan value
45      * @dev calls accrue interest on the loan contract to update the latest interest payable
46      * @return the updated collateral ratio in 18 decimals
47      */
48     function _getLatestCollateralRatio() internal returns (uint256) {
49         ILoan(loan).accrueInterest();
50         uint256 debtValue = ILoan(loan).getOutstandingDebt();
51         uint256 collateralValue = _getCollateralValue();
52         if (collateralValue == 0) return 0;
53         if (debtValue == 0) return MAX_INT;
54 
55         return _percent(collateralValue, debtValue, 18);
56     }
```

Proof of Concept:

1. All necessary contracts are deployed and initialized: CreditToken, RevenueToken, SimpleOracle, LoanLib, EscrowedLoan.
2. As arbiter, enable the RevenueToken token as collateral.
3. As the borrower, add 200_000_000_000_000 of RevenueToken tokens as collateral.
4. As borrower and lender, add debt position.
5. As the borrower, borrow all deposits.
6. As the borrower, attempt to call releaseCollateral() to withdraw all remaining collateral. Observe that the transaction reverts with Escrow: cannot release collateral if cratio becomes lower than the minimum error. Note that 200_000_000_000_000 of RevenueToken tokens are locked in the Escrow contract.



```
revenueToken.balanceOf(escrow) 200,000,000,000,000
securedLoan.loanStatus() 2
securedLoan.credits(positionId1) (1000000000000000, 0, 6336, 0, 18, '0x21b42413bA931038f35e7A
creditToken.balanceOf(securedLoan) 1,000,000,000,000,000
revenueToken.balanceOf(securedLoan) 0
creditToken.balanceOf(borrower) 10,000,000,000,000,000
revenueToken.balanceOf(borrower) 9,800,000,000,000,000
Calling -> securedLoan.borrow(positionId1, 1_000_000_000_000_000)
Transaction sent: 0x279765fde7678fd0f643b3a15d748e2783910be847501f5f247fd4a18e3ecc2a
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1392
  SecuredLoan.borrow confirmed   Block: 5416   Gas used: 194175 (1.62%)

revenueToken.balanceOf(escrow) 200,000,000,000,000
securedLoan.loanStatus() 2
securedLoan.credits(positionId1) (1000000000000000, 1000000000000000, 9504, 0, 18, '0x21b4241
creditToken.balanceOf(securedLoan) 0
revenueToken.balanceOf(securedLoan) 0
creditToken.balanceOf(borrower) 11,000,000,000,000,000
revenueToken.balanceOf(borrower) 9,800,000,000,000,000
Calling -> securedLoan.depositAndClose({'from': borrower})
Transaction sent: 0x7f7e8340601edfe8b024d41e1d8de7a6173d0a3f0154d5abca810ad91473fab4
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1393
  SecuredLoan.depositAndClose confirmed   Block: 5417   Gas used: 108012 (0.90%)

revenueToken.balanceOf(escrow) 200,000,000,000,000
securedLoan.loanStatus() 10
securedLoan.credits(positionId1) (0, 0, 0, 0, 0, '0x0000000000000000000000000000000000000000'
creditToken.balanceOf(securedLoan) 0
revenueToken.balanceOf(securedLoan) 0
creditToken.balanceOf(borrower) 9,999,999,999,990,496
revenueToken.balanceOf(borrower) 9,800,000,000,000,000
Calling -> escrow.releaseCollateral(200_000_000_000_000, revenueToken, borrower, {'from': bor
Transaction sent: 0xc29727456a0bbedc52098450bebe930ccead98899c4b0c13a5c0b82deb8a43c6
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1394
  Escrow.releaseCollateral confirmed (Escrow: cannot release collateral if cratio becomes low

revenueToken.balanceOf(escrow) 200,000,000,000,000
securedLoan.loanStatus() 10
securedLoan.credits(positionId1) (0, 0, 0, 0, 0, '0x0000000000000000000000000000000000000000'
creditToken.balanceOf(securedLoan) 0
revenueToken.balanceOf(securedLoan) 0
creditToken.balanceOf(borrower) 9,999,999,999,990,496
revenueToken.balanceOf(borrower) 9,800,000,000,000,000
```

Risk Level:

Likelihood - 5

Impact - 5

Recommendation:

It is recommended to fix the implementation of releaseCollateral() and _getLatestCollateralRatio() functions to allow the borrower to withdraw of the remaining collateral.

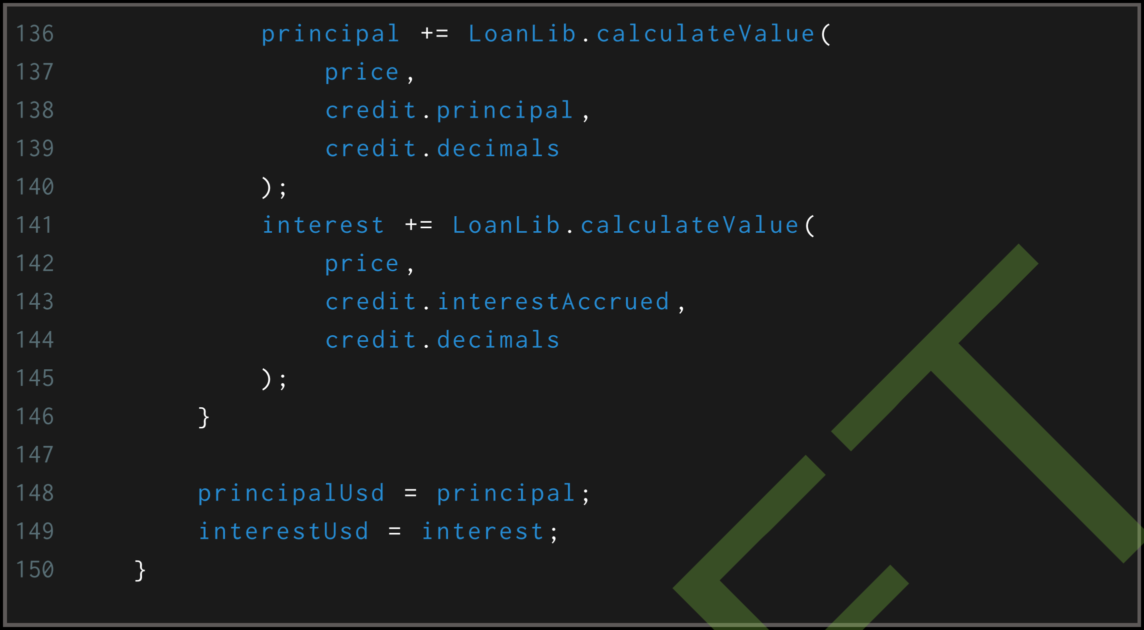
3.8 (HAL-08) GETOUTSTANDINGDEBT FUNCTION RETURNS UNDERSTATED VALUE - HIGH

Description:

In the LineOfCredit contract, the getOutstandingDebt() function allows users to get information about total outstanding debt valued in USD. The presented amount is a sum of principal and interest. The assessment revealed that the getOutstandingDebt() function does not consider the accrued interest from the last evaluation period. Thus, it presents misleading, understated value to the users as actual debt is higher.

Listing 16: LineOfCredit.sol

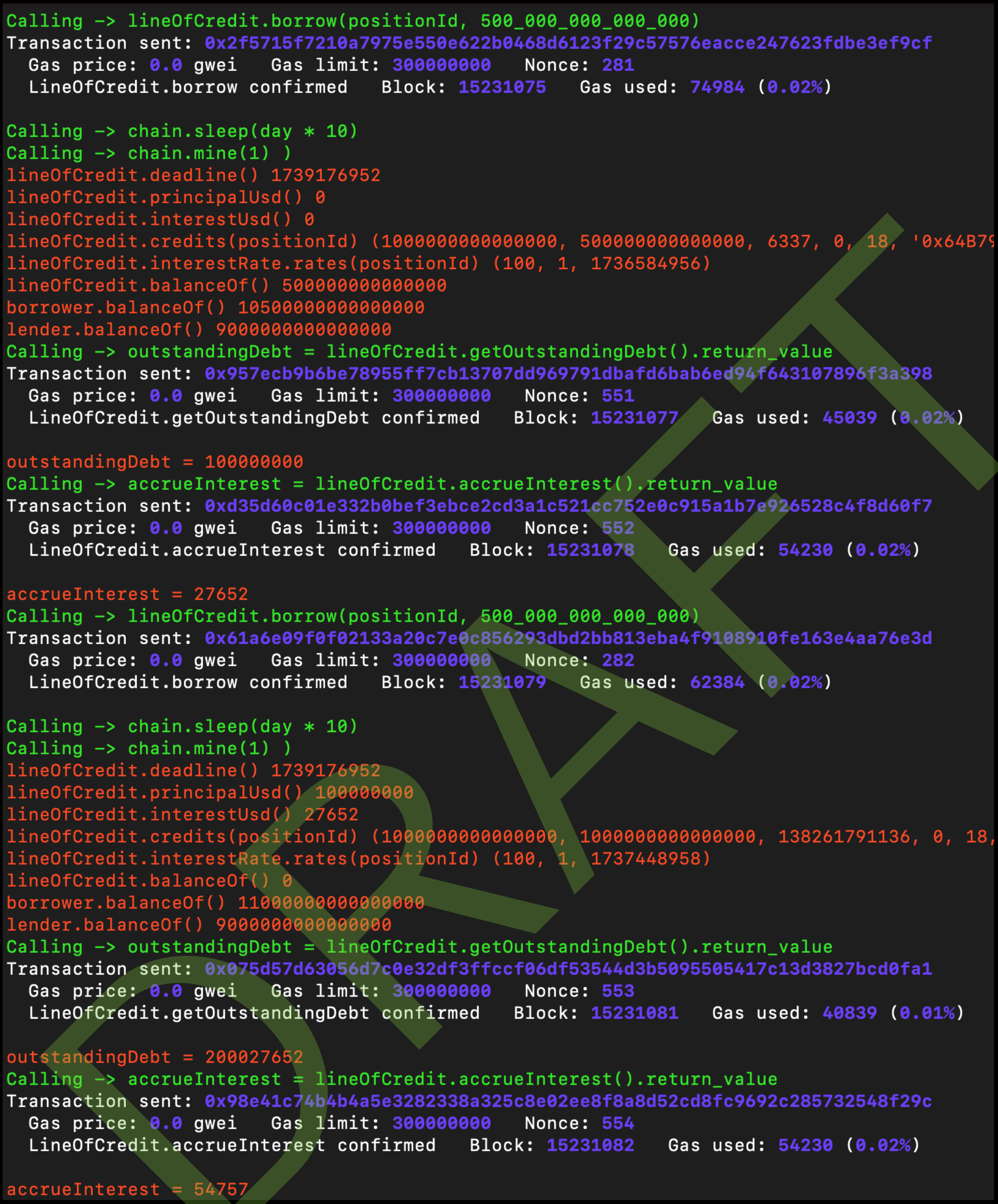
```
/**
  * @notice - Returns total credit obligation of borrower.
               Aggregated across all lenders.
               Denominated in USD 1e8.
  * @dev     - callable by anyone
  */
    function getOutstandingDebt() external override returns (
 ↳ uint256) {
        (uint256 p, uint256 i) = _updateOutstandingCredit();
        return p + i;
    }

    function _updateOutstandingCredit()
        internal
        returns (uint256 principal, uint256 interest)
    {
        uint256 len = ids.length;
        if (len == 0) return (0, 0);

        Credit memory credit;
        for (uint256 i = 0; i < len; i++) {
            credit = credits[ids[i]];

            int256 price = oracle.getLatestAnswer(credit.token);

```



```
            principal += LoanLib.calculateValue(
                price,
                credit.principal,
                credit.decimals
            );
            interest += LoanLib.calculateValue(
                price,
                credit.interestAccrued,
                credit.decimals
            );
        }

        principalUsd = principal;
        interestUsd = interest;
    }
```

Proof of Concept:

1. All necessary contracts are deployed and initialized: RevenueToken, SimpleOracle, LoanLib, LineOfCredit.
2. As borrower and lender1, add credit position for 10_000_000_000_-000_000 tokens.
3. As the borrower, borrow 500_000_000_000_000 tokens.
4. Forward blockchain time for 10 days.
5. Call getOutstandingDebt() function. Observe that returned value of 100000000 does not include accrued interest.
6. Call accrueInterest() function. Observe that returned value of 27652.
7. As borrower, borrow 500_000_000_000_000 tokens.
8. Forward blockchain time for 10 days.
9. Call the getOutstandingDebt() function. Observe that returned value of 200027652. Note that this value includes the accrued interest from step 6.
10. Call the accrueInterest() function. Observe that returned value of 54757. Note that this value includes the accrued interest only from the last 10 days and was not included in step 9.



```
Calling -> lineOfCredit.borrow(positionId, 500_000_000_000_000)
Transaction sent: 0x2f5715f7210a7975e550e622b0468d6123f29c57576eacce247623fdbe3ef9cf
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 281
  LineOfCredit.borrow confirmed   Block: 15231075   Gas used: 74984 (0.02%)

Calling -> chain.sleep(day * 10)
Calling -> chain.mine(1) )
lineOfCredit.deadline() 1739176952
lineOfCredit.principalUsd() 0
lineOfCredit.interestUsd() 0
lineOfCredit.credits(positionId) (1000000000000000, 500000000000000, 6337, 0, 18, '0x64B79
lineOfCredit.interestRate.rates(positionId) (100, 1, 1736584956)
lineOfCredit.balanceOf() 500000000000000
borrower.balanceOf() 10500000000000000
lender.balanceOf() 9000000000000000
Calling -> outstandingDebt = lineOfCredit.getOutstandingDebt().return_value
Transaction sent: 0x957ecb9b6be78955ff7cb13707dd969791dbafd6bab6ed94f643107896f3a398
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 551
  LineOfCredit.getOutstandingDebt confirmed   Block: 15231077   Gas used: 45039 (0.02%)

outstandingDebt = 100000000
Calling -> accrueInterest = lineOfCredit.accrueInterest().return_value
Transaction sent: 0xd35d60c01e332b0bef3ebce2cd3a1c521cc752e0c915a1b7e926528c4f8d60f7
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 552
  LineOfCredit.accrueInterest confirmed   Block: 15231078   Gas used: 54230 (0.02%)

accrueInterest = 27652
Calling -> lineOfCredit.borrow(positionId, 500_000_000_000_000)
Transaction sent: 0x61a6e09f0f02133a20c7e0c856293dbd2bb813eba4f9108910fe163e4aa76e3d
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 282
  LineOfCredit.borrow confirmed   Block: 15231079   Gas used: 62384 (0.02%)

Calling -> chain.sleep(day * 10)
Calling -> chain.mine(1) )
lineOfCredit.deadline() 1739176952
lineOfCredit.principalUsd() 100000000
lineOfCredit.interestUsd() 27652
lineOfCredit.credits(positionId) (1000000000000000, 1000000000000000, 138261791136, 0, 18,
lineOfCredit.interestRate.rates(positionId) (100, 1, 1737448958)
lineOfCredit.balanceOf() 0
borrower.balanceOf() 11000000000000000
lender.balanceOf() 9000000000000000
Calling -> outstandingDebt = lineOfCredit.getOutstandingDebt().return_value
Transaction sent: 0x075d57d63056d7c0e32df3ffccf06df53544d3b5095505417c13d3827bcd0fa1
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 553
  LineOfCredit.getOutstandingDebt confirmed   Block: 15231081   Gas used: 40839 (0.01%)

outstandingDebt = 200027652
Calling -> accrueInterest = lineOfCredit.accrueInterest().return_value
Transaction sent: 0x98e41c74b4b4a5e3282338a325c8e02ee8f8a8d52cd8fc9692c285732548f29c
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 554
  LineOfCredit.accrueInterest confirmed   Block: 15231082   Gas used: 54230 (0.02%)

accrueInterest = 54757
```

Risk Level:

Likelihood - 5

Impact - 3

Recommendation:

It is recommended that the getOutstandingDebt() function returns a value that includes the total accrued interest.

3.9 (HAL-09) BORROWING FROM NON-FIRST POSITION CAN DEADLOCK CONTRACT - HIGH

Description:

In the LineOfCredit contract, a borrower can add multiple credits with various lenders. The borrower can borrow any amount from any credit position using the borrow(bytes32 id, uint256 amount) function. However, the borrower can only repay first credit position using depositAndRepay() or depositAndClose() functions.

Listing 17: LineOfCredit.sol (Lines 309,342)

```
302 /**
303     * @notice - Transfers enough tokens to repay entire credit position from `borrower` to Loan contract.
304     * @dev - callable by borrower
305     */
306     function depositAndClose()
307         external
308         override
309         whileBorrowing
310         onlyBorrower
311         returns (bool)
312     {
313         bytes32 id = ids[0];
314         _accrueInterest(id);
315 
316         uint256 totalOwed = credits[id].principal + credits[id].interestAccrued;
317 
318         // borrower deposits remaining balance not already repaid and held in contract
319         bool success = IERC20(credits[id].token).transferFrom(
320             msg.sender,
321             address(this),
322             totalOwed
323         );
324         require(success, "Loan: deposit failed");
```

```
        // clear the credit
        _repay(id, totalOwed);

        require(_close(id));
        return true;
    }

    /**
     * @dev - Transfers token used in credit position from msg.sender to Loan contract.
     * @dev - callable by anyone
     * @notice - see _repay() for more details
     * @param amount - amount of `token` in `id` to pay back
     */

    function depositAndRepay(uint256 amount)
        external
        override
        whileBorrowing
        returns (bool)
    {
        bytes32 id = ids[0];
        _accrueInterest(id);

        require(amount <= credits[id].principal + credits[id].interestAccrued);

        bool success = IERC20(credits[id].token).transferFrom(
            msg.sender,
            address(this),
            amount
        );
        require(success, "Loan: failed repayment");

        _repay(id, amount);
        return true;
    }
```

When the borrower borrow() deposit from the second position, repaying the debt will not be possible, as the whileBorrowing() modifier would block that operation. At this point, the close(bytes32 id) function can be called to close the first credit position unless the _accrueInterest() internal function is called and interest is accrued, preventing the closing of the unpaid debt.

Listing 18: LineOfCredit.sol

```
    modifier whileBorrowing() {
        require(ids.length > 0 && credits[ids[0]].principal > 0);
        _;
    }
```

To escape from the situation, the borrower can still borrow() any amount from the first position unless the lender does not withdraw all liquidity. The withdrawal operation accrues the interest, which cannot be paid, as the whileBorrowing() modifier only considers the principal. Also, the borrower can't borrow() anymore from the empty deposit. As a result, a deadlock occurs in the contract, and the borrower can't pay off the debt.

The root cause of this issue is the _sortIntoQ(bytes32 p) function, which supposes to shift the credit position with the borrowed deposit to the beginning of the ids collection, but it does not work as expected. Sample invalid flow:

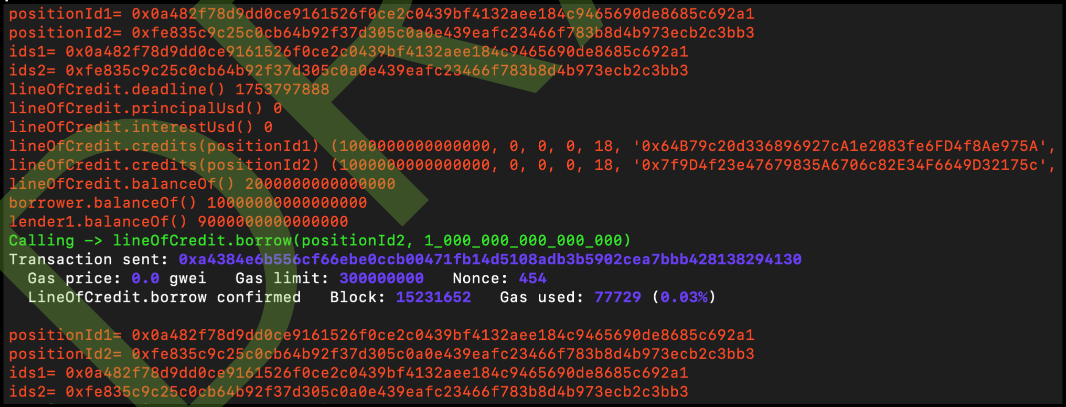
1. For i = 0; _i = i = 0 as first credit's principal is 0.
2. For i = 1, function returns true, as _i = 0.
3. No position swap occurs.

Listing 19: LineOfCredit.sol

```
function _sortIntoQ(bytes32 p) internal returns (bool) {
        uint256 len = ids.length;
        uint256 _i = 0; // index that p should be moved to

        for (uint256 i = 0; i < len; i++) {
            bytes32 id = ids[i];
            if (p != id) {
                if (credits[id].principal > 0) continue; // `id` should be placed before `p`
                _i = i; // index of first undrawn LoC found
            } else {
                if (_i == 0) return true; // `p` in earliest possible index
                // swap positions
                ids[i] = ids[_i];
                ids[_i] = p;
            }
        }

        return true;
    }
```

Proof of Concept:

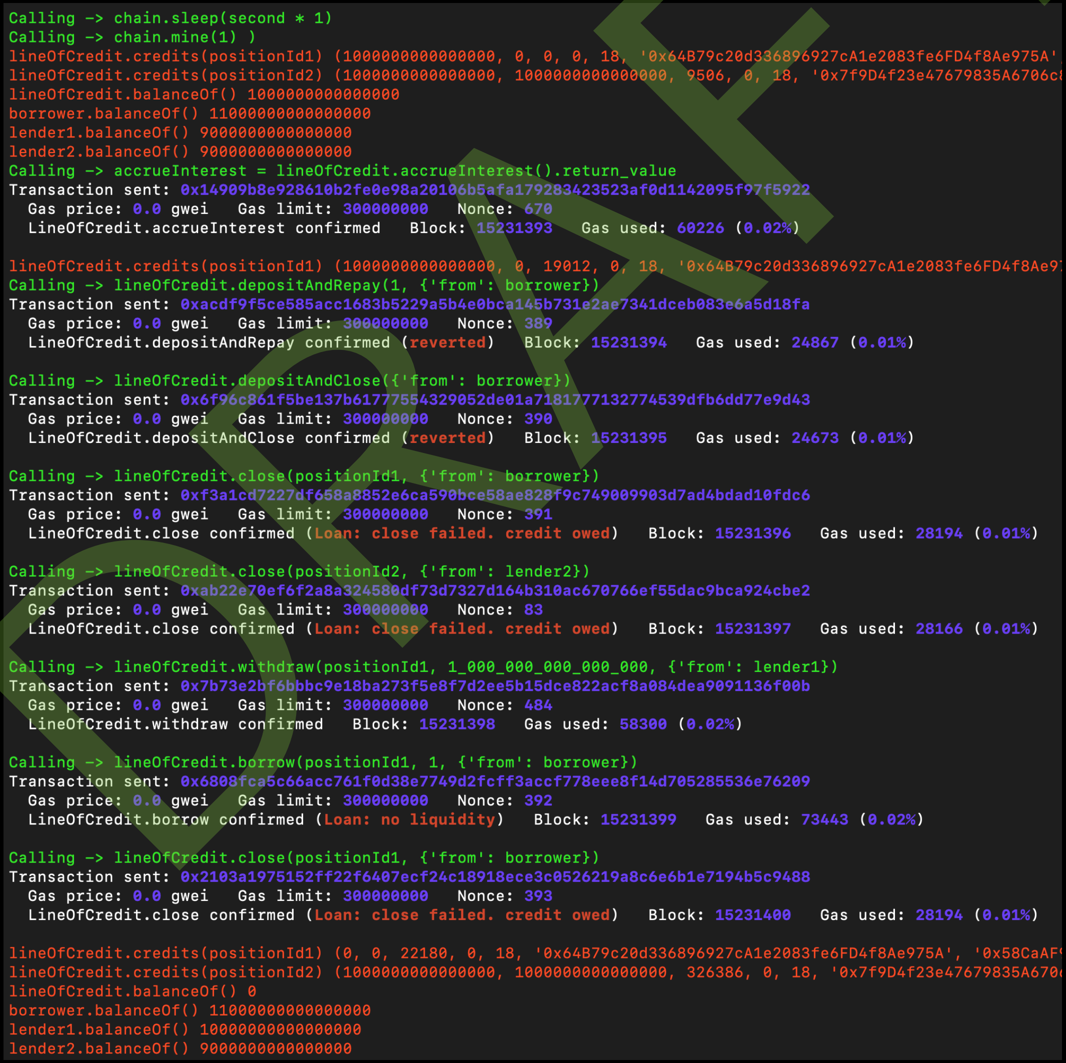
1. All necessary contracts are deployed and initialized: RevenueToken, SimpleOracle, LoanLib, LineOfCredit.
2. As borrower and lender1, add credit position for 10_000_000_000_000_000 tokens. This action registers the first credit position.
3. As a borrower and lender2, add credit position for 10_000_000_000_000_000 tokens. This action registers the second credit position.
4. As the borrower, borrow 10_000_000_000_000_000 tokens from the second position.
5. Note that the ids collection was not updated; the second credit position is not set to the first.



```
positionId1= 0x0a482f78d9dd0ce9161526f0ce2c0439bf4132aee184c9465690de8685c692a1
positionId2= 0xfe835c9c25c0cb64b92f37d305c0a0e439eafc23466f783b8d4b973ecb2c3bb3
ids1= 0x0a482f78d9dd0ce9161526f0ce2c0439bf4132aee184c9465690de8685c692a1
ids2= 0xfe835c9c25c0cb64b92f37d305c0a0e439eafc23466f783b8d4b973ecb2c3bb3
lineOfCredit.deadline() 1753797888
lineOfCredit.principalUsd() 0
lineOfCredit.interestUsd() 0
lineOfCredit.credits(positionId1) (1000000000000000, 0, 0, 0, 18, '0x64B79c20d336896927cA1e2083fe6FD4f8Ae975A',
lineOfCredit.credits(positionId2) (1000000000000000, 0, 0, 0, 18, '0x7f9D4f23e47679835A6706c82E34F6649D32175c',
lineOfCredit.balanceOf() 2000000000000000
borrower.balanceOf() 10000000000000000
lender1.balanceOf() 9000000000000000
Calling -> lineOfCredit.borrow(positionId2, 1_000_000_000_000_000)
Transaction sent: 0xa4384e6b556cf66ebe0ccb00471fb14d5108adb3b5902cea7bbb428138294130
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 454
  LineOfCredit.borrow confirmed   Block: 15231652   Gas used: 77729 (0.03%)

positionId1= 0x0a482f78d9dd0ce9161526f0ce2c0439bf4132aee184c9465690de8685c692a1
positionId2= 0xfe835c9c25c0cb64b92f37d305c0a0e439eafc23466f783b8d4b973ecb2c3bb3
ids1= 0x0a482f78d9dd0ce9161526f0ce2c0439bf4132aee184c9465690de8685c692a1
ids2= 0xfe835c9c25c0cb64b92f37d305c0a0e439eafc23466f783b8d4b973ecb2c3bb3
```

6. Forward blockchain time for 1 second.
7. Call the accrueInterest function. Note that the interestAccrued parameter value in the first credit record is updated.
8. As the borrower, attempt to depositAndRepay with any value. Observe that the transaction reverted.
9. As the borrower, attempt to depositAndClose. Observe that the transaction reverted.

10. As the borrower, attempt to close the first credit position. Observe that the transaction reverted with Loan: close failed. credit owed error.
11. As the lender2, attempt to close the second credit position. Observe that the transaction reverted with Loan: close failed. credit owed error.
12. As the lender1, withdraw all liquidity.
13. As the borrower, attempt to borrow. Observe that the transaction reverted with a Loan: no liquidity error.
14. As the borrower, attempt to close the first credit position again. Observe that the transaction reverted with Loan: close failed. credit owed error.



```
Calling -> chain.sleep(second * 1)
Calling -> chain.mine(1) )
lineOfCredit.credits(positionId1) (1000000000000000, 0, 0, 0, 18, '0x64B79c20d336896927cA1e2083fe6FD4f8Ae975A',
lineOfCredit.credits(positionId2) (1000000000000000, 1000000000000000, 9506, 0, 18, '0x7f9D4f23e47679835A6706c
lineOfCredit.balanceOf() 1000000000000000
borrower.balanceOf() 11000000000000000
lender1.balanceOf() 9000000000000000
lender2.balanceOf() 9000000000000000
Calling -> accrueInterest = lineOfCredit.accrueInterest().return_value
Transaction sent: 0x14909b8e928610b2fe0e98a20106b5afa179283423523af0d1142095f97f5922
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 670
  LineOfCredit.accrueInterest confirmed   Block: 15231393   Gas used: 60226 (0.02%)

lineOfCredit.credits(positionId1) (1000000000000000, 0, 19012, 0, 18, '0x64B79c20d336896927cA1e2083fe6FD4f8Ae97
Calling -> lineOfCredit.depositAndRepay(1, {'from': borrower})
Transaction sent: 0xacdf9f5ce585acc1683b5229a5b4e0bca145b731e2ae7341dceb083e6a5d18fa
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 389
  LineOfCredit.depositAndRepay confirmed (reverted)   Block: 15231394   Gas used: 24867 (0.01%)

Calling -> lineOfCredit.depositAndClose({'from': borrower})
Transaction sent: 0x6f96c861f5be137b61777554329052de01a7181777132774539dfb6dd77e9d43
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 390
  LineOfCredit.depositAndClose confirmed (reverted)   Block: 15231395   Gas used: 24673 (0.01%)

Calling -> lineOfCredit.close(positionId1, {'from': borrower})
Transaction sent: 0xf3a1cd7227df658a8852e6ca590bce58ae828f9c749009903d7ad4bdad10fdc6
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 391
  LineOfCredit.close confirmed (Loan: close failed. credit owed)   Block: 15231396   Gas used: 28194 (0.01%)

Calling -> lineOfCredit.close(positionId2, {'from': lender2})
Transaction sent: 0xab22e70ef6f2a8a324580df73d7327d164b310ac670766ef55dac9bca924cbe2
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 83
  LineOfCredit.close confirmed (Loan: close failed. credit owed)   Block: 15231397   Gas used: 28166 (0.01%)

Calling -> lineOfCredit.withdraw(positionId1, 1_000_000_000_000_000, {'from': lender1})
Transaction sent: 0x7b73e2bf6bbbc9e18ba273f5e8f7d2ee5b15dce822acf8a084dea9091136f00b
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 484
  LineOfCredit.withdraw confirmed   Block: 15231398   Gas used: 58300 (0.02%)

Calling -> lineOfCredit.borrow(positionId1, 1, {'from': borrower})
Transaction sent: 0x6808fca5c66acc761f0d38e7749d2fcff3accf778eee8f14d705285536e76209
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 392
  LineOfCredit.borrow confirmed (Loan: no liquidity)   Block: 15231399   Gas used: 73443 (0.02%)

Calling -> lineOfCredit.close(positionId1, {'from': borrower})
Transaction sent: 0x2103a1975152ff22f6407ecf24c18918ece3c0526219a8c6e6b1e7194b5c9488
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 393
  LineOfCredit.close confirmed (Loan: close failed. credit owed)   Block: 15231400   Gas used: 28194 (0.01%)

lineOfCredit.credits(positionId1) (0, 0, 22180, 0, 18, '0x64B79c20d336896927cA1e2083fe6FD4f8Ae975A', '0x58CaAF
lineOfCredit.credits(positionId2) (1000000000000000, 1000000000000000, 326386, 0, 18, '0x7f9D4f23e47679835A670
lineOfCredit.balanceOf() 0
borrower.balanceOf() 11000000000000000
lender1.balanceOf() 10000000000000000
lender2.balanceOf() 9000000000000000
```

Risk Level:

Likelihood - 3
Impact - 5

Recommendation:

It is recommended to allow the borrower to pay off any credit position without extra steps (such as calling borrow() function).

3.10 (HAL-10) UPDATEOWNERSPLIT FUNCTION CAN BE ABUSED BY LENDER OR BORROWER - HIGH

Description:

In the SpigotedLoan contract, a borrower and arbiter can add a spigot. Both of them must agree on a percentage value in ownerSplit parameter from SpigotSettings structure. The ownerSplit parameter determines the amount of revenue tokens that stays in SpigotController contract and are used to pay off the debt.

Listing 20: SpigotedLoan.sol

```
    function addSpigot(
        address revenueContract,
        SpigotController.SpigotSettings calldata setting
    ) external mutualConsent(arbiter, borrower) returns (bool) {
        return spigot.addSpigot(revenueContract, setting);
    }
```

Listing 21: Spigot.sol

```
    struct SpigotSettings {
        address token;                // token to claim as revenue
 ↳ from contract
        uint8 ownerSplit;             // x/100 % to Owner, rest to
 ↳ Treasury
        bytes4 claimFunction;         // function signature on
 ↳ contract to call and claim revenue
        bytes4 transferOwnerFunction; // function signature on
 ↳ conract to call and transfer ownership
    }
```

The updateOwnerSplit() function can be used to update the ownerSplit parameter in particular spigot basing on loan health. If the loan status is active, then defaultRevenueSplit is applied. The defaultRevenueSplit parameter is set in constructor. Based on unit tests, this parameter should have 10 as a value.

Listing 22: Spigot.sol (Lines 55,67,70)

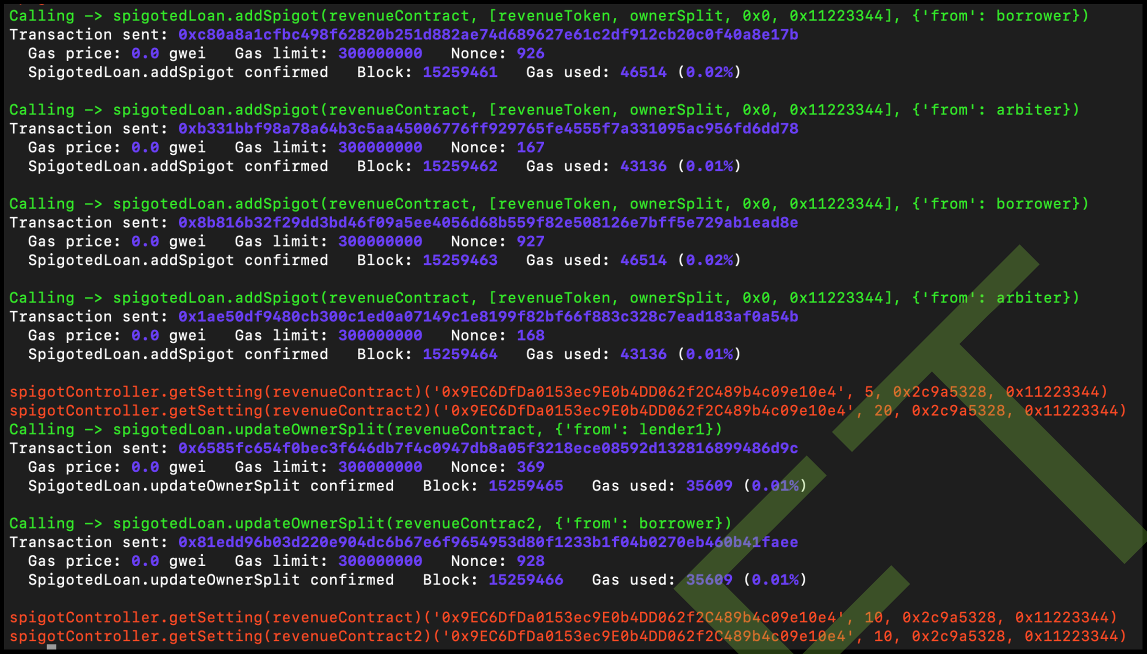
```
54      /**
55       * @notice changes the revenue split between borrower treasury
↳ and lan repayment based on loan health
56       * @dev     - callable `arbiter` + `borrower`
57       * @param revenueContract - spigot to update
58       */
59      function updateOwnerSplit(address revenueContract) external
↳ returns (bool) {
60          (, uint8 split, , bytes4 transferFunc) = spigot.getSetting
↳ (
61              revenueContract
62          );
63
64          require(transferFunc != bytes4(0), "SpgtLoan: no spigot");
65
66          if (
67              loanStatus == LoanLib.STATUS.ACTIVE && split !=
↳ defaultRevenueSplit
68          ) {
69              // if loan is healthy set split to default take rate
70              spigot.updateOwnerSplit(revenueContract,
↳ defaultRevenueSplit);
71          } else if (
72              loanStatus == LoanLib.STATUS.LIQUIDATABLE && split !=
↳ MAX_SPLIT
73          ) {
74              // if loan is in distress take all revenue to repay
↳ loan
75              spigot.updateOwnerSplit(revenueContract, MAX_SPLIT);
76          }
77
78          return true;
79      }
```

The assessment revealed that the updateOwnerSplit() function could be abused by the borrower or the lender, depending on the situation. If the ownerSplit parameter is set below the defaultRevenueSplit parameter, the lender can call it to increase the split percentage. Then more revenue would be used to pay off the debt. If the ownerSplit parameter is set above the defaultRevenueSplit parameter, the borrower can call it to decrease the split percentage. Then less revenue would be used to pay off the debt, and more revenue would return to the treasury (which is by default set to the borrower). Also, the healthy loan is always set to ACTIVE.

Additionally, based on the comment, the function is meant to be called by the arbiter or the borrower; however, no such authorization check is implemented.

Proof of Concept:

1. All necessary contracts are deployed and initialized: CreditToken, RevenueToken, SimpleOracle, LoanLib, SpigotedLoan, SimpleRevenueContract. In the SpigotedLoan set the defaultRevenueSplit parameter to 10.
2. As borrower and arbiter, add a new spigot with the addSpigot() function. Set the ownerSplit parameter to 5.
3. As the lender, call the updateOwnerSplit() function for the spigot added in the previous step. Note that the ownerSplit parameter is now set to 10.
4. As borrower and arbiter, add a new spigot with the addSpigot() function. Set the ownerSplit parameter to 20.
5. As the borrower, call the updateOwnerSplit() function for the spigot added in the previous step. Note that the ownerSplit parameter is now set to 10.



```
Calling -> spigotedLoan.addSpigot(revenueContract, [revenueToken, ownerSplit, 0x0, 0x11223344], {'from': borrower})
Transaction sent: 0xc80a8a1cfbc498f62820b251d882ae74d689627e61c2df912cb20c0f40a8e17b
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 926
  SpigotedLoan.addSpigot confirmed   Block: 15259461   Gas used: 46514 (0.02%)

Calling -> spigotedLoan.addSpigot(revenueContract, [revenueToken, ownerSplit, 0x0, 0x11223344], {'from': arbiter})
Transaction sent: 0xb331bbf98a78a64b3c5aa45006776ff929765fe4555f7a331095ac956fd6dd78
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 167
  SpigotedLoan.addSpigot confirmed   Block: 15259462   Gas used: 43136 (0.01%)

Calling -> spigotedLoan.addSpigot(revenueContract, [revenueToken, ownerSplit, 0x0, 0x11223344], {'from': borrower})
Transaction sent: 0x8b816b32f29dd3bd46f09a5ee4056d68b559f82e508126e7bff5e729ab1ead8e
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 927
  SpigotedLoan.addSpigot confirmed   Block: 15259463   Gas used: 46514 (0.02%)

Calling -> spigotedLoan.addSpigot(revenueContract, [revenueToken, ownerSplit, 0x0, 0x11223344], {'from': arbiter})
Transaction sent: 0x1ae50df9480cb300c1ed0a07149c1e8199f82bf66f883c328c7ead183af0a54b
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 168
  SpigotedLoan.addSpigot confirmed   Block: 15259464   Gas used: 43136 (0.01%)

spigotController.getSetting(revenueContract)('0x9EC6DfDa0153ec9E0b4DD062f2C489b4c09e10e4', 5, 0x2c9a5328, 0x11223344)
spigotController.getSetting(revenueContract2)('0x9EC6DfDa0153ec9E0b4DD062f2C489b4c09e10e4', 20, 0x2c9a5328, 0x11223344)
Calling -> spigotedLoan.updateOwnerSplit(revenueContract, {'from': lender1})
Transaction sent: 0x6585fc654f0bec3f646db7f4c0947db8a05f3218ece08592d132816899486d9c
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 369
  SpigotedLoan.updateOwnerSplit confirmed   Block: 15259465   Gas used: 35609 (0.01%)

Calling -> spigotedLoan.updateOwnerSplit(revenueContrac2, {'from': borrower})
Transaction sent: 0x81edd96b03d220e904dc6b67e6f9654953d80f1233b1f04b0270eb460b41faee
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 928
  SpigotedLoan.updateOwnerSplit confirmed   Block: 15259466   Gas used: 35609 (0.01%)

spigotController.getSetting(revenueContract)('0x9EC6DfDa0153ec9E0b4DD062f2C489b4c09e10e4', 10, 0x2c9a5328, 0x11223344)
spigotController.getSetting(revenueContract2)('0x9EC6DfDa0153ec9E0b4DD062f2C489b4c09e10e4', 10, 0x2c9a5328, 0x11223344)
```

Risk Level:

Likelihood - 5

Impact - 3

Recommendation:

It is recommended to add an authorization check that only the arbiter can call the updateOwnerSplit() function.

3.11 (HAL-11) UNUSED REVENUE TOKENS LOCKOUT WHILE LOAN IS ACTIVE - HIGH

Description:

In the SpigotedLoan contract, the borrower can add spigots with revenue contracts that will support repayment of the debt. The revenue contract earns revenue tokens that later can be exchanged for the credit tokens using the claimAndRepay() and claimAndTrade() functions. These functions call the _claimAndTrade() internal function responsible for claiming escrowed tokens from the SpigotController contract and trading the claimed tokens. Also, the _claimAndTrade() function updates the unusedTokens collection with claimed tokens (revenue tokens) that were not traded.

The assessment revealed that it is possible to trade less revenue tokens than previously claimed. The rest of the claimed tokens are locked in the SpigotedLoan contract. They remain locked until the debt is paid off by other means, and the borrower can transfer all tokens from unusedTokens collection using the sweep() function. The SpigotedLoan contract has no other function that allows to trade and repay revenue tokens that were previously claimed.

This vulnerability remains the same for both cases when the revenue token is either ERC20 token or ether.

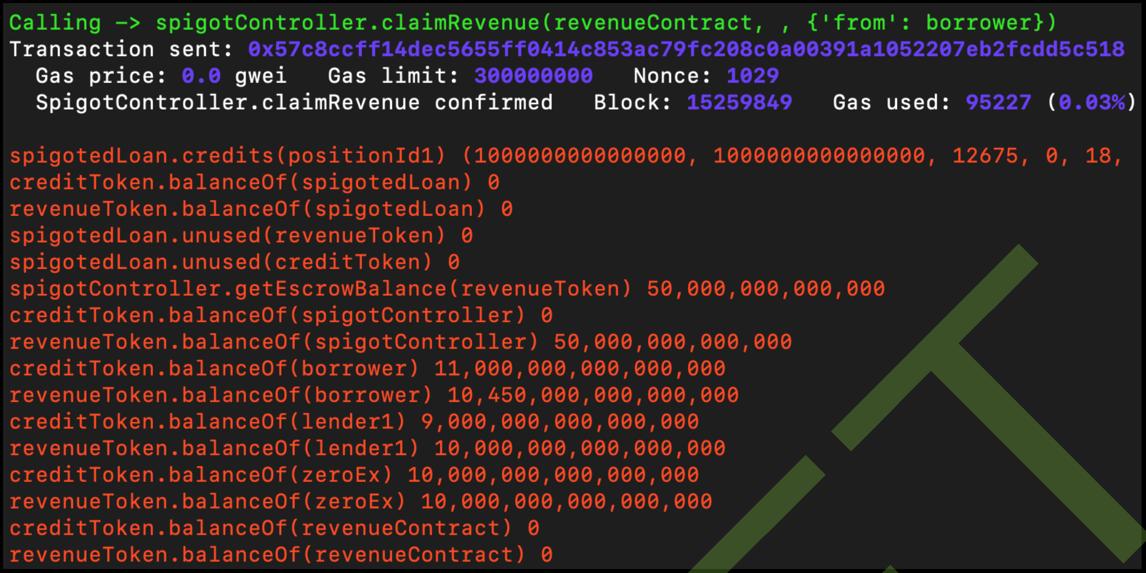
This vulnerability can also be abused by the borrower to use less revenue tokens to pay off the debt than was intended, making the revenue split mechanism ineffective.

Listing 23: SpigotedLoan.sol (Lines 199-203)

```
function _claimAndTrade(
        address claimToken,
        address targetToken,
        bytes calldata zeroExTradeData
    ) internal returns (uint256 tokensBought) {
        uint256 existingClaimTokens = IERC20(claimToken).balanceOf(
            address(this)
        );
        uint256 existingTargetTokens = IERC20(targetToken).balanceOf(
            address(this)
        );

        uint256 tokensClaimed = spigot.claimEscrow(claimToken);

        if (claimToken == address(0)) {
            // if claiming/trading eth send as msg.value to dex
            (bool success, ) = swapTarget.call{value: tokensClaimed}(
                zeroExTradeData
            );
            require(success, "SpigotCnsm: trade failed");
        } else {
            IERC20(claimToken).approve(
                swapTarget,
                existingClaimTokens + tokensClaimed
            );
            (bool success, ) = swapTarget.call(zeroExTradeData);
            require(success, "SpigotCnsm: trade failed");
        }

        uint256 targetTokens = IERC20(targetToken).balanceOf(address(this));

        // ideally we could use oracle to calculate # of tokens to receive
        // but claimToken might not have oracle. targetToken must have oracle

        // underflow revert ensures we have more tokens than we started with
        tokensBought = targetTokens - existingTargetTokens;

        emit TradeSpigotRevenue(
            claimToken,
            tokensClaimed,
            targetToken,
            tokensBought
        );

        // update unused if we didnt sell all claimed tokens in trade
        // also underflow revert protection here
        unusedTokens[claimToken] +=
            IERC20(claimToken).balanceOf(address(this)) -
            existingClaimTokens;
    }
```

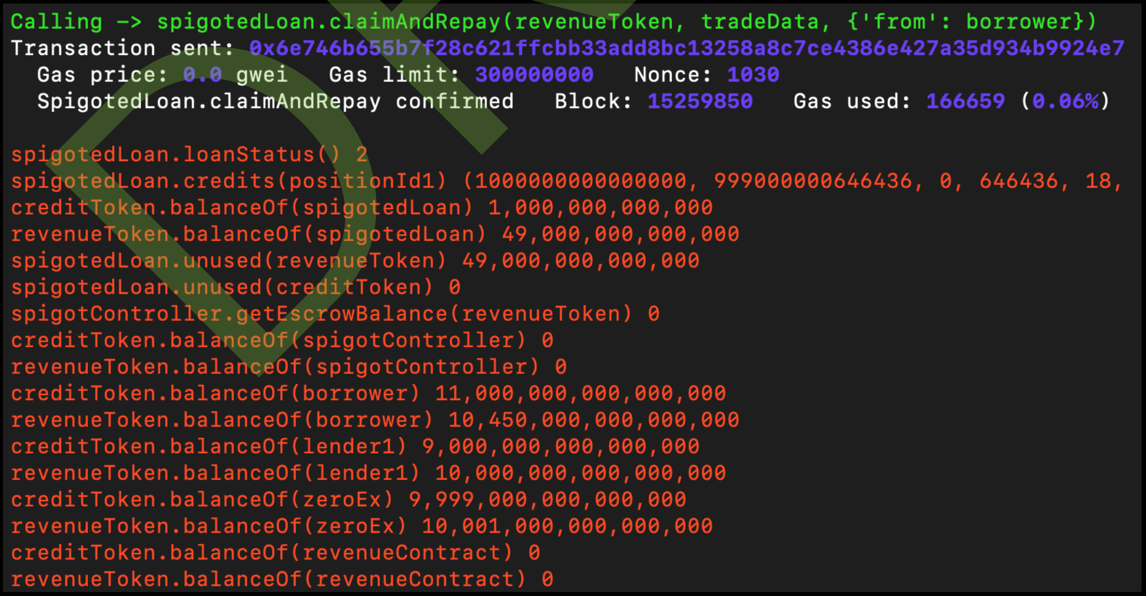
Proof of Concept:

1. All necessary contracts are deployed and initialized: CreditToken, RevenueToken, SimpleOracle, LoanLib, SpigotedLoan, SimpleRevenueContract.
2. As borrower and lender1, add credit position for 10_000_000_000_000_000 tokens.
3. As borrower, borrow() all deposits.
4. As borrower and arbiter, add new spigot with addSpigot() function and RevenueContract as input. Set the ownerSplit parameter to 10.
5. As the borrower, transfer the ownership of the RevenueContract contract to the SpigotController.
6. Mint 500_000_000_000_000 revenue tokens to the RevenueContract contract to simulate token gain.
7. As borrower, call claimRevenue() in SpigotController contract to claim revenue tokens. Note that 90% of tokens (450_000_000_000_000) are transferred to the treasury (which is the borrower), the 10% (50_000_000_000_000) of tokens are transferred to the SpigotController.



```
Calling -> spigotController.claimRevenue(revenueContract, , {'from': borrower})
Transaction sent: 0x57c8ccff14dec5655ff0414c853ac79fc208c0a00391a1052207eb2fcdd5c518
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 1029
  SpigotController.claimRevenue confirmed   Block: 15259849   Gas used: 95227 (0.03%)

spigotedLoan.credits(positionId1) (1000000000000000, 1000000000000000, 12675, 0, 18,
creditToken.balanceOf(spigotedLoan) 0
revenueToken.balanceOf(spigotedLoan) 0
spigotedLoan.unused(revenueToken) 0
spigotedLoan.unused(creditToken) 0
spigotController.getEscrowBalance(revenueToken) 50,000,000,000,000
creditToken.balanceOf(spigotController) 0
revenueToken.balanceOf(spigotController) 50,000,000,000,000
creditToken.balanceOf(borrower) 11,000,000,000,000,000
revenueToken.balanceOf(borrower) 10,450,000,000,000,000
creditToken.balanceOf(lender1) 9,000,000,000,000,000
revenueToken.balanceOf(lender1) 10,000,000,000,000,000
creditToken.balanceOf(zeroEx) 10,000,000,000,000,000
revenueToken.balanceOf(zeroEx) 10,000,000,000,000,000
creditToken.balanceOf(revenueContract) 0
revenueToken.balanceOf(revenueContract) 0
```

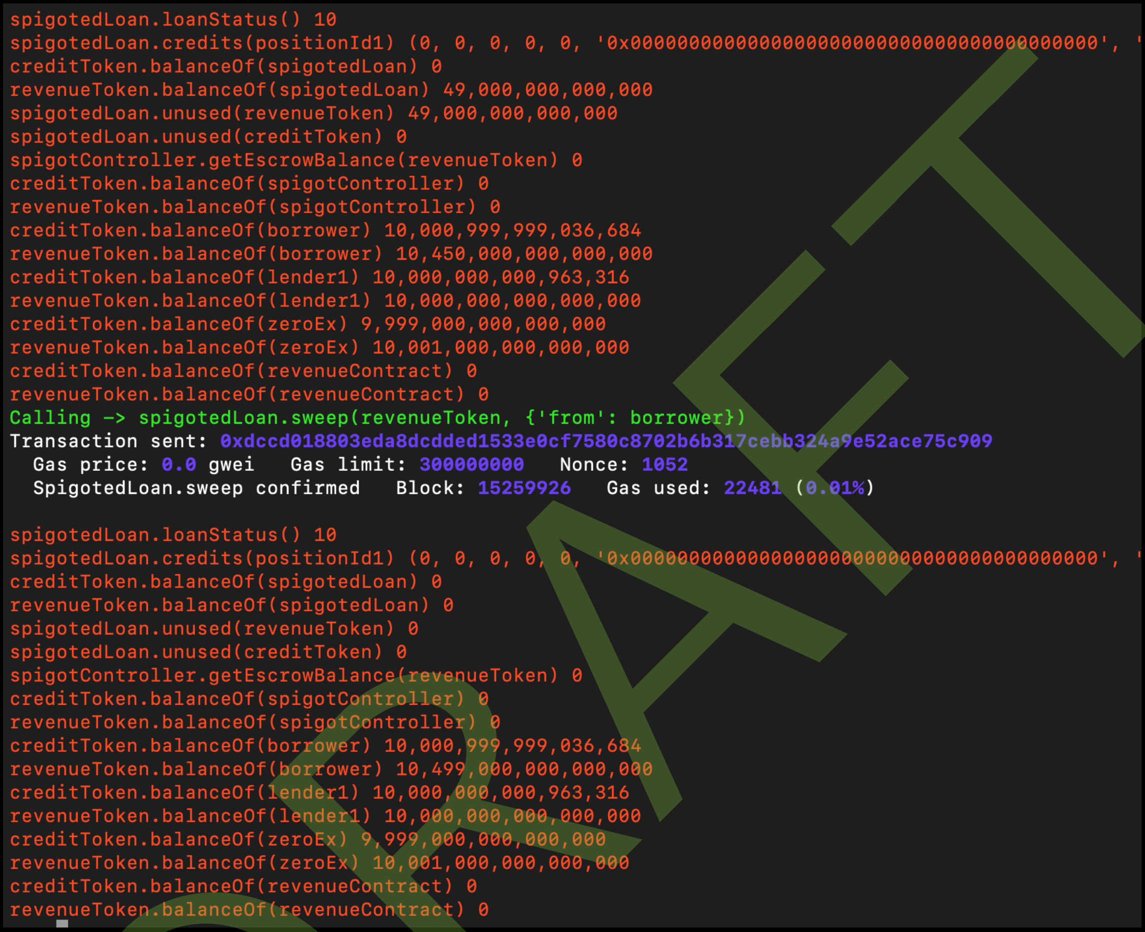
6. As the borrower, call claimAndRepay() in the SpigotedLoan contract to trade escrowed revenue tokens and pay off part of the debt. As an input, trade exchange data provide 1_000_000_000_000 of revenue tokens that should be exchanged for credit tokens.
7. Observe the SpigotedLoan balances. Note that the contract has 1,000,000,000,000 credit tokens and 49,000,000,000,000 revenue tokens. Also, 49,000,000,000,000 revenue tokens are stored in unusedTokens collection.



```
Calling -> spigotedLoan.claimAndRepay(revenueToken, tradeData, {'from': borrower})
Transaction sent: 0x6e746b655b7f28c621ffcbb33add8bc13258a8c7ce4386e427a35d934b9924e7
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 1030
  SpigotedLoan.claimAndRepay confirmed   Block: 15259850   Gas used: 166659 (0.06%)

spigotedLoan.loanStatus() 2
spigotedLoan.credits(positionId1) (1000000000000000, 999000000646436, 0, 646436, 18,
creditToken.balanceOf(spigotedLoan) 1,000,000,000,000
revenueToken.balanceOf(spigotedLoan) 49,000,000,000,000
spigotedLoan.unused(revenueToken) 49,000,000,000,000
spigotedLoan.unused(creditToken) 0
spigotController.getEscrowBalance(revenueToken) 0
creditToken.balanceOf(spigotController) 0
revenueToken.balanceOf(spigotController) 0
creditToken.balanceOf(borrower) 11,000,000,000,000,000
revenueToken.balanceOf(borrower) 10,450,000,000,000,000
creditToken.balanceOf(lender1) 9,000,000,000,000,000
revenueToken.balanceOf(lender1) 10,000,000,000,000,000
creditToken.balanceOf(zeroEx) 9,999,000,000,000,000
revenueToken.balanceOf(zeroEx) 10,001,000,000,000,000
creditToken.balanceOf(revenueContract) 0
revenueToken.balanceOf(revenueContract) 0
```

8. As the borrower, call depositAndClose() to pay off the debt.

9. As the borrower, call sweep() function with revenue token address as an input to receive 49,000,000,000,000 revenue tokens from SpigotedLoan contract.



```
spigotedLoan.loanStatus() 10
spigotedLoan.credits(positionId1) (0, 0, 0, 0, 0, '0x0000000000000000000000000000000000000000', '
creditToken.balanceOf(spigotedLoan) 0
revenueToken.balanceOf(spigotedLoan) 49,000,000,000,000
spigotedLoan.unused(revenueToken) 49,000,000,000,000
spigotedLoan.unused(creditToken) 0
spigotController.getEscrowBalance(revenueToken) 0
creditToken.balanceOf(spigotController) 0
revenueToken.balanceOf(spigotController) 0
creditToken.balanceOf(borrower) 10,000,999,999,036,684
revenueToken.balanceOf(borrower) 10,450,000,000,000,000
creditToken.balanceOf(lender1) 10,000,000,000,963,316
revenueToken.balanceOf(lender1) 10,000,000,000,000,000
creditToken.balanceOf(zeroEx) 9,999,000,000,000,000
revenueToken.balanceOf(zeroEx) 10,001,000,000,000,000
creditToken.balanceOf(revenueContract) 0
revenueToken.balanceOf(revenueContract) 0
Calling -> spigotedLoan.sweep(revenueToken, {'from': borrower})
Transaction sent: 0xdccd018803eda8dcdded1533e0cf7580c8702b6b317cebb324a9e52ace75c909
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 1052
  SpigotedLoan.sweep confirmed   Block: 15259926   Gas used: 22481 (0.01%)

spigotedLoan.loanStatus() 10
spigotedLoan.credits(positionId1) (0, 0, 0, 0, 0, '0x0000000000000000000000000000000000000000', '
creditToken.balanceOf(spigotedLoan) 0
revenueToken.balanceOf(spigotedLoan) 0
spigotedLoan.unused(revenueToken) 0
spigotedLoan.unused(creditToken) 0
spigotController.getEscrowBalance(revenueToken) 0
creditToken.balanceOf(spigotController) 0
revenueToken.balanceOf(spigotController) 0
creditToken.balanceOf(borrower) 10,000,999,999,036,684
revenueToken.balanceOf(borrower) 10,499,000,000,000,000
creditToken.balanceOf(lender1) 10,000,000,000,963,316
revenueToken.balanceOf(lender1) 10,000,000,000,000,000
creditToken.balanceOf(zeroEx) 9,999,000,000,000,000
revenueToken.balanceOf(zeroEx) 10,001,000,000,000,000
creditToken.balanceOf(revenueContract) 0
revenueToken.balanceOf(revenueContract) 0
```

Risk Level:

Likelihood - 4
Impact - 4

Recommendation:

It is recommended to allow users to trade all claimed revenue tokens to pay off the debt.

3.12 (HAL-12) PAYING OFF DEBT WITH SPIGOT EARNING IN ETHER IS NOT POSSIBLE - HIGH

Description:

In the SpigotedLoan contract, the borrower can add spigots with revenue contracts that will support repayment of the debt. The revenue contract earns revenue tokens that later can be exchanged for the credit tokens using the claimAndRepay() and claimAndTrade() functions. These functions call the _claimAndTrade() internal function responsible for claiming escrowed tokens from the SpigotController contract and trading the claimed tokens. It is possible to add a spigot with revenue contracts that earns revenue in ether. However, the assessment revealed that claiming and trading ether from a spigot is not possible in the _claimAndTrade() function. This function assumes that the revenue token is an ERC20 token in every case. As a result, the escrowed ether remains locked until the debt is paid off by other means, and the borrower can transfer it from the SpigotController contract.

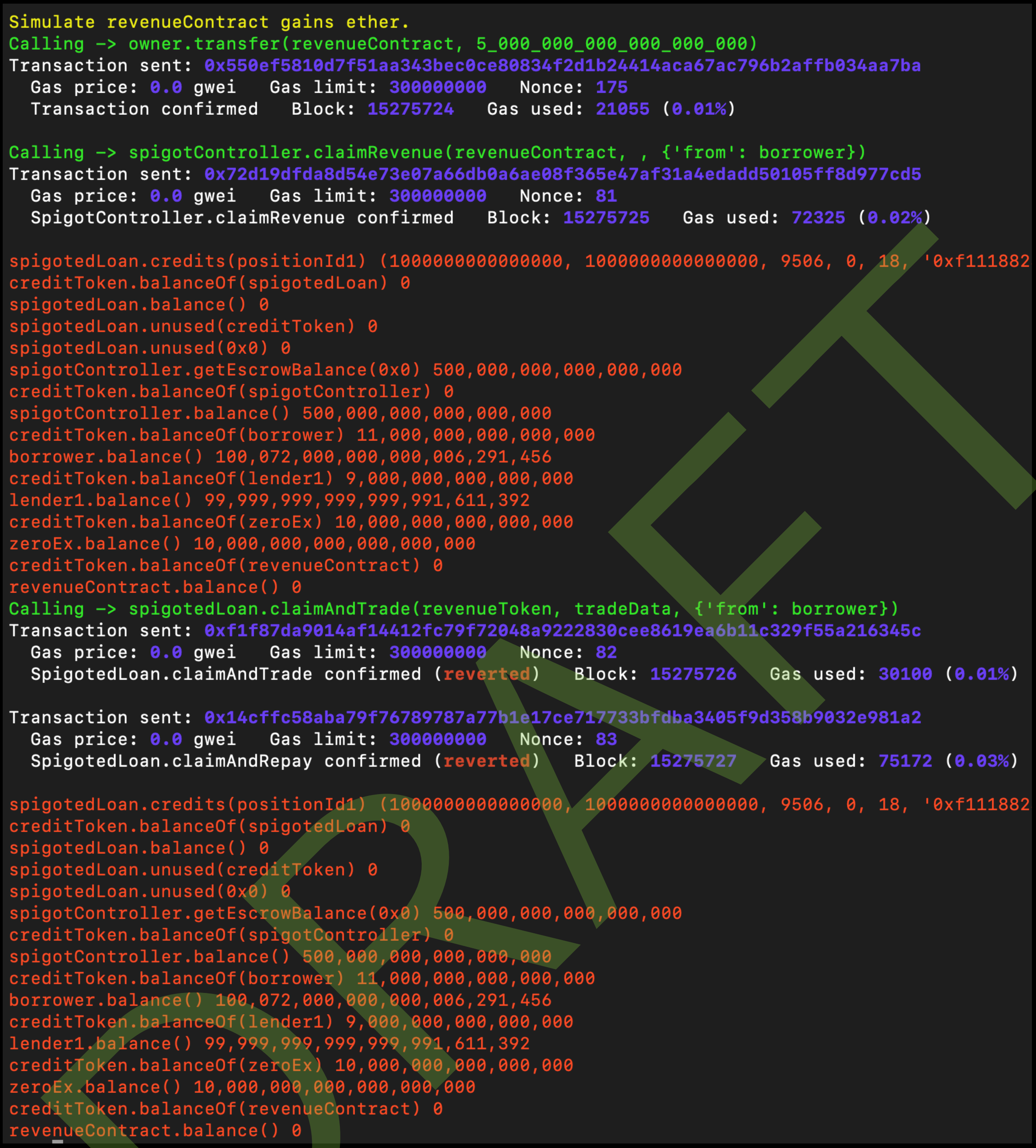
Listing 24: SpigotedLoan.sol (Lines 160-162,201-203)

```
155 function _claimAndTrade(
156         address claimToken,
157         address targetToken,
158         bytes calldata zeroExTradeData
159     ) internal returns (uint256 tokensBought) {
160         uint256 existingClaimTokens = IERC20(claimToken).balanceOf(
161             address(this)
162         );
163         uint256 existingTargetTokens = IERC20(targetToken).balanceOf(
164             address(this)
165         );
166
167         uint256 tokensClaimed = spigot.claimEscrow(claimToken);
168
169         if (claimToken == address(0)) {
```

```
            // if claiming/trading eth send as msg.value to dex
            (bool success, ) = swapTarget.call{value: tokensClaimed}(
                zeroExTradeData
            );
            require(success, "SpigotCnsm: trade failed");
        } else {
            IERC20(claimToken).approve(
                swapTarget,
                existingClaimTokens + tokensClaimed
            );
            (bool success, ) = swapTarget.call(zeroExTradeData);
            require(success, "SpigotCnsm: trade failed");
        }

        uint256 targetTokens = IERC20(targetToken).balanceOf(address(this));

        // ideally we could use oracle to calculate # of tokens to receive
        // but claimToken might not have oracle. targetToken must have oracle

        // underflow revert ensures we have more tokens than we started with
        tokensBought = targetTokens - existingTargetTokens;

        emit TradeSpigotRevenue(
            claimToken,
            tokensClaimed,
            targetToken,
            tokensBought
        );

        // update unused if we didnt sell all claimed tokens in trade
        // also underflow revert protection here
        unusedTokens[claimToken] +=
            IERC20(claimToken).balanceOf(address(this)) -
            existingClaimTokens;
    }
```

Proof of Concept:

1. All necessary contracts are deployed and initialized: CreditToken, RevenueToken, SimpleOracle, LoanLib, SpigotedLoan, SimpleRevenueContract.
2. As the borrower and lender1, add credit position for 10_000_000_-000_000_000 tokens.
3. As the borrower, borrow() all deposits.
4. As the borrower and arbiter, add a new spigot with addSpigot() function and RevenueContract as input. Set the revenue token to zero address (0x0). Note that zero address represents ether in the SpigotController contract.
5. As the borrower, transfer the ownership of the RevenueContract contract to the SpigotController.
6. Transfer 5_000_000_000_000_000_000 ether to the RevenueContract contract to simulate ether gain.
7. As the borrower, call claimRevenue() in SpigotController contract to claim ether.
8. As the borrower, attempt to call claimAndTrade() or claimAndRepay() in the SpigotedLoan contract to trade escrowed ether and pay off part of the debt.
9. Observe that the above transaction reverts. Note that ether remains in SpigotController.



```
Simulate revenueContract gains ether.
Calling -> owner.transfer(revenueContract, 5_000_000_000_000_000_000)
Transaction sent: 0x550ef5810d7f51aa343bec0ce80834f2d1b24414aca67ac796b2affb034aa7ba
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 175
  Transaction confirmed   Block: 15275724   Gas used: 21055 (0.01%)

Calling -> spigotController.claimRevenue(revenueContract, , {'from': borrower})
Transaction sent: 0x72d19dfda8d54e73e07a66db0a6ae08f365e47af31a4edadd50105ff8d977cd5
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 81
  SpigotController.claimRevenue confirmed   Block: 15275725   Gas used: 72325 (0.02%)

spigotedLoan.credits(positionId1) (1000000000000000, 1000000000000000, 9506, 0, 18, '0xf111882
creditToken.balanceOf(spigotedLoan) 0
spigotedLoan.balance() 0
spigotedLoan.unused(creditToken) 0
spigotedLoan.unused(0x0) 0
spigotController.getEscrowBalance(0x0) 500,000,000,000,000,000
creditToken.balanceOf(spigotController) 0
spigotController.balance() 500,000,000,000,000,000
creditToken.balanceOf(borrower) 11,000,000,000,000,000
borrower.balance() 100,072,000,000,000,006,291,456
creditToken.balanceOf(lender1) 9,000,000,000,000,000
lender1.balance() 99,999,999,999,999,991,611,392
creditToken.balanceOf(zeroEx) 10,000,000,000,000,000
zeroEx.balance() 10,000,000,000,000,000,000
creditToken.balanceOf(revenueContract) 0
revenueContract.balance() 0
Calling -> spigotedLoan.claimAndTrade(revenueToken, tradeData, {'from': borrower})
Transaction sent: 0xf1f87da9014af14412fc79f72048a9222830cee8619ea6b11c329f55a216345c
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 82
  SpigotedLoan.claimAndTrade confirmed (reverted)   Block: 15275726   Gas used: 30100 (0.01%)

Transaction sent: 0x14cffc58aba79f76789787a77b1e17ce717733bfdba3405f9d358b9032e981a2
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 83
  SpigotedLoan.claimAndRepay confirmed (reverted)   Block: 15275727   Gas used: 75172 (0.03%)

spigotedLoan.credits(positionId1) (1000000000000000, 1000000000000000, 9506, 0, 18, '0xf111882
creditToken.balanceOf(spigotedLoan) 0
spigotedLoan.balance() 0
spigotedLoan.unused(creditToken) 0
spigotedLoan.unused(0x0) 0
spigotController.getEscrowBalance(0x0) 500,000,000,000,000,000
creditToken.balanceOf(spigotController) 0
spigotController.balance() 500,000,000,000,000,000
creditToken.balanceOf(borrower) 11,000,000,000,000,000
borrower.balance() 100,072,000,000,000,006,291,456
creditToken.balanceOf(lender1) 9,000,000,000,000,000
lender1.balance() 99,999,999,999,999,991,611,392
creditToken.balanceOf(zeroEx) 10,000,000,000,000,000
zeroEx.balance() 10,000,000,000,000,000,000
creditToken.balanceOf(revenueContract) 0
revenueContract.balance() 0
```

Risk Level:

Likelihood - 5

Impact - 4

Recommendation:

It is recommended to adjust the _claimAndTrade() internal function to check the proper balance when ether is used as a claim token.

3.13 (HAL-13) DOUBLE UPDATE OF UNUSEDTOKENS COLLECTION POSSIBLE - HIGH

Description:

In the SpigotedLoan contract, the borrower can add spigots with revenue contracts that will support repayment of the debt. The revenue contract earns revenue tokens that later can be exchanged to the credit tokens using the claimAndRepay() and claimAndTrade() functions. These functions call the _claimAndTrade() internal function responsible for claiming escrowed tokens from the SpigotController contract and trading the claimed tokens. Also, the claimAndTrade() function updates the unusedTokens collection with target tokens (credit tokens) that were traded. It is possible to add a new spigot with the revenue contract that earns in credit tokens. Then, the assumption is that in the _claimAndTrade() function, the trade of the tokens is done with a ratio of 1-to-1, or the decentralized exchange contract simply returns a false value (instead of revert). This scenario was confirmed by the project team. As a result, the unusedTokens collection is updated twice for credit tokens, first time in the _claimAndTrade() function and second time in the claimAndTrade() function.

Listing 25: SpigotedLoan.sol (Lines 152,176-180,201-203)

```
    function claimAndTrade(address claimToken, bytes calldata
 ↳ zeroExTradeData)
        external
        whileBorrowing
        returns (uint256 tokensBought)
    {
        require(msg.sender == borrower || msg.sender == arbiter);

        address targetToken = credits[ids[0]].token;
        tokensBought = _claimAndTrade(
            claimToken,
            targetToken,
            zeroExTradeData
        );

        // add bought tokens to unused balance
        unusedTokens[targetToken] += tokensBought;
    }

function _claimAndTrade(
        address claimToken,
        address targetToken,
        bytes calldata zeroExTradeData
    ) internal returns (uint256 tokensBought) {
        uint256 existingClaimTokens = IERC20(claimToken).balanceOf
↳ (
            address(this)
        );
        uint256 existingTargetTokens = IERC20(targetToken).
↳ balanceOf(
            address(this)
        );

        uint256 tokensClaimed = spigot.claimEscrow(claimToken);

        if (claimToken == address(0)) {
            // if claiming/trading eth send as msg.value to dex
            (bool success, ) = swapTarget.call{value:
↳ tokensClaimed}(
                zeroExTradeData
            );
            require(success, "SpigotCnsm: trade failed");
        } else {
            IERC20(claimToken).approve(
                swapTarget,
                existingClaimTokens + tokensClaimed
            );
            (bool success, ) = swapTarget.call(zeroExTradeData);
            require(success, "SpigotCnsm: trade failed");
        }

        uint256 targetTokens = IERC20(targetToken).balanceOf(
↳ address(this));

        // ideally we could use oracle to calculate # of tokens to
↳  receive
```

```
187         // but claimToken might not have oracle. targetToken must
  ↳ have oracle
188
189         // underflow revert ensures we have more tokens than we
  ↳ started with
190         tokensBought = targetTokens - existingTargetTokens;
191
192         emit TradeSpigotRevenue(
193             claimToken,
194             tokensClaimed,
195             targetToken,
196             tokensBought
197         );
198
199         // update unused if we didnt sell all claimed tokens in
  ↳ trade
200         // also underflow revert protection here
201         unusedTokens[claimToken] +=
202             IERC20(claimToken).balanceOf(address(this)) -
203             existingClaimTokens;
204     }
```

This flaw impacts the sweep() function, which reverts as the contract's balance has fewer tokens than recorded in the unusedTokens collection. Thus, the tokens are locked in the contract.

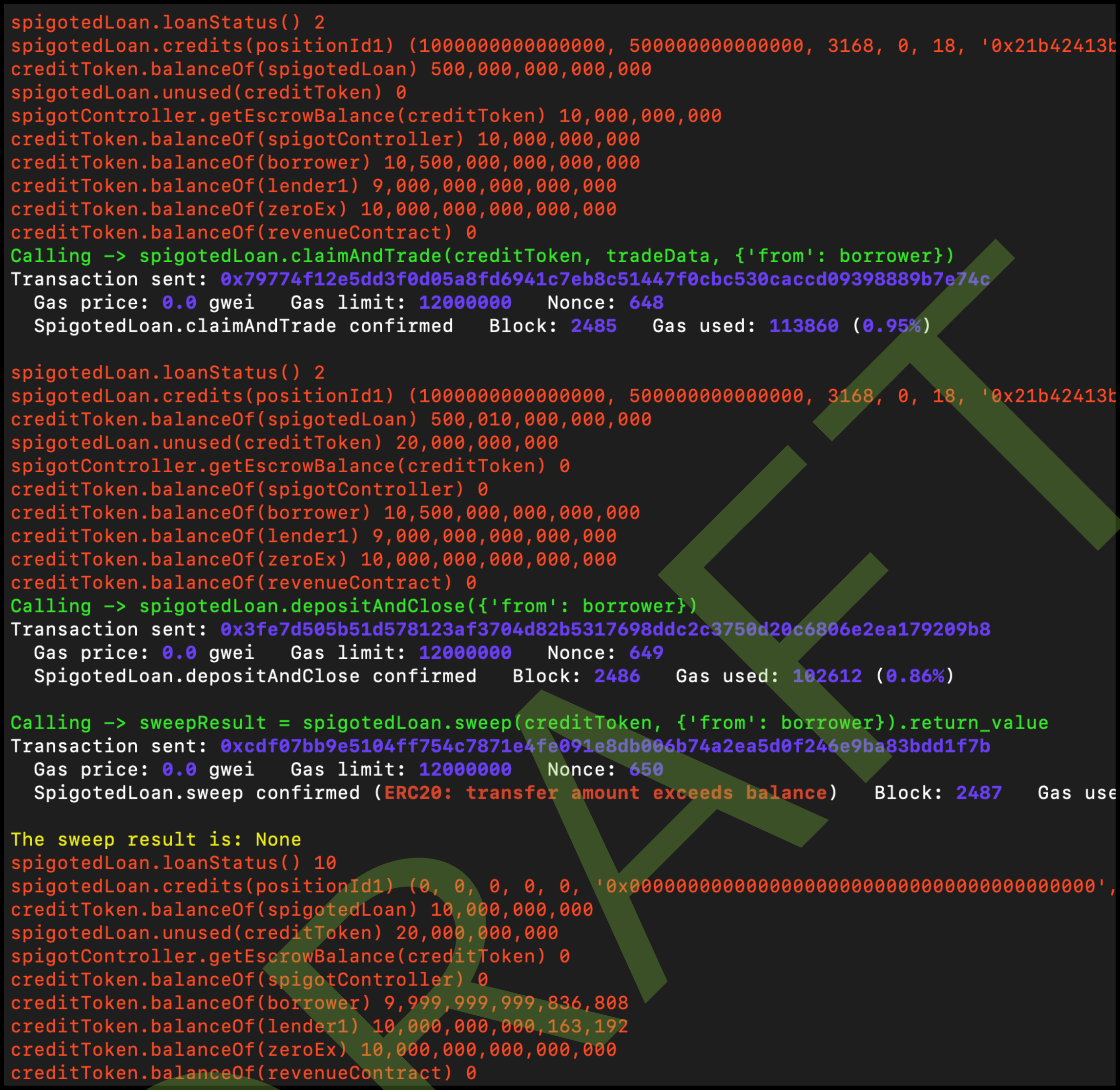
Listing 26: SpigotedLoan.sol (Lines 278,282)

```
266     function sweep(address token) external returns (uint256) {
267         if (loanStatus == LoanLib.STATUS.REPAID) {
268             return _sweep(borrower, token);
269         }
270         if (loanStatus == LoanLib.STATUS.INSOLVENT) {
271             return _sweep(arbiter, token);
272         }
273
274         return 0;
275     }
276
277     function _sweep(address to, address token) internal returns (
  ↳ uint256 x) {
278         x = unusedTokens[token];
279         if (token == address(0)) {
```

```
            payable(to).transfer(x);
        } else {
            require(IERC20(token).transfer(to, x));
        }
        delete unusedTokens[token];
    }
```

Proof of Concept:

1. All necessary contracts are deployed and initialized: CreditToken, RevenueToken, SimpleOracle, LoanLib, SpigotedLoan, SimpleRevenueContract. Note that SimpleRevenueContract must earn in credit tokens.
2. As borrower and lender1, add credit position for 10_000_000_000_000_000 tokens.
3. As borrower, borrow() half of the deposit.
4. As borrower and arbiter, add new spigot with addSpigot() function and RevenueContract as input. Set the ownerSplit parameter to 100.
5. As the borrower, transfer the ownership of the RevenueContract contract to the SpigotController.
6. Mint 10_000_000_000 credit tokens to the RevenueContract contract to simulate token gain.
7. As the borrower, call claimRevenue() in SpigotController contract to claim revenue tokens. Note that 100% of tokens (10_000_000_000) are transferred to the SpigotController.
8. As the borrower, call claimAndTrade() in SpigotedLoan contract to trade escrowed credit tokens. As an input, trade exchange data provide 10_000_000_000 revenue tokens (credit tokens) that should be exchanged for credit tokens.
9. Observe the SpigotedLoan balances. Note that the contract has 500,010,000,000,000 credit tokens. Also, 20,000,000,000 credit tokens are stored in unusedTokens collection.
10. As the borrower, call depositAndClose(). Observe that credit is paid off. Note that the contract has 10,000,000,000 credit tokens.
11. As the borrower, attempt to call sweep() function with credit token address as an input. Note that the function reverts the ERC20: transfer amount exceeds balance error.



```
spigotedLoan.loanStatus() 2
spigotedLoan.credits(positionId1) (1000000000000000, 500000000000000, 3168, 0, 18, '0x21b42413b
creditToken.balanceOf(spigotedLoan) 500,000,000,000,000
spigotedLoan.unused(creditToken) 0
spigotController.getEscrowBalance(creditToken) 10,000,000,000
creditToken.balanceOf(spigotController) 10,000,000,000
creditToken.balanceOf(borrower) 10,500,000,000,000,000
creditToken.balanceOf(lender1) 9,000,000,000,000,000
creditToken.balanceOf(zeroEx) 10,000,000,000,000,000
creditToken.balanceOf(revenueContract) 0
Calling -> spigotedLoan.claimAndTrade(creditToken, tradeData, {'from': borrower})
Transaction sent: 0x79774f12e5dd3f0d05a8fd6941c7eb8c51447f0cbc530caccd09398889b7e74c
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 648
  SpigotedLoan.claimAndTrade confirmed   Block: 2485   Gas used: 113860 (0.95%)

spigotedLoan.loanStatus() 2
spigotedLoan.credits(positionId1) (1000000000000000, 500000000000000, 3168, 0, 18, '0x21b42413b
creditToken.balanceOf(spigotedLoan) 500,010,000,000,000
spigotedLoan.unused(creditToken) 20,000,000,000
spigotController.getEscrowBalance(creditToken) 0
creditToken.balanceOf(spigotController) 0
creditToken.balanceOf(borrower) 10,500,000,000,000,000
creditToken.balanceOf(lender1) 9,000,000,000,000,000
creditToken.balanceOf(zeroEx) 10,000,000,000,000,000
creditToken.balanceOf(revenueContract) 0
Calling -> spigotedLoan.depositAndClose({'from': borrower})
Transaction sent: 0x3fe7d505b51d578123af3704d82b5317698ddc2c3750d20c6806e2ea179209b8
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 649
  SpigotedLoan.depositAndClose confirmed   Block: 2486   Gas used: 102612 (0.86%)

Calling -> sweepResult = spigotedLoan.sweep(creditToken, {'from': borrower}).return_value
Transaction sent: 0xcdf07bb9e5104ff754c7871e4fe091e8db006b74a2ea5d0f246e9ba83bdd1f7b
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 650
  SpigotedLoan.sweep confirmed (ERC20: transfer amount exceeds balance)   Block: 2487   Gas use

The sweep result is: None
spigotedLoan.loanStatus() 10
spigotedLoan.credits(positionId1) (0, 0, 0, 0, 0, '0x0000000000000000000000000000000000000000',
creditToken.balanceOf(spigotedLoan) 10,000,000,000
spigotedLoan.unused(creditToken) 20,000,000,000
spigotController.getEscrowBalance(creditToken) 0
creditToken.balanceOf(spigotController) 0
creditToken.balanceOf(borrower) 9,999,999,999,836,808
creditToken.balanceOf(lender1) 10,000,000,000,163,192
creditToken.balanceOf(zeroEx) 10,000,000,000,000,000
creditToken.balanceOf(revenueContract) 0
```

Risk Level:

Likelihood - 4

Impact - 4

Recommendation:

When the spigot's revenue contract earns in credit tokens as revenue tokens, the unusedTokens should be updated only once.

3.14 (HAL-14) UNEXPECTED LIQUIDATABLE STATUS IN NEW ESCROWEDLOAN - HIGH

Description:

In the EscrowedLoan contract, the _healthcheck() internal function is supposed to check if the loan has the correct collateral ratio. Otherwise, the loan status is set to LIQUIDATABLE. The _healthcheck() function base on the _getLatestCollateralRatio() function, which returns 0 for empty collateral. Therefore, calling the healthcheck() function at the beginning of the loan's life cycle will set the EscrowedLoan contract's status to LIQUIDATABLE.

When contract has LIQUIDATABLE status several spigot's functions can be called without authorisation: releaseSpigot(), updateOwnerSplit(), sweep(). As a result, the loan can get malformed state:
- the spigot's owner can be set to the arbiter, and it will not be usable by EscrowedLoan (SpigotedLoan),
- the spigot's split can be set to 100%,
- the unused tokens from EscrowedLoan (SpigotedLoan) can be transferred to the arbiter if only some tokens were stored in the contract.

The borrower can set the contract's status back to ACTIVE by adding collateral and repeatedly calling the healthcheck() function.

Listing 27: EscrowedLoan.sol (Line 26)

```
25  function _healthcheck() virtual internal returns(LoanLib.STATUS) {
26      if(escrow.getCollateralRatio() < escrow.minimumCollateralRatio()) {
27          return LoanLib.STATUS.LIQUIDATABLE;
28      }
29
30      return LoanLib.STATUS.ACTIVE;
31  }
```

Listing 28: Escrow.sol

```
    /**
     * @notice calculates the cratio
     * @dev callable by anyone
     * @return - the calculated cratio
     */
    function getCollateralRatio() external returns (uint256) {
        return _getLatestCollateralRatio();
    }
```

Listing 29: Escrow.sol (Line 52)

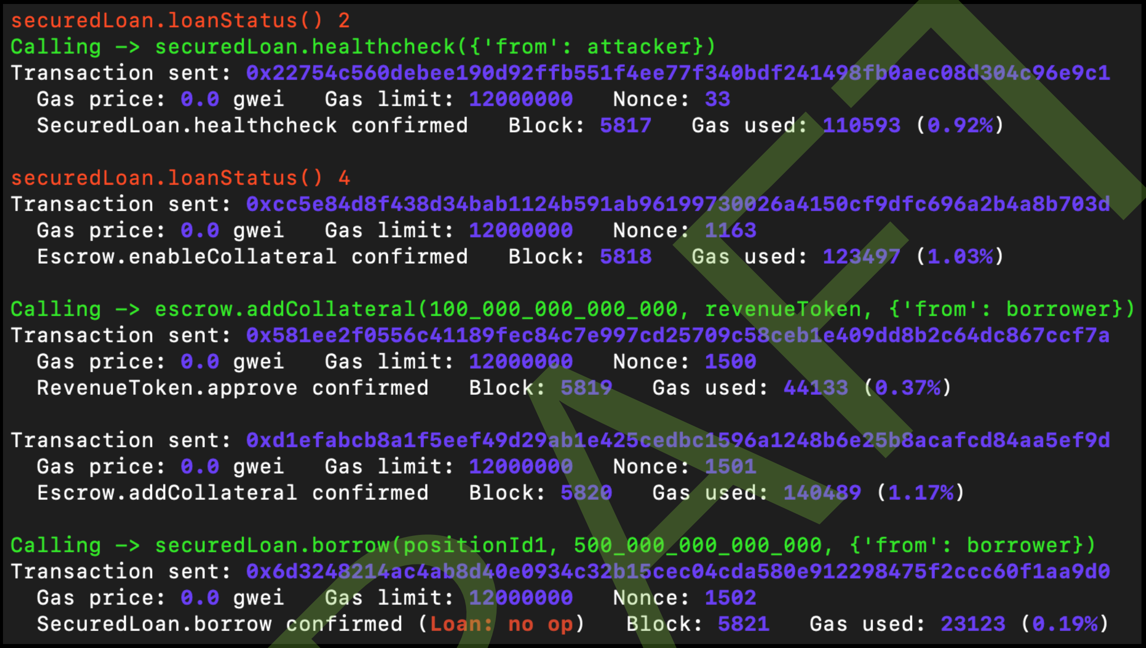
```
43      /**
44       * @notice updates the cratio according to the collateral value vs loan value
45       * @dev calls accrue interest on the loan contract to update the latest interest payable
46       * @return the updated collateral ratio in 18 decimals
47       */
48      function _getLatestCollateralRatio() internal returns (uint256) {
49          ILoan(loan).accrueInterest();
50          uint256 debtValue = ILoan(loan).getOutstandingDebt();
51          uint256 collateralValue = _getCollateralValue();
52          if (collateralValue == 0) return 0;
53          if (debtValue == 0) return MAX_INT;
54
55          return _percent(collateralValue, debtValue, 18);
56      }
```

Proof of Concept:

1. All necessary contracts are deployed and initialized: CreditToken, RevenueToken, SimpleOracle, LoanLib, EscrowedLoan.
2. As borrower and lender, add credit position.
3. As borrower and arbiter, add new spigot with addSpigot() function and RevenueContract as input. Set the ownerSplit parameter to 10.
4. As the borrower, transfer the ownership of the RevenueContract contract to the SpigotController.
5. As the attacker, call thehealthcheck() function. Note that the

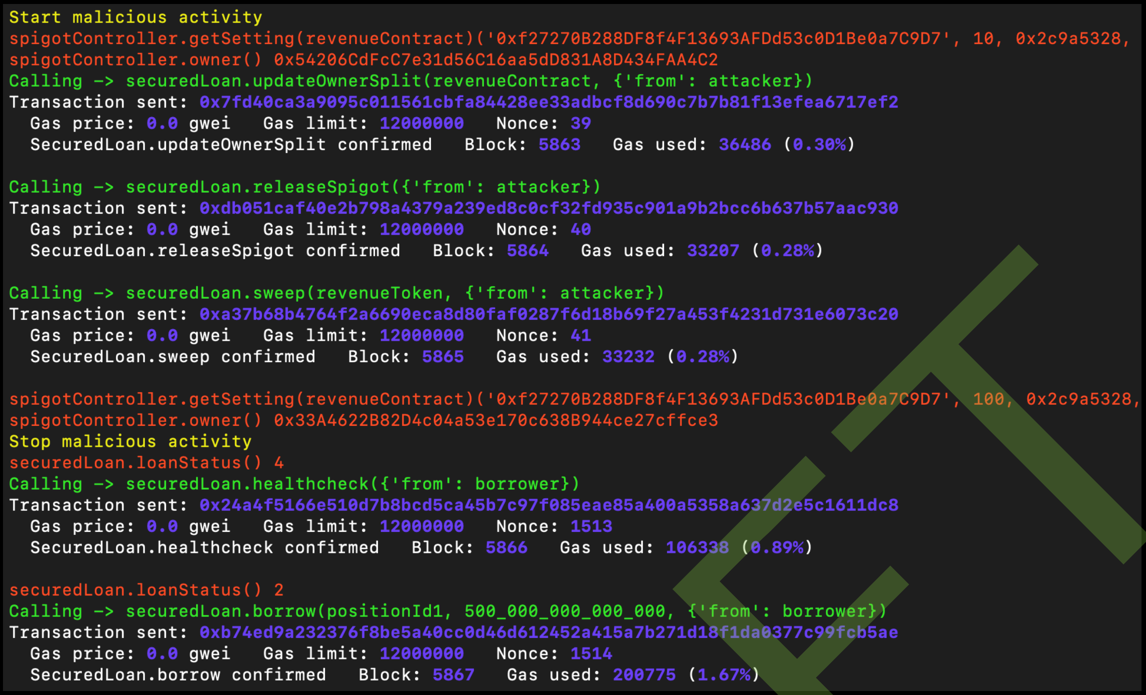
loan’s status is set to LIQUIDATABLE.

6. As the arbiter, enable the RevenueToken token as collateral.
7. As the borrower, add 200_000_000_000_000 of RevenueToken tokens as collateral.
8. As the borrower, attempt to borrow all deposit. Observe that transaction reverts due to the (Loan: no op) error.



```
securedLoan.loanStatus() 2
Calling -> securedLoan.healthcheck({'from': attacker})
Transaction sent: 0x22754c560debee190d92ffb551f4ee77f340bdf241498fb0aec08d304c96e9c1
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 33
  SecuredLoan.healthcheck confirmed   Block: 5817   Gas used: 110593 (0.92%)

securedLoan.loanStatus() 4
Transaction sent: 0xcc5e84d8f438d34bab1124b591ab96199730026a4150cf9dfc696a2b4a8b703d
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1163
  Escrow.enableCollateral confirmed   Block: 5818   Gas used: 123497 (1.03%)

Calling -> escrow.addCollateral(100_000_000_000_000, revenueToken, {'from': borrower})
Transaction sent: 0x581ee2f0556c41189fec84c7e997cd25709c58ceb1e409dd8b2c64dc867ccf7a
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1500
  RevenueToken.approve confirmed   Block: 5819   Gas used: 44133 (0.37%)

Transaction sent: 0xd1efabcb8a1f5eef49d29ab1e425cedbc1596a1248b6e25b8acafcd84aa5ef9d
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1501
  Escrow.addCollateral confirmed   Block: 5820   Gas used: 140489 (1.17%)

Calling -> securedLoan.borrow(positionId1, 500_000_000_000_000, {'from': borrower})
Transaction sent: 0x6d3248214ac4ab8d40e0934c32b15cec04cda580e912298475f2ccc60f1aa9d0
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1502
  SecuredLoan.borrow confirmed (Loan: no op)   Block: 5821   Gas used: 23123 (0.19%)
```

9. As the attacker, call updateOwnerSplit() function. Observe that owner split is now set to 100.
10. As the attacker, call releaseSpigot() function. Observe that spigot’s owner’s address has been changed.
11. As the borrower, call thehealthcheck() function. Note that the loan’s status is set to ACTIVE.
12. As the borrower, borrow all deposits. Observe that transaction finishes successfully.



```
Start malicious activity
spigotController.getSetting(revenueContract)('0xf27270B288DF8f4F13693AFDd53c0D1Be0a7C9D7', 10, 0x2c9a5328,
spigotController.owner() 0x54206CdFcC7e31d56C16aa5dD831A8D434FAA4C2
Calling -> securedLoan.updateOwnerSplit(revenueContract, {'from': attacker})
Transaction sent: 0x7fd40ca3a9095c011561cbfa84428ee33adbcf8d690c7b7b81f13efea6717ef2
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 39
  SecuredLoan.updateOwnerSplit confirmed   Block: 5863   Gas used: 36486 (0.30%)

Calling -> securedLoan.releaseSpigot({'from': attacker})
Transaction sent: 0xdb051caf40e2b798a4379a239ed8c0cf32fd935c901a9b2bcc6b637b57aac930
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 40
  SecuredLoan.releaseSpigot confirmed   Block: 5864   Gas used: 33207 (0.28%)

Calling -> securedLoan.sweep(revenueToken, {'from': attacker})
Transaction sent: 0xa37b68b4764f2a6690eca8d80faf0287f6d18b69f27a453f4231d731e6073c20
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 41
  SecuredLoan.sweep confirmed   Block: 5865   Gas used: 33232 (0.28%)

spigotController.getSetting(revenueContract)('0xf27270B288DF8f4F13693AFDd53c0D1Be0a7C9D7', 100, 0x2c9a5328,
spigotController.owner() 0x33A4622B82D4c04a53e170c638B944ce27cffce3
Stop malicious activity
securedLoan.loanStatus() 4
Calling -> securedLoan.healthcheck({'from': borrower})
Transaction sent: 0x24a4f5166e510d7b8bcd5ca45b7c97f085eae85a400a5358a637d2e5c1611dc8
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1513
  SecuredLoan.healthcheck confirmed   Block: 5866   Gas used: 106338 (0.89%)

securedLoan.loanStatus() 2
Calling -> securedLoan.borrow(positionId1, 500_000_000_000_000, {'from': borrower})
Transaction sent: 0xb74ed9a232376f8be5a40cc0d46d612452a415a7b271d18f1da0377c99fcb5ae
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1514
  SecuredLoan.borrow confirmed   Block: 5867   Gas used: 200775 (1.67%)
```

Risk Level:

Likelihood - 4

Impact - 4

Recommendation:

It is recommended to forbid changing the new loan’s status to LIQUIDATABLE .

3.15 (HAL-15) CANNOT LIQUIDATE LIQUIDATABLE SECUREDLOAN DUE TO COLLATERAL RATIO CHECK - HIGH

Description:

The SecuredLoan can get the LIQUIDATABLE status in two cases: when the loan's deadline has passed or when the collateral ratio is below the threshold. When the loan has the LIQUIDATABLE status, the arbiter can take control over the spigots and collateral and use the released funds to repay the debt for the lender. The assessment revealed that in some cases, the arbiter could not liquidate the collateral when the loan's deadline has passed, and the loan gets the LIQUIDATABLE status. The Escrow's liquidate() function has an assertion that checks if the collateral ratio value is below the threshold. The collateral ratio decreases while the loan's interest increases. Thus, it may take time to get the ratio low enough.
Additionally, the borrower can still release part of the collateral within this time.

Listing 30: SecuredLoan.sol (Line 57)

```
  // Liquidation
  /**
   * @notice - Forcefully take collateral from borrower and repay debt for lender
   * @dev - only called by neutral arbiter party/contract
   * @dev - `loanStatus` must be LIQUIDATABLE
   * @dev - callable by `arbiter`
   * @param positionId -the debt position to pay down debt on
   * @param amount - amount of `targetToken` expected to be sold off in  _liquidate
   * @param targetToken - token in escrow that will be sold of to repay position
   */

  function liquidate(
    bytes32 positionId,
    uint256 amount,
48     address targetToken
49   )
50     external
51     returns(uint256)
52   {
53     require(msg.sender == arbiter);
54
55     _updateLoanStatus(_healthcheck());
56
57     require(loanStatus == LoanLib.STATUS.LIQUIDATABLE, "Loan: not
 ↳ liquidatable");
58
59     // send tokens to arbiter for OTC sales
60     return _liquidate(positionId, amount, targetToken, msg.sender)
 ↳ ;
61   }
62
```

Listing 31: EscrowedLoan.sol (Line 53)

```
44   function _liquidate(
45     bytes32 positionId,
46     uint256 amount,
47     address targetToken,
48     address to
49   )
50     virtual internal
51     returns(uint256)
52   {
53     require(escrow.liquidate(amount, targetToken, to));
54
55     emit Liquidate(positionId, amount, targetToken);
56
57     return amount;
58   }
```

Listing 32: Escrow.sol (Lines 263-266)

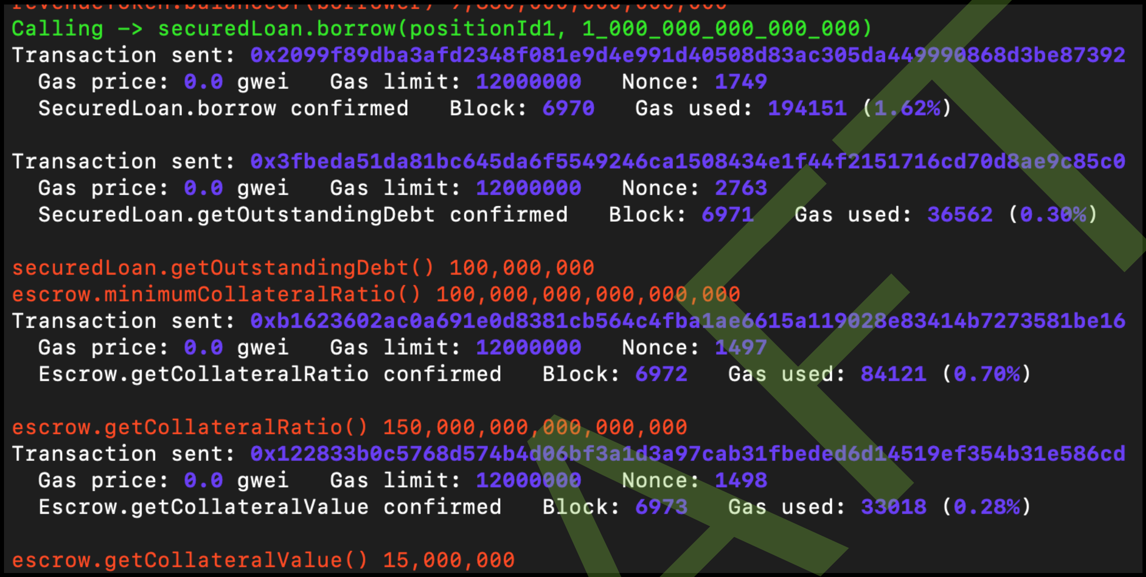
```
244 /**
245      * @notice liquidates borrowers collateral by token and amount
246      * @dev requires that the cratio is at or below the
 ↳ liquidation threshold
247      * @dev callable by `loan`
```

```
     * @param amount - the amount of tokens to liquidate
     * @param token - the address of the token to draw funds from
     * @param to - the address to receive the funds
     * @return - true if successful
     */
    function liquidate(
        uint256 amount,
        address token,
        address to
    ) external returns (bool) {
        require(amount > 0, "Escrow: amount is 0");
        require(
            msg.sender == loan,
            "Escrow: msg.sender must be the loan contract"
        );
        require(
            minimumCollateralRatio > _getLatestCollateralRatio(),
            "Escrow: not eligible for liquidation"
        );
        require(
            deposited[token].amount >= amount,
            "Escrow: insufficient balance"
        );

        deposited[token].amount -= amount;
        require(IERC20(token).transfer(to, amount));

        emit Liquidate(token, amount);

        return true;
    }
```

Proof of Concept:

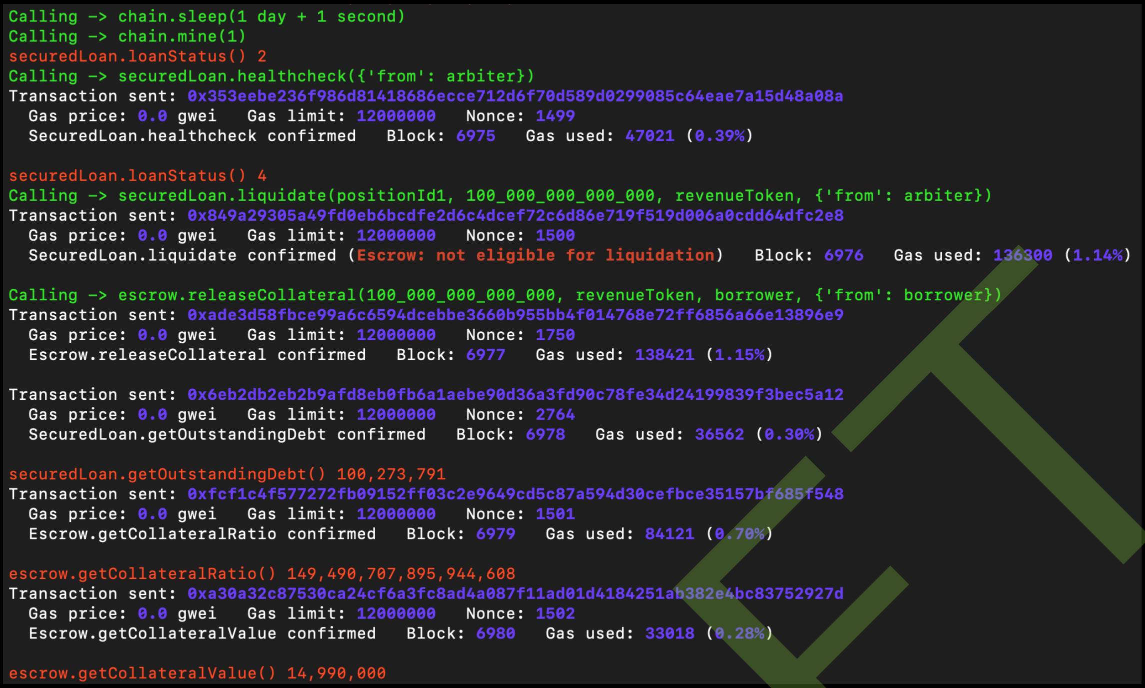
1. All necessary contracts are deployed and initialized: CreditToken, RevenueToken, SimpleOracle, LoanLib, EscrowedLoan. Set the ttl parameter to 1 day. Set the minimumCollateralRatio parameter to 10%.
2. As the borrower and lender1, add credit position for 1_000_000_000_-000_000 tokens withdrawn rate set to 10000 and facility rate set to 100.

3. As the arbiter, enable the RevenueToken token as collateral.
4. As the borrower, add 150_000_000_000_000 of RevenueToken tokens as collateral.
5. As the borrower, borrow all deposits. Note that the collateral ratio value is ‘15%‘‘.



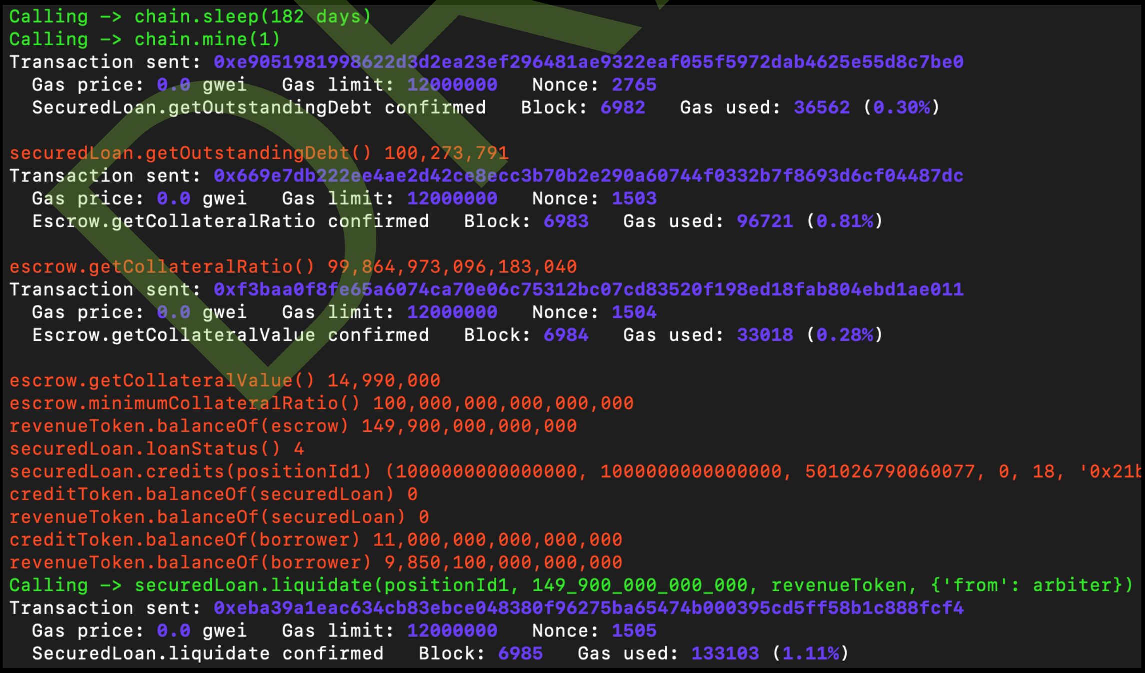
```
revenueToken.balanceOf(borrower) 7,550,000,000,000,000
Calling -> securedLoan.borrow(positionId1, 1_000_000_000_000_000)
Transaction sent: 0x2099f89dba3afd2348f081e9d4e991d40508d83ac305da449990868d3be87392
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1749
  SecuredLoan.borrow confirmed   Block: 6970   Gas used: 194151 (1.62%)

Transaction sent: 0x3fbeda51da81bc645da6f5549246ca1508434e1f44f2151716cd70d8ae9c85c0
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2763
  SecuredLoan.getOutstandingDebt confirmed   Block: 6971   Gas used: 36562 (0.30%)

securedLoan.getOutstandingDebt() 100,000,000
escrow.minimumCollateralRatio() 100,000,000,000,000,000
Transaction sent: 0xb1623602ac0a691e0d8381cb564c4fba1ae6615a119028e83414b7273581be16
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1497
  Escrow.getCollateralRatio confirmed   Block: 6972   Gas used: 84121 (0.70%)

escrow.getCollateralRatio() 150,000,000,000,000,000
Transaction sent: 0x122833b0c5768d574b4d06bf3a1d3a97cab31fbeded6d14519ef354b31e586cd
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1498
  Escrow.getCollateralValue confirmed   Block: 6973   Gas used: 33018 (0.28%)

escrow.getCollateralValue() 15,000,000
```

6. Forward blockchain time for 1 day and 1 second.
7. As the arbiter, call thehealthcheck() function. Note that the loan’s status is set to LIQUIDATABLE.
8. As the arbiter, attempt to call liquidate() function. Observe that the transaction reverts with the Escrow: not eligible for liquidation error.
9. As the borrower, call releaseCollateral() function. Observe that transaction finishes successfully. Note that the collateral ratio value is ‘14,9%‘‘.



```
Calling -> chain.sleep(1 day + 1 second)
Calling -> chain.mine(1)
securedLoan.loanStatus() 2
Calling -> securedLoan.healthcheck({'from': arbiter})
Transaction sent: 0x353eebe236f986d81418686ecce712d6f70d589d0299085c64eae7a15d48a08a
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1499
  SecuredLoan.healthcheck confirmed   Block: 6975   Gas used: 47021 (0.39%)

securedLoan.loanStatus() 4
Calling -> securedLoan.liquidate(positionId1, 100_000_000_000_000, revenueToken, {'from': arbiter})
Transaction sent: 0x849a29305a49fd0eb6bcdfe2d6c4dcef72c6d86e719f519d006a0cdd64dfc2e8
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1500
  SecuredLoan.liquidate confirmed (Escrow: not eligible for liquidation)   Block: 6976   Gas used: 136300 (1.14%)

Calling -> escrow.releaseCollateral(100_000_000_000_000, revenueToken, borrower, {'from': borrower})
Transaction sent: 0xade3d58fbce99a6c6594dcebbe3660b955bb4f014768e72ff6856a66e13896e9
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1750
  Escrow.releaseCollateral confirmed   Block: 6977   Gas used: 138421 (1.15%)

Transaction sent: 0x6eb2db2eb2b9afd8eb0fb6a1aebe90d36a3fd90c78fe34d24199839f3bec5a12
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2764
  SecuredLoan.getOutstandingDebt confirmed   Block: 6978   Gas used: 36562 (0.30%)

securedLoan.getOutstandingDebt() 100,273,791
Transaction sent: 0xfcf1c4f577272fb09152ff03c2e9649cd5c87a594d30cefbce35157bf685f548
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1501
  Escrow.getCollateralRatio confirmed   Block: 6979   Gas used: 84121 (0.70%)

escrow.getCollateralRatio() 149,490,707,895,944,608
Transaction sent: 0xa30a32c87530ca24cf6a3fc8ad4a087f11ad01d4184251ab382e4bc83752927d
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1502
  Escrow.getCollateralValue confirmed   Block: 6980   Gas used: 33018 (0.28%)

escrow.getCollateralValue() 14,990,000
```

10. Forward blockchain time for 182 days. Note that the collateral ratio value is below 10%.
11. As the arbiter, call liquidate() function. Observe that the transaction finishes successfully.



```
Calling -> chain.sleep(182 days)
Calling -> chain.mine(1)
Transaction sent: 0xe9051981998622d3d2ea23ef296481ae9322eaf055f5972dab4625e55d8c7be0
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2765
  SecuredLoan.getOutstandingDebt confirmed   Block: 6982   Gas used: 36562 (0.30%)

securedLoan.getOutstandingDebt() 100,273,791
Transaction sent: 0x669e7db222ee4ae2d42ce8ecc3b70b2e290a60744f0332b7f8693d6cf04487dc
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1503
  Escrow.getCollateralRatio confirmed   Block: 6983   Gas used: 96721 (0.81%)

escrow.getCollateralRatio() 99,864,973,096,183,040
Transaction sent: 0xf3baa0f8fe65a6074ca70e06c75312bc07cd83520f198ed18fab804ebd1ae011
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1504
  Escrow.getCollateralValue confirmed   Block: 6984   Gas used: 33018 (0.28%)

escrow.getCollateralValue() 14,990,000
escrow.minimumCollateralRatio() 100,000,000,000,000,000
revenueToken.balanceOf(escrow) 149,900,000,000,000
securedLoan.loanStatus() 4
securedLoan.credits(positionId1) (1000000000000000, 1000000000000000, 501026790060077, 0, 18, '0x21b
creditToken.balanceOf(securedLoan) 0
revenueToken.balanceOf(securedLoan) 0
creditToken.balanceOf(borrower) 11,000,000,000,000,000
revenueToken.balanceOf(borrower) 9,850,100,000,000,000
Calling -> securedLoan.liquidate(positionId1, 149_900_000_000_000, revenueToken, {'from': arbiter})
Transaction sent: 0xeba39a1eac634cb83ebce048380f96275ba65474b000395cd5ff58b1c888fcf4
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1505
  SecuredLoan.liquidate confirmed   Block: 6985   Gas used: 133103 (1.11%)
```

Risk Level:

Likelihood - 4
Impact - 4

Recommendation:

It is recommended to allow the call liquidate() function when the loan’s status is set to LIQUIDATABLE despite the collateral ratio value.

3.16 (HAL-16) CREDIT CAN BE CLOSED WITHOUT PAYING INTEREST FROM UNUSED FUNDS - MEDIUM

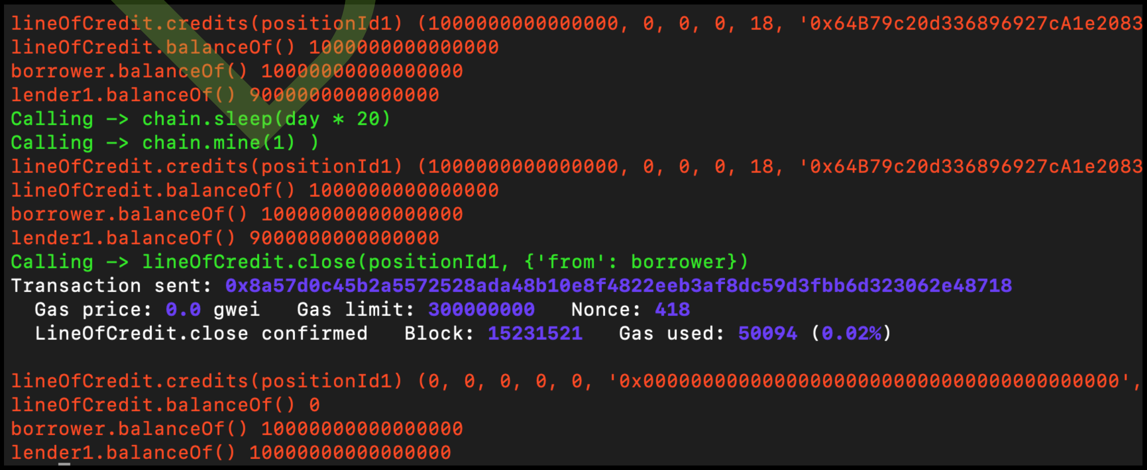
Description:

In the LineOfCredit contract, a borrower and lender can add credit. The algorithm assumes that unused funds accrue interest based on the facility rate (frate parameter). There is no requirement that the borrower must borrow the deposit upon closing the credit. The close(id) function does not call _accrueInterest() internal function. Hence, closing credit without accruing interest from unused funds is possible.

Proof of Concept:

Scenario 1 - without accrueInterest function calling

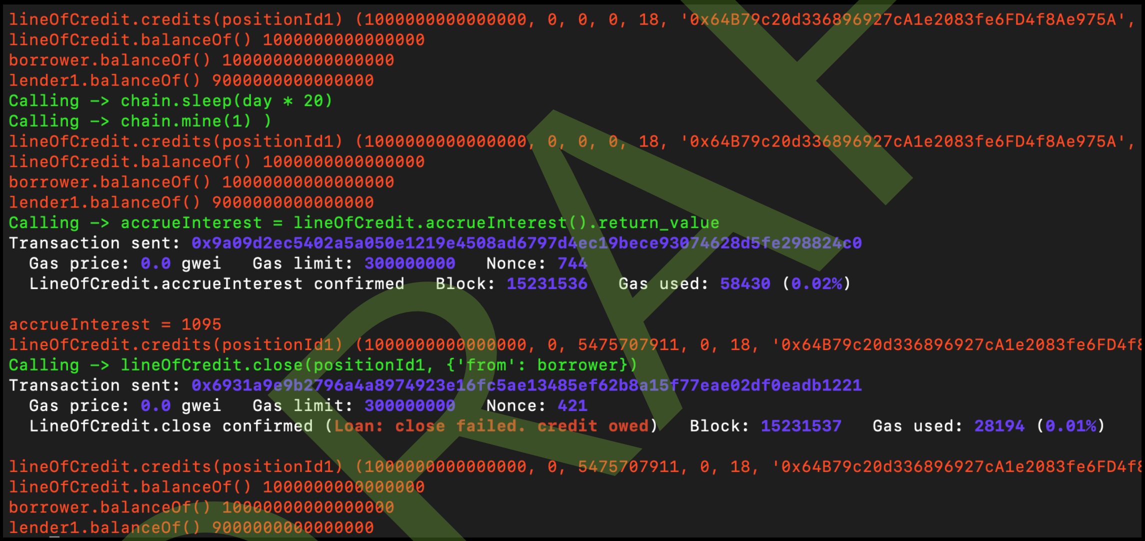
1. All necessary contracts are deployed and initialized: RevenueToken, SimpleOracle, LoanLib, LineOfCredit.
2. As borrower and lender1, add credit position for 10_000_000_000_-000_000 tokens with facility rate set to 1.
3. Forward blockchain time for 10 days.
4. As the borrower, close the credit. Observe that credit is closed without accruing interest from unused funds.



```
lineOfCredit.credits(positionId1) (1000000000000000, 0, 0, 0, 18, '0x64B79c20d336896927cA1e2083
lineOfCredit.balanceOf() 1000000000000000
borrower.balanceOf() 10000000000000000
lender1.balanceOf() 9000000000000000
Calling -> chain.sleep(day * 20)
Calling -> chain.mine(1) )
lineOfCredit.credits(positionId1) (1000000000000000, 0, 0, 0, 18, '0x64B79c20d336896927cA1e2083
lineOfCredit.balanceOf() 1000000000000000
borrower.balanceOf() 10000000000000000
lender1.balanceOf() 9000000000000000
Calling -> lineOfCredit.close(positionId1, {'from': borrower})
Transaction sent: 0x8a57d0c45b2a5572528ada48b10e8f4822eeb3af8dc59d3fbb6d323062e48718
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 418
  LineOfCredit.close confirmed   Block: 15231521   Gas used: 50094 (0.02%)

lineOfCredit.credits(positionId1) (0, 0, 0, 0, 0, '0x0000000000000000000000000000000000000000',
lineOfCredit.balanceOf() 0
borrower.balanceOf() 10000000000000000
lender1.balanceOf() 10000000000000000
```

Scenario 2 - with accrueInterest function calling

1. All necessary contracts are deployed and initialized: RevenueToken, SimpleOracle, LoanLib, LineOfCredit.
2. As borrower and lender1, add credit position for 10_000_000_000_000_000 tokens with facility rate set to 1.
3. Forward blockchain time for 10 days.
4. Call accrueInterest() function. Note that the interestAccrued() parameter value in first credit record is updated.
5. As the borrower, attempt to close() the credit. Observe that the transaction reverted with Loan: close failed. credit owed error.



```
lineOfCredit.credits(positionId1) (1000000000000000, 0, 0, 0, 18, '0x64B79c20d336896927cA1e2083fe6FD4f8Ae975A',
lineOfCredit.balanceOf() 1000000000000000
borrower.balanceOf() 10000000000000000
lender1.balanceOf() 9000000000000000
Calling -> chain.sleep(day * 20)
Calling -> chain.mine(1) )
lineOfCredit.credits(positionId1) (1000000000000000, 0, 0, 0, 18, '0x64B79c20d336896927cA1e2083fe6FD4f8Ae975A',
lineOfCredit.balanceOf() 1000000000000000
borrower.balanceOf() 10000000000000000
lender1.balanceOf() 9000000000000000
Calling -> accrueInterest = lineOfCredit.accrueInterest().return_value
Transaction sent: 0x9a09d2ec5402a5a050e1219e4508ad6797d4ec19bece93074628d5fe298824c0
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 744
  LineOfCredit.accrueInterest confirmed   Block: 15231536   Gas used: 58430 (0.02%)

accrueInterest = 1095
lineOfCredit.credits(positionId1) (1000000000000000, 0, 5475707911, 0, 18, '0x64B79c20d336896927cA1e2083fe6FD4f8
Calling -> lineOfCredit.close(positionId1, {'from': borrower})
Transaction sent: 0x6931a9e9b2796a4a8974923e16fc5ae13485ef62b8a15f77eae02df0eadb1221
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 421
  LineOfCredit.close confirmed (Loan: close failed. credit owed)   Block: 15231537   Gas used: 28194 (0.01%)

lineOfCredit.credits(positionId1) (1000000000000000, 0, 5475707911, 0, 18, '0x64B79c20d336896927cA1e2083fe6FD4f8
lineOfCredit.balanceOf() 1000000000000000
borrower.balanceOf() 10000000000000000
lender1.balanceOf() 9000000000000000
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

The close() function combined with depositAndRepay() should allow the user to close credit without accruing interest infinitely; however, then, the risk presented in the finding remains valid. The project team should reconsider the purpose and uncertain nature of the close

() function and possibly change the implementation while aligning with business requirements.

It is recommended to either force borrower to pay off accrued interest from unused funds up calling the close() function or remit it.

3.17 (HAL-17) CLOSE FUNCTION CAN BE FRONT-RUN BY LENDER - MEDIUM

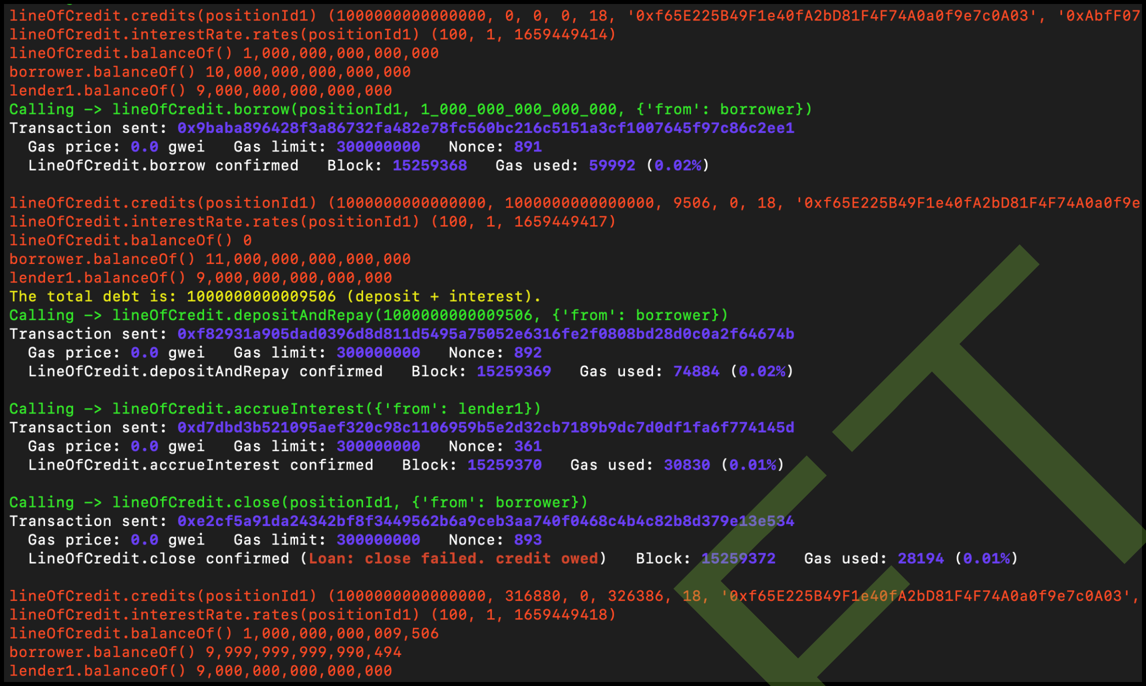
Description:

In the LineOfCredit contract, the borrower has two possibilities to pay off the debt: by calling depositAndRepay() and then close() functions, or by calling a single depositAndClose() function. The close() function combined with depositAndRepay() should allow the borrower to close the debit. The close(id) function does not call _accrueInterest() internal function.

The assessment revealed that the lender can front-run close() function called by the borrower with the accrueInterest() function. As a result, the close() function will revert, and a small amount of facility interest will be added to the debt. Eventually, the borrower could escape from the situation by using the depositAndClose() function.

Proof of Concept:

1. All necessary contracts are deployed and initialized: RevenueToken, SimpleOracle, LoanLib, LineOfCredit.
2. As borrower and lender, add credit position for 10_000_000_000_000_000 tokens with facility rate set to 1.
3. As borrower, borrow() all deposits.
4. As the borrower, pay off the debt and interest using the depositAndRepay() function.
5. As borrower, attempt to close() the credit, but firstly call accrueInterest() function as a lender to simulate front-running.
6. Observe that close() function reverts with Loan: close failed. credit owed error. Note that a small amount of facility rate is accrued.



```
lineOfCredit.credits(positionId1) (1000000000000000, 0, 0, 0, 18, '0xf65E225B49F1e40fA2bD81F4F74A0a0f9e7c0A03', '0xAbfF07
lineOfCredit.interestRate.rates(positionId1) (100, 1, 1659449414)
lineOfCredit.balanceOf() 1,000,000,000,000,000
borrower.balanceOf() 10,000,000,000,000,000
lender1.balanceOf() 9,000,000,000,000,000
Calling -> lineOfCredit.borrow(positionId1, 1_000_000_000_000_000, {'from': borrower})
Transaction sent: 0x9baba896428f3a86732fa482e78fc560bc216c5151a3cf1007645f97c86c2ee1
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 891
  LineOfCredit.borrow confirmed   Block: 15259368   Gas used: 59992 (0.02%)

lineOfCredit.credits(positionId1) (1000000000000000, 1000000000000000, 9506, 0, 18, '0xf65E225B49F1e40fA2bD81F4F74A0a0f9e
lineOfCredit.interestRate.rates(positionId1) (100, 1, 1659449417)
lineOfCredit.balanceOf() 0
borrower.balanceOf() 11,000,000,000,000,000
lender1.balanceOf() 9,000,000,000,000,000
The total debt is: 1000000000009506 (deposit + interest).
Calling -> lineOfCredit.depositAndRepay(1000000000009506, {'from': borrower})
Transaction sent: 0xf82931a905dad0396d8d811d5495a75052e6316fe2f0808bd28d0c0a2f64674b
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 892
  LineOfCredit.depositAndRepay confirmed   Block: 15259369   Gas used: 74884 (0.02%)

Calling -> lineOfCredit.accrueInterest({'from': lender1})
Transaction sent: 0xd7dbd3b521095aef320c98c1106959b5e2d32cb7189b9dc7d0df1fa6f774145d
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 361
  LineOfCredit.accrueInterest confirmed   Block: 15259370   Gas used: 30830 (0.01%)

Calling -> lineOfCredit.close(positionId1, {'from': borrower})
Transaction sent: 0xe2cf5a91da24342bf8f3449562b6a9ceb3aa740f0468c4b4c82b8d379e13e534
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 893
  LineOfCredit.close confirmed (Loan: close failed. credit owed)   Block: 15259372   Gas used: 28194 (0.01%)

lineOfCredit.credits(positionId1) (1000000000000000, 316880, 0, 326386, 18, '0xf65E225B49F1e40fA2bD81F4F74A0a0f9e7c0A03',
lineOfCredit.interestRate.rates(positionId1) (100, 1, 1659449418)
lineOfCredit.balanceOf() 1,000,000,000,009,506
borrower.balanceOf() 9,999,999,999,990,494
lender1.balanceOf() 9,000,000,000,000,000
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

It is recommended to either remove the close() function completely or change the function’s authorization, so only the lender can call it. Adjusting close() function to transfer additional funds by borrower is not an option, as it would duplicate the depositAndClose() functionality.

FINDINGS & TECH DETAILS

3.18 (HAL-18) UNUSED CREDIT TOKENS LOCKOUT UNTIL NEW REVENUE - MEDIUM

Description:

In the SpigotedLoan contract, the borrower can add spigots with revenue contracts that will support repayment of the debt. The revenue contract earns revenue tokens that later can be exchanged for the credit tokens using the claimAndRepay() and claimAndTrade() functions. These functions call the _claimAndTrade() internal function responsible for claiming escrowed tokens from the SpigotController contract and trading the claimed tokens. Also, the claimAndTrade() function updates the unusedTokens collection with target tokens (credit tokens) that were traded.

Listing 33: SpigotedLoan.sol (Line 152)

```
137     function claimAndTrade(address claimToken, bytes calldata
  ↳ zeroExTradeData)
138         external
139         whileBorrowing
140         returns (uint256 tokensBought)
141     {
142         require(msg.sender == borrower || msg.sender == arbiter);
143
144         address targetToken = credits[ids[0]].token;
145         tokensBought = _claimAndTrade(
146             claimToken,
147             targetToken,
148             zeroExTradeData
149         );
150
151         // add bought tokens to unused balance
152         unusedTokens[targetToken] += tokensBought;
153     }
```

In the SpigotController contract, the _claimRevenue() internal function is responsible for calculating the claimed token value. However, it reverts when no tokens can be claimed (line 140).

The assessment revealed that it is not possible to use traded credit tokens

to pay off the debt until new revenue is claimed in the SpigotController contract. The SpigotController contract gains tokens from the revenue contract, which is a third-party component; thus, the point of time of new revenue arrival is uncertain.
The borrower can also abuse this vulnerability to trade revenue tokens but not use them to pay off the debt, making the spigot mechanism ineffective.

Listing 34: Spigot.sol (Line 140)

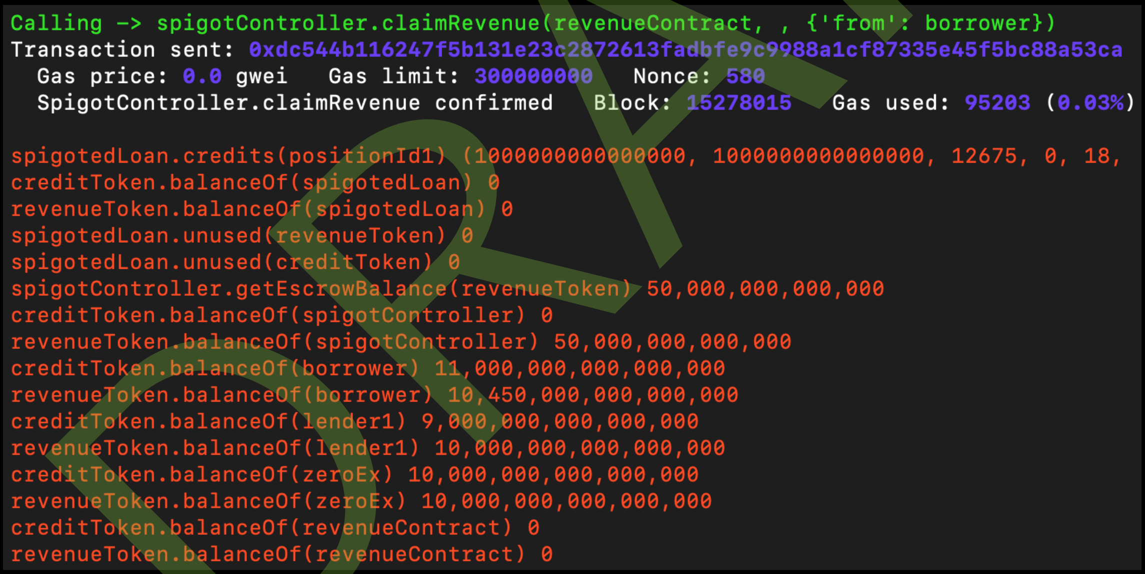
```
122 function _claimRevenue(address revenueContract, bytes calldata
 ↳ data, address token)
123     internal
124     returns (uint256 claimed)
125   {
126     uint256 existingBalance = _getBalance(token);
127     if(settings[revenueContract].claimFunction == bytes4(0)) {
128       // push payments
129       // claimed = total balance - already accounted for
 ↳ balance
130       claimed = existingBalance - escrowed[token];
131     } else {
132       // pull payments
133       require(bytes4(data) == settings[revenueContract].
 ↳ claimFunction, "Spigot: Invalid claim function");
134       (bool claimSuccess, bytes memory claimData) =
 ↳ revenueContract.call(data);
135       require(claimSuccess, "Spigot: Revenue claim failed");
136       // claimed = total balance - existing balance
137       claimed = _getBalance(token) - existingBalance;
138     }
139
140     require(claimed > 0, "Spigot: No revenue to claim");
141     if(claimed > MAX_REVENUE) claimed = MAX_REVENUE;
142
143     return claimed;
144   }
```

Proof of Concept:

1. All necessary contracts are deployed and initialized: CreditToken, RevenueToken, SimpleOracle, LoanLib, SpigotedLoan,

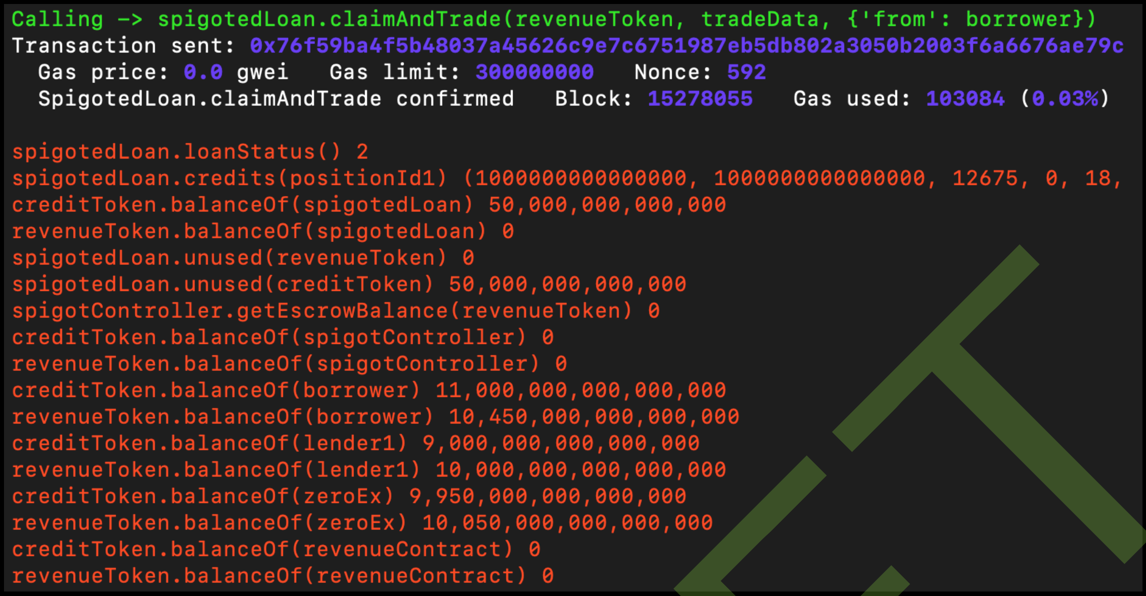
SimpleRevenueContract.

2. As borrower and lender1, add credit position for 10_000_000_000_000_000 tokens.
3. As borrower, borrow() all deposits.
4. As borrower and arbiter, add new spigot with addSpigot() function and RevenueContract as input. Set the ownerSplit parameter to 10.
5. As the borrower, transfer the ownership of the RevenueContract contract to the SpigotController.
6. Mint 500_000_000_000_000 revenue tokens to the RevenueContract contract to simulate token gain.
7. As the borrower, call claimRevenue() in SpigotController contract to claim revenue tokens. Note that 90% of tokens (450_000_000_000_000) are transferred to the treasury (which is the borrower), the 10% (50_000_000_000_000) of tokens are transferred to the SpigotController.



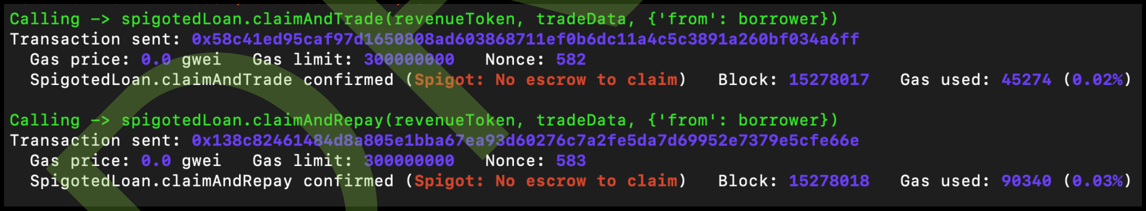
```
Calling -> spigotController.claimRevenue(revenueContract, , {'from': borrower})
Transaction sent: 0xdc544b116247f5b131e23c2872613fadbfe9c9988a1cf87335e45f5bc88a53ca
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 580
  SpigotController.claimRevenue confirmed   Block: 15278015   Gas used: 95203 (0.03%)

spigotedLoan.credits(positionId1) (1000000000000000, 1000000000000000, 12675, 0, 18,
creditToken.balanceOf(spigotedLoan) 0
revenueToken.balanceOf(spigotedLoan) 0
spigotedLoan.unused(revenueToken) 0
spigotedLoan.unused(creditToken) 0
spigotController.getEscrowBalance(revenueToken) 50,000,000,000,000
creditToken.balanceOf(spigotController) 0
revenueToken.balanceOf(spigotController) 50,000,000,000,000
creditToken.balanceOf(borrower) 11,000,000,000,000,000
revenueToken.balanceOf(borrower) 10,450,000,000,000,000
creditToken.balanceOf(lender1) 9,000,000,000,000,000
revenueToken.balanceOf(lender1) 10,000,000,000,000,000
creditToken.balanceOf(zeroEx) 10,000,000,000,000,000
revenueToken.balanceOf(zeroEx) 10,000,000,000,000,000
creditToken.balanceOf(revenueContract) 0
revenueToken.balanceOf(revenueContract) 0
```

6. As the borrower, call claimAndTrade() in the SpigotedLoan contract to trade escrowed revenue tokens. As an input, trade exchange data provide 50_000_000_000_000 revenue tokens that should be exchanged for credit tokens.



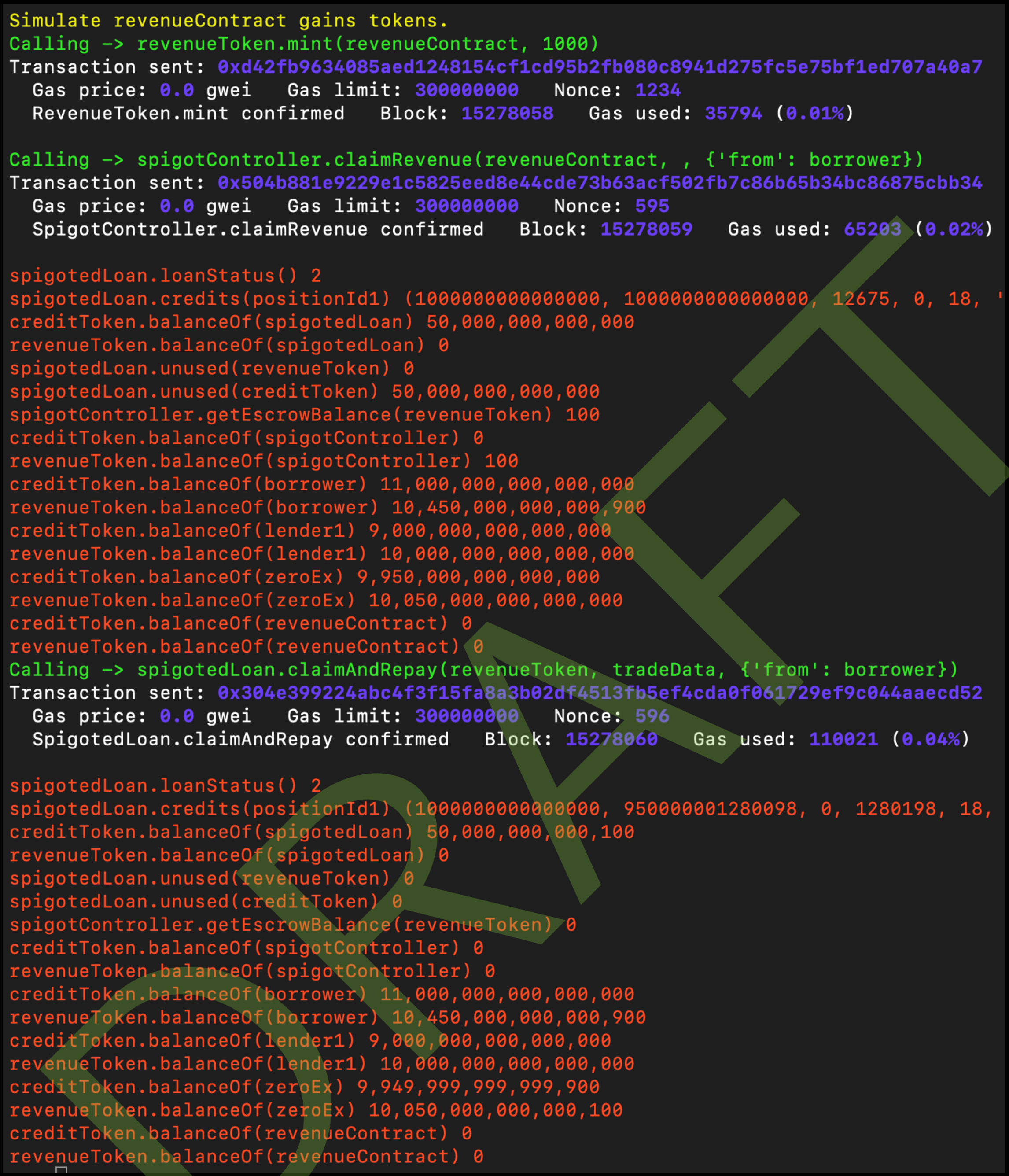
```
Calling -> spigotedLoan.claimAndTrade(revenueToken, tradeData, {'from': borrower})
Transaction sent: 0x76f59ba4f5b48037a45626c9e7c6751987eb5db802a3050b2003f6a6676ae79c
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 592
  SpigotedLoan.claimAndTrade confirmed   Block: 15278055   Gas used: 103084 (0.03%)

spigotedLoan.loanStatus() 2
spigotedLoan.credits(positionId1) (1000000000000000, 1000000000000000, 12675, 0, 18,
creditToken.balanceOf(spigotedLoan) 50,000,000,000,000
revenueToken.balanceOf(spigotedLoan) 0
spigotedLoan.unused(revenueToken) 0
spigotedLoan.unused(creditToken) 50,000,000,000,000
spigotController.getEscrowBalance(revenueToken) 0
creditToken.balanceOf(spigotController) 0
revenueToken.balanceOf(spigotController) 0
creditToken.balanceOf(borrower) 11,000,000,000,000,000
revenueToken.balanceOf(borrower) 10,450,000,000,000,000
creditToken.balanceOf(lender1) 9,000,000,000,000,000
revenueToken.balanceOf(lender1) 10,000,000,000,000,000
creditToken.balanceOf(zeroEx) 9,950,000,000,000,000
revenueToken.balanceOf(zeroEx) 10,050,000,000,000,000
creditToken.balanceOf(revenueContract) 0
revenueToken.balanceOf(revenueContract) 0
```

7. Observe the SpigotedLoan balances. Note that the contract has 50,000,000,000,000 credit tokens. Also 50,000,000,000,000 credit tokens are stored in unusedTokens collection.
8. As the borrower, attempt to call claimAndTrade() or claimAndRepay() functions. Observe that these functions revert with the error Spigot: No escrow to claim.



```
Calling -> spigotedLoan.claimAndTrade(revenueToken, tradeData, {'from': borrower})
Transaction sent: 0x58c41ed95caf97d1650808ad603868711ef0b6dc11a4c5c3891a260bf034a6ff
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 582
  SpigotedLoan.claimAndTrade confirmed (Spigot: No escrow to claim)   Block: 15278017   Gas used: 45274 (0.02%)

Calling -> spigotedLoan.claimAndRepay(revenueToken, tradeData, {'from': borrower})
Transaction sent: 0x138c82461484d8a805e1bba67ea93d60276c7a2fe5da7d69952e7379e5cfe66e
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 583
  SpigotedLoan.claimAndRepay confirmed (Spigot: No escrow to claim)   Block: 15278018   Gas used: 90340 (0.03%)
```

9. Mint 1000 revenue tokens to the RevenueContract contract to simulate token gain.
10. As the borrower, call claimRevenue() in the SpigotController contract to claim revenue tokens.
11. As the borrower, call claimAndRepay(). Observe that all unused tokens were used to pay off part of the debt.



```
Simulate revenueContract gains tokens.
Calling -> revenueToken.mint(revenueContract, 1000)
Transaction sent: 0xd42fb9634085aed1248154cf1cd95b2fb080c8941d275fc5e75bf1ed707a40a7
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 1234
  RevenueToken.mint confirmed   Block: 15278058   Gas used: 35794 (0.01%)

Calling -> spigotController.claimRevenue(revenueContract, , {'from': borrower})
Transaction sent: 0x504b881e9229e1c5825eed8e44cde73b63acf502fb7c86b65b34bc86875cbb34
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 595
  SpigotController.claimRevenue confirmed   Block: 15278059   Gas used: 65203 (0.02%)

spigotedLoan.loanStatus() 2
spigotedLoan.credits(positionId1) (1000000000000000, 1000000000000000, 12675, 0, 18, '
creditToken.balanceOf(spigotedLoan) 50,000,000,000,000
revenueToken.balanceOf(spigotedLoan) 0
spigotedLoan.unused(revenueToken) 0
spigotedLoan.unused(creditToken) 50,000,000,000,000
spigotController.getEscrowBalance(revenueToken) 100
creditToken.balanceOf(spigotController) 0
revenueToken.balanceOf(spigotController) 100
creditToken.balanceOf(borrower) 11,000,000,000,000,000
revenueToken.balanceOf(borrower) 10,450,000,000,000,900
creditToken.balanceOf(lender1) 9,000,000,000,000,000
revenueToken.balanceOf(lender1) 10,000,000,000,000,000
creditToken.balanceOf(zeroEx) 9,950,000,000,000,000
revenueToken.balanceOf(zeroEx) 10,050,000,000,000,000
creditToken.balanceOf(revenueContract) 0
revenueToken.balanceOf(revenueContract) 0
Calling -> spigotedLoan.claimAndRepay(revenueToken, tradeData, {'from': borrower})
Transaction sent: 0x304e399224abc4f3f15fa8a3b02df4513fb5ef4cda0f061729ef9c044aaecd52
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 596
  SpigotedLoan.claimAndRepay confirmed   Block: 15278060   Gas used: 110021 (0.04%)

spigotedLoan.loanStatus() 2
spigotedLoan.credits(positionId1) (1000000000000000, 950000001280098, 0, 1280198, 18,
creditToken.balanceOf(spigotedLoan) 50,000,000,000,100
revenueToken.balanceOf(spigotedLoan) 0
spigotedLoan.unused(revenueToken) 0
spigotedLoan.unused(creditToken) 0
spigotController.getEscrowBalance(revenueToken) 0
creditToken.balanceOf(spigotController) 0
revenueToken.balanceOf(spigotController) 0
creditToken.balanceOf(borrower) 11,000,000,000,000,000
revenueToken.balanceOf(borrower) 10,450,000,000,000,900
creditToken.balanceOf(lender1) 9,000,000,000,000,000
revenueToken.balanceOf(lender1) 10,000,000,000,000,000
creditToken.balanceOf(zeroEx) 9,949,999,999,999,900
revenueToken.balanceOf(zeroEx) 10,050,000,000,000,100
creditToken.balanceOf(revenueContract) 0
revenueToken.balanceOf(revenueContract) 0
```

Risk Level:

Likelihood - 4

Impact - 2

Recommendation:

It is recommended to allow users to use all traded credit tokens to pay off the debt, regardless of revenue to claim.

3.19 (HAL-19) BORROWER CAN CLAIM REVENUE WHILE LOAN IS LIQUIDATABLE - MEDIUM

Description:

In the SpigotController, the claimRevenue() function allows claiming revenue from the revenue contract, split it between treasury and escrow, and send part of it to the treasury. By default, the treasury is the borrower. The ownerSplit parameter is the initial split agreed upon and set up by the borrower and the arbiter.
When the credit deadline has passed, and it has defaulted, the healthcheck () function from the LineOfCredit contract must be called explicitly to set loan status to LIQUIDATABLE. Afterward, the arbiter can call the updateOwnerSplit() function from the SpigotedLoan contract to update the ownerSplit parameter, so 100% of revenue is escrowed, and none is sent to the treasury.

Listing 35: Spigot.sol (Line 113)

```
    /**

     * @notice - Claim push/pull payments through Spigots.
                   Calls predefined function in contract settings to claim revenue.
                   Automatically sends portion to treasury and escrows Owner's share.
     * @dev - callable by anyone
     * @param revenueContract Contract with registered settings to claim revenue from
     * @param data  Transaction data, including function signature, to properly claim revenue on revenueContract
     * @return claimed -  The amount of tokens claimed from revenueContract and split in payments to `owner` and `treasury`
    */
    function claimRevenue(address revenueContract, bytes calldata data)
        external nonReentrant
        returns (uint256 claimed)
    {
        address token = settings[revenueContract].token;
        claimed = _claimRevenue(revenueContract, data, token);

        // split revenue stream according to settings
        uint256 escrowedAmount = claimed * settings[revenueContract].ownerSplit / 100;
        // update escrowed balance
        escrowed[token] = escrowed[token] + escrowedAmount;

        // send non-escrowed tokens to Treasury if non-zero
        if(claimed > escrowedAmount) {
            require(_sendOutTokenOrETH(token, treasury, claimed - escrowedAmount));
        }

        emit ClaimRevenue(token, claimed, escrowedAmount, revenueContract);

        return claimed;
    }
```

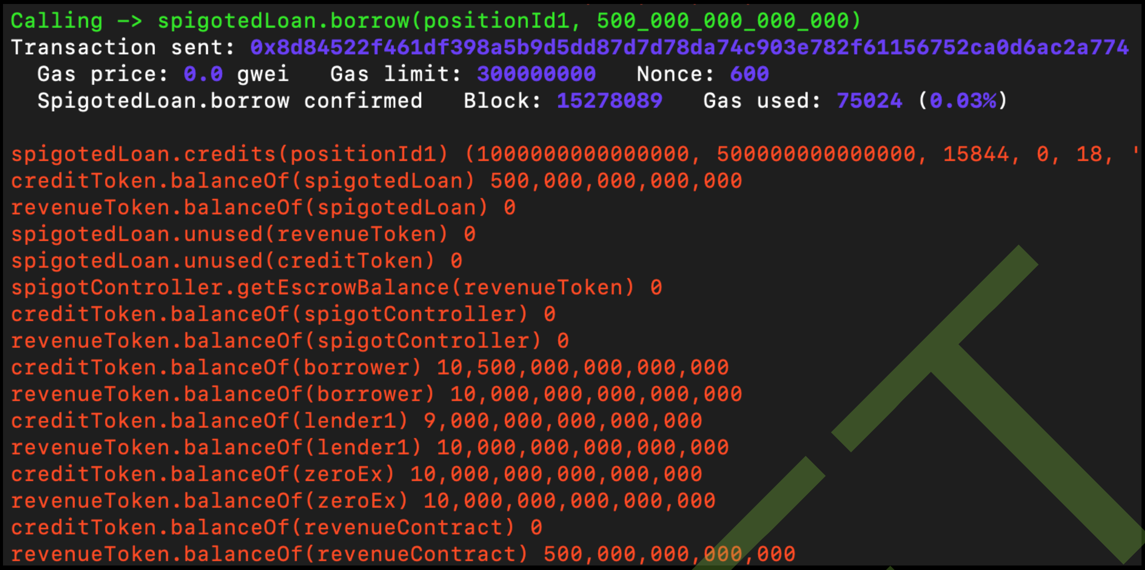
Listing 36: SpigotedLoan.sol (Lines 72,75)

```
    /**
     * @notice changes the revenue split between borrower treasury and lan repayment based on loan health
     * @dev    - callable `arbiter` + `borrower`
     * @param revenueContract - spigot to update
     */
    function updateOwnerSplit(address revenueContract) external returns (bool) {
        (, uint8 split, , bytes4 transferFunc) = spigot.getSetting(
            revenueContract
        );

        require(transferFunc != bytes4(0), "SpgtLoan: no spigot");

        if (
            loanStatus == LoanLib.STATUS.ACTIVE && split != defaultRevenueSplit
        ) {
            // if loan is healthy set split to default take rate
            spigot.updateOwnerSplit(revenueContract, defaultRevenueSplit);
        } else if (
            loanStatus == LoanLib.STATUS.LIQUIDATABLE && split != MAX_SPLIT
        ) {
            // if loan is in distress take all revenue to repay loan
            spigot.updateOwnerSplit(revenueContract, MAX_SPLIT);
        }

        return true;
    }
```

The assessment revealed that the loan's LIQUIDATABLE status is not propagated to the spigots. Also, claimRevenue() function has no authorization implemented. As a result, the borrower can front-run the updateOwnerSplit() function with the claimRevenue() to obtain one more revenue share from spigot.

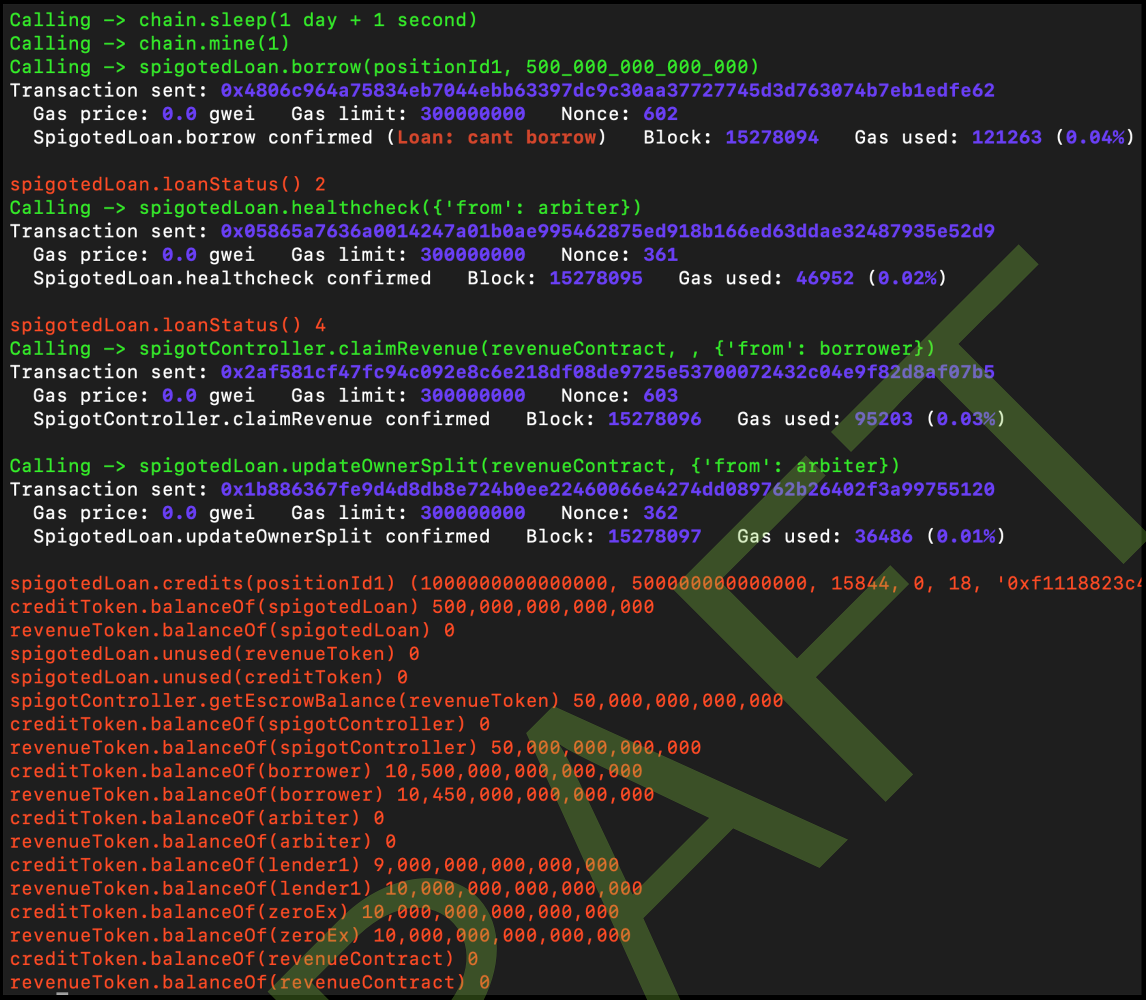
Proof of Concept:

1. All necessary contracts are deployed and initialized: CreditToken, RevenueToken, SimpleOracle, LoanLib, SpigotedLoan, SimpleRevenueContract. Set the ttl parameter to 1 day.
2. As borrower and lender1, add credit position for 10_000_000_000_-000_000 tokens.
3. As the borrower borrow() half of the deposit - 5_000_000_000_000_000.
4. As borrower and arbiter, add a new spigot with the addSpigot() function and Set the ownerSplit parameter to 10.
5. As the borrower, transfer the ownership of the RevenueContract contract to the SpigotController.
6. Mint 500_000_000_000_000 revenue tokens to the RevenueContract contract to simulate token gain.
7. Forward blockchain time for 1 day and 1 second.
8. As the borrower, attempt to borrow() the remaining deposit. Observe that the transaction reverted with the Loan: can't borrow error.



```
Calling -> spigotedLoan.borrow(positionId1, 500_000_000_000_000)
Transaction sent: 0x8d84522f461df398a5b9d5dd87d7d78da74c903e782f61156752ca0d6ac2a774
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 600
  SpigotedLoan.borrow confirmed   Block: 15278089   Gas used: 75024 (0.03%)

spigotedLoan.credits(positionId1) (1000000000000000, 500000000000000, 15844, 0, 18, '
creditToken.balanceOf(spigotedLoan) 500,000,000,000,000
revenueToken.balanceOf(spigotedLoan) 0
spigotedLoan.unused(revenueToken) 0
spigotedLoan.unused(creditToken) 0
spigotController.getEscrowBalance(revenueToken) 0
creditToken.balanceOf(spigotController) 0
revenueToken.balanceOf(spigotController) 0
creditToken.balanceOf(borrower) 10,500,000,000,000,000
revenueToken.balanceOf(borrower) 10,000,000,000,000,000
creditToken.balanceOf(lender1) 9,000,000,000,000,000
revenueToken.balanceOf(lender1) 10,000,000,000,000,000
creditToken.balanceOf(zeroEx) 10,000,000,000,000,000
revenueToken.balanceOf(zeroEx) 10,000,000,000,000,000
creditToken.balanceOf(revenueContract) 0
revenueToken.balanceOf(revenueContract) 500,000,000,000,000
```

7. As the arbiter, call the healthcheck() function. Note that the loan's status changed from ACTIVE to LIQUIDATABLE.
8. As the borrower, call claimRevenue() to simulate front-run of the updateOwnerSplit() function, so 90% of revenue will go to the borrower and 10% go to the SpigotController contract.



```
Calling -> chain.sleep(1 day + 1 second)
Calling -> chain.mine(1)
Calling -> spigotedLoan.borrow(positionId1, 500_000_000_000_000)
Transaction sent: 0x4806c964a75834eb7044ebb63397dc9c30aa37727745d3d763074b7eb1edfe62
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 602
  SpigotedLoan.borrow confirmed (Loan: cant borrow)   Block: 15278094   Gas used: 121263 (0.04%)

spigotedLoan.loanStatus() 2
Calling -> spigotedLoan.healthcheck({'from': arbiter})
Transaction sent: 0x05865a7636a0014247a01b0ae995462875ed918b166ed63ddae32487935e52d9
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 361
  SpigotedLoan.healthcheck confirmed   Block: 15278095   Gas used: 46952 (0.02%)

spigotedLoan.loanStatus() 4
Calling -> spigotController.claimRevenue(revenueContract, , {'from': borrower})
Transaction sent: 0x2af581cf47fc94c092e8c6e218df08de9725e53700072432c04e9f82d8af07b5
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 603
  SpigotController.claimRevenue confirmed   Block: 15278096   Gas used: 95203 (0.03%)

Calling -> spigotedLoan.updateOwnerSplit(revenueContract, {'from': arbiter})
Transaction sent: 0x1b886367fe9d4d8db8e724b0ee22460066e4274dd089762b26402f3a99755120
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 362
  SpigotedLoan.updateOwnerSplit confirmed   Block: 15278097   Gas used: 36486 (0.01%)

spigotedLoan.credits(positionId1) (1000000000000000, 500000000000000, 15844, 0, 18, '0xf1118823c
creditToken.balanceOf(spigotedLoan) 500,000,000,000,000
revenueToken.balanceOf(spigotedLoan) 0
spigotedLoan.unused(revenueToken) 0
spigotedLoan.unused(creditToken) 0
spigotController.getEscrowBalance(revenueToken) 50,000,000,000,000
creditToken.balanceOf(spigotController) 0
revenueToken.balanceOf(spigotController) 50,000,000,000,000
creditToken.balanceOf(borrower) 10,500,000,000,000,000
revenueToken.balanceOf(borrower) 10,450,000,000,000,000
creditToken.balanceOf(arbiter) 0
revenueToken.balanceOf(arbiter) 0
creditToken.balanceOf(lender1) 9,000,000,000,000,000
revenueToken.balanceOf(lender1) 10,000,000,000,000,000
creditToken.balanceOf(zeroEx) 10,000,000,000,000,000
revenueToken.balanceOf(zeroEx) 10,000,000,000,000,000
creditToken.balanceOf(revenueContract) 0
revenueToken.balanceOf(revenueContract) 0
```

Risk Level:

Likelihood - 4

Impact - 3

Recommendation:

It is recommended to implement safety measures that prevent claiming revenue by the borrower when credit is defaulted.

3.20 (HAL-20) MINIMUMCOLLATERALRATIO LACKS INPUT VALIDATION - MEDIUM

Description:

In the Escrow contract, the minimumCollateralRatio parameter defines the minimum threshold of collateral ratio that allows the borrower to borrow the deposit (using the _healthcheck() function). Basing on project's documentation this parameter is expected to have 18 decimals, e.g. 1 ether = 100%. However, the contract does not implement any input validation, thus prone to human errors. A user could set too small a value, e.g., 100, which would make the mechanism ineffective. No function to update the minimumCollateralRatio parameter is available in the contract.

Listing 37: Escrow.sol

```
    // the minimum value of the collateral in relation to the outstanding debt e.g. 10% of outstanding debt
    uint256 public minimumCollateralRatio;
```

Listing 38: Escrow.sol (Line 37)

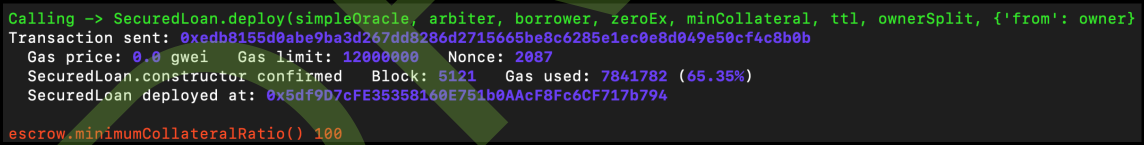
```
31    constructor(
32        uint256 _minimumCollateralRatio,
33        address _oracle,
34        address _loan,
35        address _borrower
36    ) public {
37        minimumCollateralRatio = _minimumCollateralRatio;
38        oracle = _oracle;
39        loan = _loan;
40        borrower = _borrower;
41    }
```

Listing 39: EscrowedLoan.sol (Line 26)

```
function _healthcheck() virtual internal returns(LoanLib.STATUS)
↳ {
    if(escrow.getCollateralRatio() < escrow.minimumCollateralRatio
↳ ()) {
      return LoanLib.STATUS.LIQUIDATABLE;
    }

    return LoanLib.STATUS.ACTIVE;
}
```

Proof of Concept:

1. All necessary contracts are deployed and initialized: CreditToken, RevenueToken, SimpleOracle, LoanLib, EscrowedLoan. For the minimumCollateralRatio parameter in the EscrowedLoan contract, provide small value, e.g. 100.
2. Observe that the EscrowedLoan contract deployed successfully.
3. As any account, call the minimumCollateralRatio() function. Observe the result, note that this confirms the contract's constructor appceted too low value for the minimumCollateralRatio parameter.



```
Calling -> SecuredLoan.deploy(simpleOracle, arbiter, borrower, zeroEx, minCollateral, ttl, ownerSplit, {'from': owner}
Transaction sent: 0xedb8155d0abe9ba3d267dd8286d2715665be8c6285e1ec0e8d049e50cf4c8b0b
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2087
  SecuredLoan.constructor confirmed   Block: 5121   Gas used: 7841782 (65.35%)
  SecuredLoan deployed at: 0x5df9D7cFE35358160E751b0AAcF8Fc6CF717b794

escrow.minimumCollateralRatio() 100
```

Risk Level:

Likelihood - 3
Impact - 4

Recommendation:

It is recommended to implement input validation for the minimumCollateralRatio parameter in the Escrow contract.

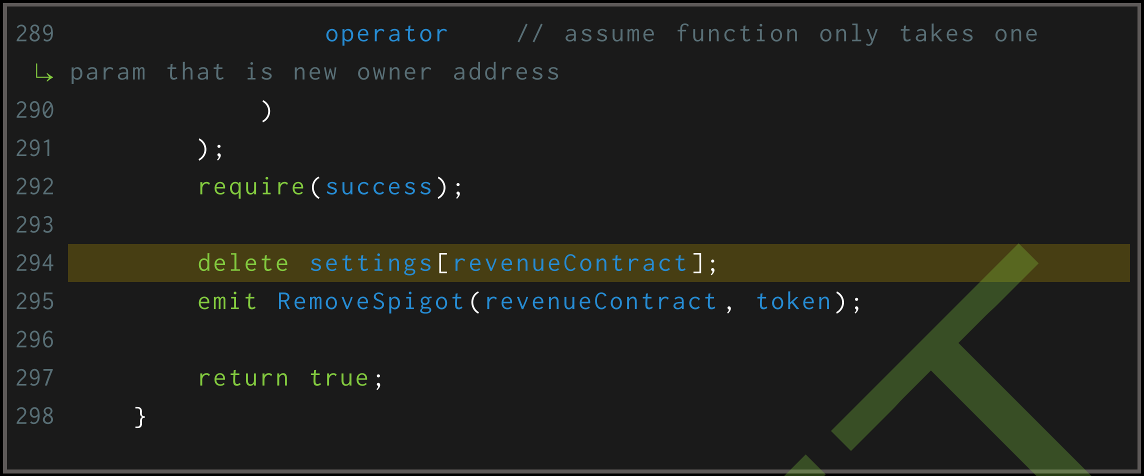
3.21 (HAL-21) REVENUE CONTRACT OWNERSHIP LOCKOUT POSSIBLE IN REMOVESPIGOT - MEDIUM

Description:

In the Spigot contract, the removeSpigot() function allows the contract's owner to transfer tokens held by the spigot, and transfer the ownership of the revenueContract to the operator (the borrower) and remove the revenueContract's data from spigot's settings collection. The amount of tokens to transfer is based on the escrowed collection. The Spigot allows multiple revenueContract contracts, and such contracts can provide revenue in the same revenue tokens. In such circumstances, the first call to the removeSpigot() function will transfer all escrowed tokens from every related revenueContract, but escrowed[token] will not be reset. Any subsequent call will revert, as a function would attempt to transfer escrowed tokens with an empty contract's balance. As a result, transferring the ownership of the remaining revenueContract will not be possible. However, the contract's owner could escape from this situation by transferring the tokens back to the contract, which is cumbersome.

Listing 40: Spigot.sol (Lines 279,280,294)

```
276     function removeSpigot(address revenueContract) external
  ↳ returns (bool) {
277         require(msg.sender == owner);
278
279         address token = settings[revenueContract].token;
280         uint256 claimable = escrowed[token];
281         if(claimable > 0) {
282             require(_sendOutTokenOrETH(token, owner, claimable));
283             emit ClaimEscrow(token, claimable, owner);
284         }
285
286         (bool success, bytes memory callData) = revenueContract.
  ↳ call(
287             abi.encodeWithSelector(
288                 settings[revenueContract].transferOwnerFunction,
```

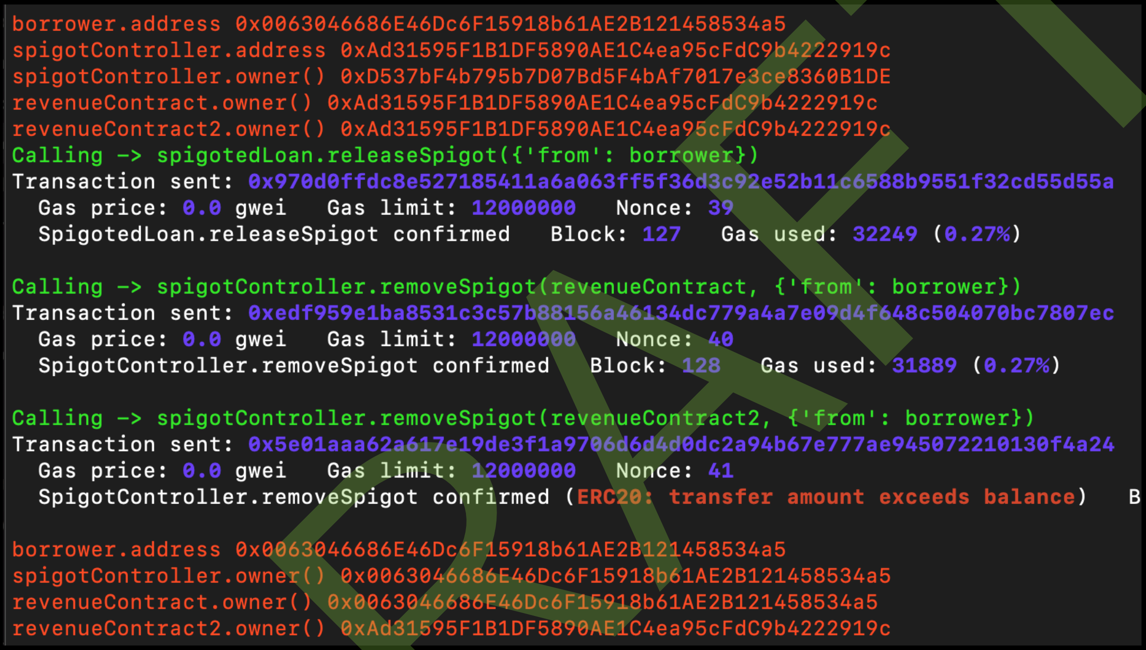


```
                operator     // assume function only takes one param that is new owner address
            )
        );
        require(success);

        delete settings[revenueContract];
        emit RemoveSpigot(revenueContract, token);

        return true;
    }
```

Proof of Concept:

1. All necessary contracts are deployed and initialized: CreditToken, RevenueToken, SimpleOracle, LoanLib, SpigotedLoan.
2. Deploy the first SimpleRevenueContract with RevenueToken.
3. Deploy the second SimpleRevenueContract with RevenueToken.
4. As borrower and lender, add credit position for 1_000_000_000_000_000 tokens.
5. As borrower and arbiter, add first spigot with addSpigot() function and the first SimpleRevenueContract as input.
6. As borrower and arbiter, add second spigot with addSpigot() function and the second SimpleRevenueContract as input.
7. As the borrower, transfer the ownership of the first SimpleRevenueContract contract to the SpigotController.
8. As the borrower, transfer the ownership of the second SimpleRevenueContract contract to the SpigotController.
9. Mint 10_000_000_000_000 revenue tokens to the first SimpleRevenueContract contract to simulate token gain.
10. Mint 10_000_000_000_000 revenue tokens to the second SimpleRevenueContract contract to simulate token gain.
11. As the borrower, borrow() all deposits.
12. As borrower, claim revenue for the first SimpleRevenueContract contract in SpigotController.
13. As borrower, claim revenue for the second SimpleRevenueContract contract in SpigotController.
14. As the borrower, pay off the debt.

15. As a borrower, release the spigot in the SpigotedLoan controller.
16. As borrower, call removeSpigot() function with the first SimpleRevenueContract contract as an input.
17. As borrower, attempt to call removeSpigot() function with the second SimpleRevenueContract contract as an input. Observe that the transaction reverts with the ERC20: transfer amount exceeds balance error. Note that the contract's ownership still belongs to the SpigotController.



```
borrower.address 0x0063046686E46Dc6F15918b61AE2B121458534a5
spigotController.address 0xAd31595F1B1DF5890AE1C4ea95cFdC9b4222919c
spigotController.owner() 0xD537bF4b795b7D07Bd5F4bAf7017e3ce8360B1DE
revenueContract.owner() 0xAd31595F1B1DF5890AE1C4ea95cFdC9b4222919c
revenueContract2.owner() 0xAd31595F1B1DF5890AE1C4ea95cFdC9b4222919c
Calling -> spigotedLoan.releaseSpigot({'from': borrower})
Transaction sent: 0x970d0ffdc8e527185411a6a063ff5f36d3c92e52b11c6588b9551f32cd55d55a
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 39
  SpigotedLoan.releaseSpigot confirmed   Block: 127   Gas used: 32249 (0.27%)

Calling -> spigotController.removeSpigot(revenueContract, {'from': borrower})
Transaction sent: 0xedf959e1ba8531c3c57b88156a46134dc779a4a7e09d4f648c504070bc7807ec
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 40
  SpigotController.removeSpigot confirmed   Block: 128   Gas used: 31889 (0.27%)

Calling -> spigotController.removeSpigot(revenueContract2, {'from': borrower})
Transaction sent: 0x5e01aaa62a617e19de3f1a9706d6d4d0dc2a94b67e777ae945072210130f4a24
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 41
  SpigotController.removeSpigot confirmed (ERC20: transfer amount exceeds balance)   B

borrower.address 0x0063046686E46Dc6F15918b61AE2B121458534a5
spigotController.owner() 0x0063046686E46Dc6F15918b61AE2B121458534a5
revenueContract.owner() 0x0063046686E46Dc6F15918b61AE2B121458534a5
revenueContract2.owner() 0xAd31595F1B1DF5890AE1C4ea95cFdC9b4222919c
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

It is recommended to reset the escrowed[token] collection in the removeSpigot() function.

3.22 (HAL-22) MALICIOUS ARBITER CAN ALLOW OWNERSHIP TRANSFER FUNCTION TO OPERATOR - LOW

Description:

In the SpigotedLoan the solution assumes that the borrower adds a spigot with the revenue contract and transfers the ownership of the revenue contract to the SpigotController.

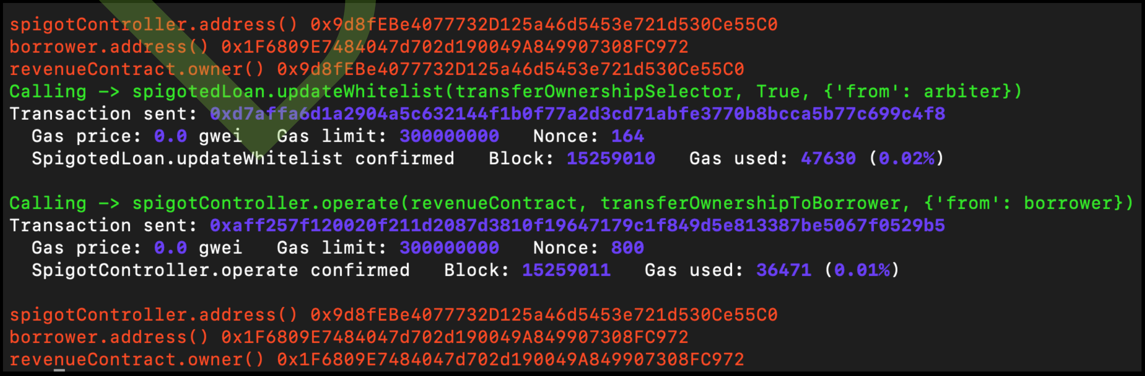
In the SpigotedLoan contract, the arbiter can call the updateWhitelist function to allow or disallow execution of the particular function for the operator (and the operator is the borrower by default) in the context of the revenue contract.
The assessment revealed that a malicious arbiter could whitelist the ownership transfer function for the operator. Thus, the operator can transfer the ownership from SpigotController back to the borrower using the _operate() function. This functions disallows to execute claimFunction(), however it does not disallow to execute transferOwnerFunction.

Listing 41: Spigot.sol

```
    /**
     * @notice - Checks that operation is whitelisted by Spigot Owner and calls revenue contract with supplied data
     * @param revenueContract - smart contracts to call
     * @param data - tx data, including function signature, to call contracts with
     */
    function _operate(address revenueContract, bytes calldata data) internal nonReentrant returns (bool) {
        // extract function signature from tx data and check whitelist
        require(whitelistedFunctions[bytes4(data)], "Spigot: Unauthorized action");
        // cant claim revenue via operate() because that fucks up accounting logic. Owner shouldn't whitelist it anyway but just in case
        require(settings[revenueContract].claimFunction != bytes4(data), "Spigot: Unauthorized action");


        (bool success, bytes memory opData) = revenueContract.call(data);
        require(success, "Spigot: Operation failed");

        return true;
    }
```

Proof of Concept:

1. All necessary contracts are deployed and initialized: CreditToken, RevenueToken, SimpleOracle, LoanLib, SpigotedLoan, SimpleRevenueContract.
2. As borrower and arbiter, add a new spigot with the addSpigot() function and RevenueContract as input.
3. As the borrower, transfer the ownership of the RevenueContract contract to the SpigotController.
4. As arbiter whitelist transferOwnership() function from RevenueContract contract using updateWhitelist() function and the transferOwnership() selector as an input.
5. As the borrower, transfer the ownership again using the operate() function. Note that the encoded transferOwnership() function selector and borrower's address must be provided as input.



```
spigotController.address() 0x9d8fEBe4077732D125a46d5453e721d530Ce55C0
borrower.address() 0x1F6809E7484047d702d190049A849907308FC972
revenueContract.owner() 0x9d8fEBe4077732D125a46d5453e721d530Ce55C0
Calling -> spigotedLoan.updateWhitelist(transferOwnershipSelector, True, {'from': arbiter})
Transaction sent: 0xd7affa6d1a2904a5c632144f1b0f77a2d3cd71abfe3770b8bcca5b77c699c4f8
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 164
  SpigotedLoan.updateWhitelist confirmed   Block: 15259010   Gas used: 47630 (0.02%)

Calling -> spigotController.operate(revenueContract, transferOwnershipToBorrower, {'from': borrower})
Transaction sent: 0xaff257f120020f211d2087d3810f19647179c1f849d5e813387be5067f0529b5
  Gas price: 0.0 gwei   Gas limit: 300000000   Nonce: 800
  SpigotController.operate confirmed   Block: 15259011   Gas used: 36471 (0.01%)

spigotController.address() 0x9d8fEBe4077732D125a46d5453e721d530Ce55C0
borrower.address() 0x1F6809E7484047d702d190049A849907308FC972
revenueContract.owner() 0x1F6809E7484047d702d190049A849907308FC972
```

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

It is recommended to disallow execution of the transferOwnerFunction() function, as it is done for claimFunction().

3.23 (HAL-23) UPDATEWHITELISTFUNCTION EVENT IS ALWAYS EMITTED WITH TRUE VALUE - LOW

Description:

In the SpigotController contract, the contract’s owner can call the _updateWhitelist function to allow or disallow execution of the particular function for the operator in the context of the revenue contract. Upon function execution, the UpdateWhitelistFunction event is emitted with true value, despite the actual value present in the allowed parameter. As a result, the contract may emit false and misleading information when some function is disallowed.

Listing 42: Spigot.sol (Lines 370,374)

```
366     /**
367
368      * @notice - Allows Owner to whitelist function methods across
 ↳  all revenue contracts for Operator to call.
369      * @param func - smart contract function signature to
 ↳ whitelist
370      * @param allowed - true/false whether to allow this function
 ↳ to be called by Operator
371      */
372     function _updateWhitelist(bytes4 func, bool allowed) internal
 ↳ returns (bool) {
373         whitelistedFunctions[func] = allowed;
374         emit UpdateWhitelistFunction(func, true);
375         return true;
376     }
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

It is recommended to emit the event with the allowed parameter as an input instead of fixed true value.

3.24 (HAL-24) BORROWER CAN MINIMIZE DRAWN INTEREST ACCRUING - LOW

Description:

The borrower can postpone borrowing the deposit (borrow() function), which would start drawing interest accruing as far as the credit would be required. When the lender attempt to withdraw() the deposit, the borrower could front-run it with the borrow() function, and immediately call the repay() function to pay off the debt, so no drawn interest would be applied, and all deposit still would be available. The borrower could repeat that until the credit deadline has passed, or the deposit would be required. During this time, the facility interest would still be accrued.

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended to apply cooldown periods between borrower’s function calls.

3.25 (HAL-25) REMOVESPIGOT DOES NOT CHECK CONTRACT'S BALANCE - LOW

Description:

In the Spigot contract, the removeSpigot() allows the contract's owner to transfer tokens held by the spigot, and transfer the ownership of the revenueContract to the operator (the borrower) and remove the revenueContract's data from the spigot's settings collection.

The amount of tokens to transfer is based on the escrowed collection. However, the contract's balance and the escrowed collection may contain different values. The contract's balance may be affected by push payments or by the MAX_REVENUE check in the _claimRevenue() function. The contract's balance may have a higher value than recorded in the escrowed collection. Thus, spigot removal may end up with an amount of tokens locked in the contract.
Adding a new spigot may be required to unlock the tokens, which is cumbersome.

Listing 43: Spigot.sol (Lines 280,282)

```
276     function removeSpigot(address revenueContract) external
  ↳ returns (bool) {
277         require(msg.sender == owner);
278
279         address token = settings[revenueContract].token;
280         uint256 claimable = escrowed[token];
281         if(claimable > 0) {
282             require(_sendOutTokenOrETH(token, owner, claimable));
283             emit ClaimEscrow(token, claimable, owner);
284         }
285
286         (bool success, bytes memory callData) = revenueContract.
  ↳ call(
287             abi.encodeWithSelector(
288                 settings[revenueContract].transferOwnerFunction,
289                 operator    // assume function only takes one
  ↳ param that is new owner address
290             )
```

```
        );
        require(success);

        delete settings[revenueContract];
        emit RemoveSpigot(revenueContract, token);

        return true;
    }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended to verify if value in the escrowed collection is equal to the contract’s balance and revert otherwise.

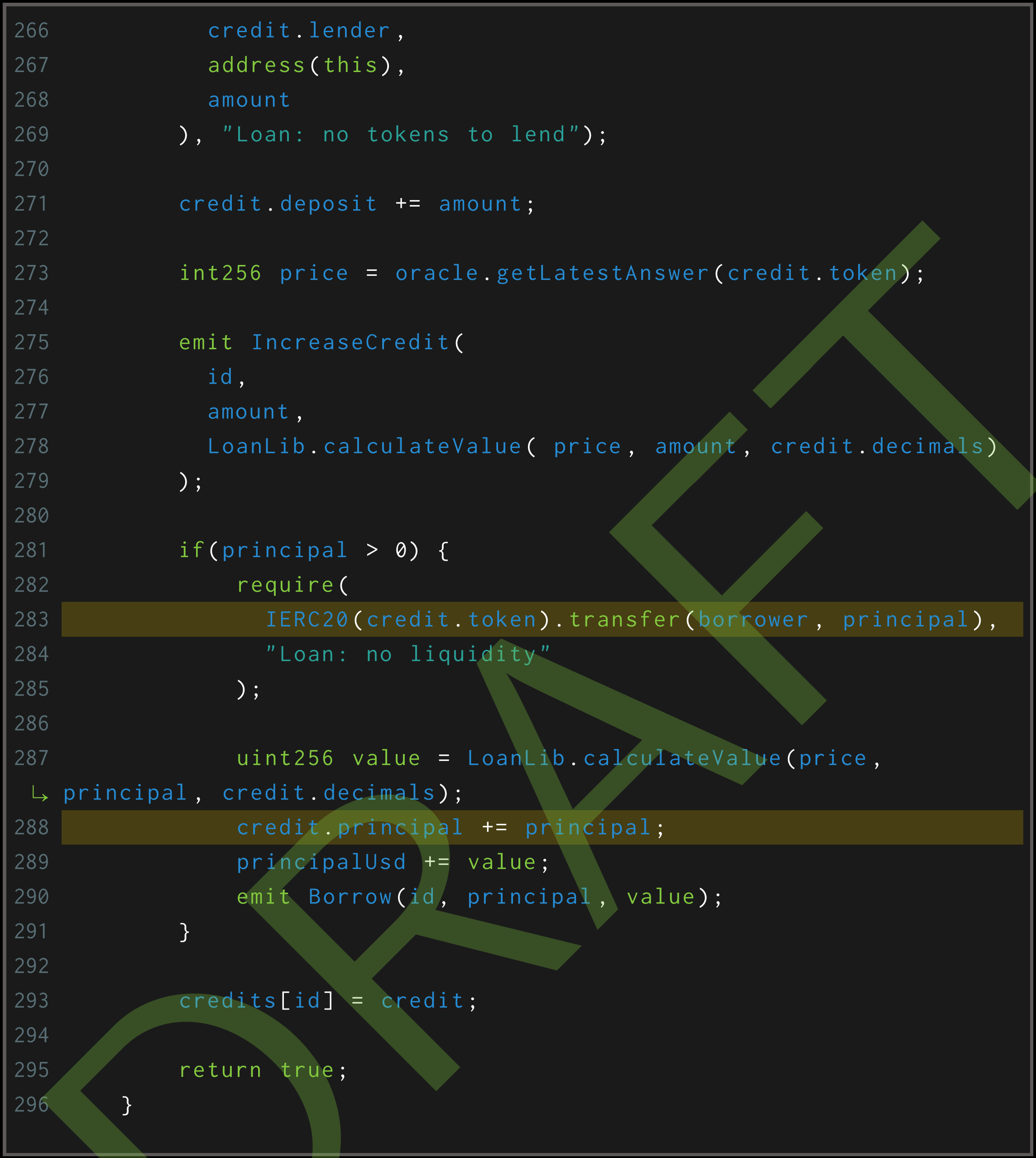
3.26 (HAL-26) INCREASECREDIT FUNCTION LACKS CALL TO SORTINTOQ - LOW

Description:

In the LineOfCredit contract, the increaseCredit() allows borrowers and lenders to increase their credit. This function also allows transferring specified principal to the borrower immediately. However, this function does not call the _sortIntoQ() function, which updates the credit position in the repaid queue. As a result, the credit position with increased credit and the transferred principal may not be updated in the queue and left behind other credit positions with the repaid principal. In some rare scenarios, to update the queue, the borrower would be forced to close other credit positions or to borrow from such a position (if the deposit is available).

Listing 44: LineOfCredit.sol (Lines 283,288)

```
function increaseCredit(
        bytes32 id,
        address lender,
        uint256 amount,
        uint256 principal
    )
      external
      override
      whileActive
      mutualConsent(lender, borrower)
      returns (bool)
    {
        _accrueInterest(id);
        require(principal <= amount, 'LoC: amount must be over princpal');
        Credit memory credit = credits[id];
        require(lender == credit.lender, 'LoC: only lender can increase');

        require(IERC20(credit.token).transferFrom(
```



```
          credit.lender,
          address(this),
          amount
        ), "Loan: no tokens to lend");

        credit.deposit += amount;

        int256 price = oracle.getLatestAnswer(credit.token);

        emit IncreaseCredit(
          id,
          amount,
          LoanLib.calculateValue( price, amount, credit.decimals)
        );

        if(principal > 0) {
            require(
              IERC20(credit.token).transfer(borrower, principal),
              "Loan: no liquidity"
            );

            uint256 value = LoanLib.calculateValue(price, principal, credit.decimals);
            credit.principal += principal;
            principalUsd += value;
            emit Borrow(id, principal, value);
        }

        credits[id] = credit;

        return true;
    }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to call the _sortIntoQ() function in the increaseCredit() function when principal is not 0.

3.27 (HAL-27) GAS OVER-CONSUMPTION IN LOOPS - INFORMATIONAL

Description:

In all the loops, the counter variable is incremented using i++. It is known that, in loops, using ++i costs less gas per iteration than i++.

Code Location:

LineOfCredit.sol
- Line 94: for (uint256 i = 0; i < len; i++)
- Line 131: for (uint256 i = 0; i < len; i++)
- Line 159: for (uint256 i = 0; i < len; i++)
- Line 686: for (uint256 i = 0; i < len; i++)

Escrow.sol
- Line 88: for (uint256 i = 0; i < length; i++)

Spigot.sol
- Line 206: for(uint256 i = 0; i < data.length; i++)

LoanLib.sol
- Line 122: for(uint i = 0; i < positions.length; i++)
- Line 151: for(uint i = 1; i < len; i++)

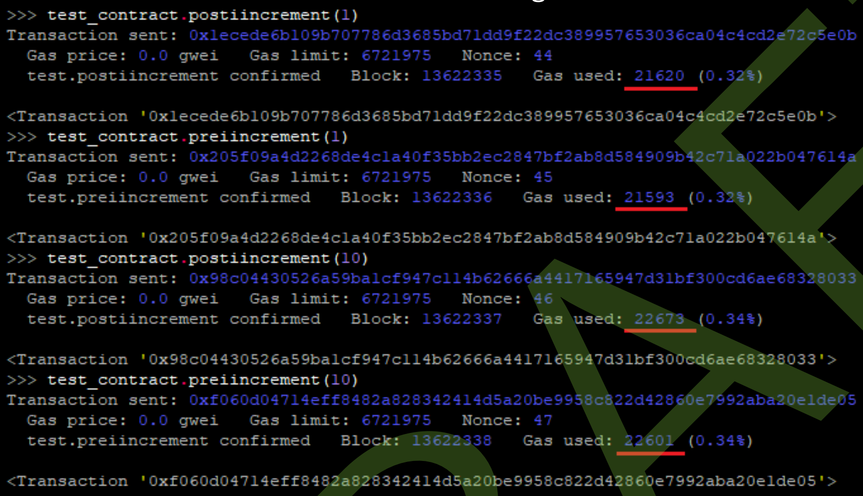
Proof of Concept:

For example, based in the following test contract:

Listing 45: Test.sol

```
//SPDX-License-Identifier: MIT
pragma solidity 0.8.9;

contract test {
    function postiincrement(uint256 iterations) public {
        for (uint256 i = 0; i < iterations; i++) {
        }
    }
    function preiincrement(uint256 iterations) public {
        for (uint256 i = 0; i < iterations; ++i) {
        }
    }
}
```

We can see the difference in the gas costs:



```
>>> test_contract.postiincrement(1)
Transaction sent: 0x1ecede6b109b707786d3685bd71dd9f22dc389957653036ca04c4cd2e72c5e0b
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 44
  test.postiincrement confirmed   Block: 13622335   Gas used: 21620 (0.32%)

<Transaction '0x1ecede6b109b707786d3685bd71dd9f22dc389957653036ca04c4cd2e72c5e0b'>
>>> test_contract.preiincrement(1)
Transaction sent: 0x205f09a4d2268de4c1a40f35bb2ec2847bf2ab8d584909b42c71a022b047614a
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 45
  test.preiincrement confirmed   Block: 13622336   Gas used: 21593 (0.32%)

<Transaction '0x205f09a4d2268de4c1a40f35bb2ec2847bf2ab8d584909b42c71a022b047614a'>
>>> test_contract.postiincrement(10)
Transaction sent: 0x98c04430526a59ba1cf947c114b62666a4417165947d31bf300cd6ae68328033
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 46
  test.postiincrement confirmed   Block: 13622337   Gas used: 22673 (0.34%)

<Transaction '0x98c04430526a59ba1cf947c114b62666a4417165947d31bf300cd6ae68328033'>
>>> test_contract.preiincrement(10)
Transaction sent: 0xf060d04714eff8482a828342414d5a20be9958c822d42860e7992aba20e1de05
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 47
  test.preiincrement confirmed   Block: 13622338   Gas used: 22601 (0.34%)

<Transaction '0xf060d04714eff8482a828342414d5a20be9958c822d42860e7992aba20e1de05'>
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to use ++i instead of i++ to increment the value of an uint variable inside a loop to save some gas. This is not applicable outside of loops.

3.28 (HAL-28) UNNEEDED INITIALIZATION OF UINT256 VARIABLES TO 0 - INFORMATIONAL

Description:

As i is an uint256, it is already initialized to 0. uint256 i = 0 reassigns the 0 to i which wastes gas.

Code Location:

Code Location:

LineOfCredit.sol

- Line 94: for (uint256 i = 0; i < len; i++)
- Line 131: for (uint256 i = 0; i < len; i++)
- Line 159: for (uint256 i = 0; i < len; i++)
- Line 684: uint256 _i = 0; // index that p should be moved to
- Line 686: for (uint256 i = 0; i < len; i++)

Escrow.sol

- Line 83: uint256 collateralValue = 0;
- Line 88: for (uint256 i = 0; i < length; i++)

Spigot.sol

- Line 206: for(uint256 i = 0; i < data.length; i++)

LoanLib.sol

- Line 119: uint256 count = 0;
- Line 122: for(uint i = 0; i < positions.length; i++)
- Line 151: for(uint i = 1; i < len; i++)

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to not initialize uint256 variables to 0 to save some gas. For example, use instead:

`for (uint256 i; i < proposal.targets.length; ++i)`.

3.29 (HAL-29) ASSERTIONS LACK MESSAGES - INFORMATIONAL

Description:

Several instances of assertions without messages were identified. The lack of message in require assertion might be unfavorable for end users.

Code Location:

LineOfCredit.sol

- Line 69: require(ids.length > 0 && credits[ids[0]].principal > 0);
- Line 199: require(interestRate.setRate(id, drate, frate));
- Line 228: require(interestRate.setRate(id, drate, frate));
- Line 344: require(amount <= credits[id].principal + credits[id].interestAccrued)
- Line 401: require(_sortIntoQ(id));
- Line 491: require(_close(id));

SecuredLoan.sol

- Line 53: require(msg.sender == arbiter);

SpigotedLoan.sol

- Line 97: require(msg.sender == borrower || msg.sender == arbiter);
- Line 142: require(msg.sender == borrower || msg.sender == arbiter);
- Line 229: require(msg.sender == arbiter);

Escrow.sol

- Line 144: require(msg.sender == ILoan(loan).arbiter());

Spigot.sol

- Line 154: require(msg.sender == owner);
- Line 193: require(msg.sender == operator);
- Line 205: require(msg.sender == operator);
- Line 243: require(msg.sender == operator);
- Line 256: require(revenueContract != address(this));

- Line 277: `require(msg.sender == owner);`
- Line 292: `require(success);`
- Line 277: `require(msg.sender == owner);`
- Line 315: `require(msg.sender == owner);`
- Line 330: `require(msg.sender == operator);`
- Line 345: `require(msg.sender == treasury || msg.sender == operator);`
- Line 330: `require(msg.sender == operator);`
- Line 362: `require(msg.sender == owner);`

EscrowedLoan.sol
- Line 55: `require(escrow.liquidate(amount, targetToken, to));`

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to add a meaningful message to each assertion.

3.30 (HAL-30) DEFAULTREVENUESPLIT LACKS INPUT VALIDATION - INFORMATIONAL

Description:

The defaultRevenueSplit parameter lacks input validation in the SpigotedLoan contract.

Code Location:

Listing 46: SpigotedLoan.sol (Line 45)

```
35 constructor(
36         address oracle_,
37         address arbiter_,
38         address borrower_,
39         address swapTarget_,
40         uint256 ttl_,
41         uint8 defaultRevenueSplit_
42     ) LineOfCredit(oracle_, arbiter_, borrower_, ttl_) {
43         spigot = new SpigotController(address(this), borrower,
   borrower);
44
45         defaultRevenueSplit = defaultRevenueSplit_;
46
47         swapTarget = swapTarget_;
48     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to add input validation for defaultRevenueSplit parameter.

3.31 (HAL-31) UNUSED CODE - INFORMATIONAL

Description:

Within the LoanLib contract, part of the code seems unused and redundant: calculateValue() function, DEBT_TOKEN constant, and some values from STATUS enum: UNINITIALIZED, INITIALIZED, UNDERCOLLATERALIZED, DELINQUENT, LIQUIDATING, OVERDRAWN, DEFAULT, ARBITRATION, INSOLVENT. Note that INSOLVENT is being used, but it is never set.

Within the EscrowedLoan contract, the _liquidate() function has positionId input parameter. This variable is used only to emit the Liquidate event from the IEscrowedLoan interface. Apart from that, no processing related to the credit with positionId is being done. Also, the liquidate() function from Escrow contract emits another Liquidate event from IEscrow interface. The positionId parameter and emission of the Liquidate event from the IEscrowedLoan interface seem to be redundant.

Code Location:

Listing 47: LoanLib.sol

```
address constant DEBT_TOKEN = address(0xdebf);

enum STATUS {
        // £hoo dis
        // Loan has been deployed but terms and conditions are still being signed off by parties
        UNINITIALIZED,
        INITIALIZED,

        // ITS ALLLIIIIVVEEE
        // Loan is operational and actively monitoring status
        ACTIVE,
        UNDERCOLLATERALIZED,
        LIQUIDATABLE, // [#X
        DELINQUENT,

        // Loan is in distress and paused
        LIQUIDATING,
        OVERDRAWN,
        DEFAULT,
        ARBITRATION,

        // Lön izz ded
        // Loan is no longer active, successfully repaid or insolvent
        REPAID,
        INSOLVENT
    }
```

Listing 48: LoanLib.sol

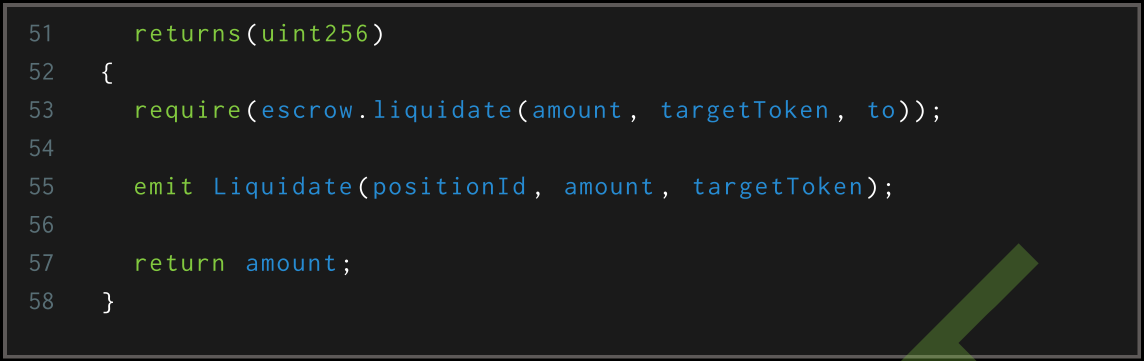
```
    function calculateValue(
      int price,
      uint256 amount,
      uint8 decimals
    )
      internal
      returns(uint256)
    {
      return _calculateValue(price, amount, decimals);
    }
```

Listing 49: IEscrowedLoan.sol

```
event Liquidate(bytes32 indexed positionId, uint256 indexed amount, address indexed token);
```

Listing 50: EscrowedLoan.sol

```
  function _liquidate(
    bytes32 positionId,
    uint256 amount,
    address targetToken,
    address to
  )
    virtual internal
```



```
    returns(uint256)
  {
    require(escrow.liquidate(amount, targetToken, to));

    emit Liquidate(positionId, amount, targetToken);

    return amount;
  }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to remove the redundant functionality to save some gas.

3.32 (HAL-32) LACK OF CHECK EFFECTS INTERACTIONS PATTERN OR REENTRENCY GUARD - INFORMATIONAL

Description:

The SpigotController contract inherits ReentrancyGuard. The functions _operate(), claimRevenue(), and claimEscrow() are protected by the nonReentrant modifier. The functions claimRevenue() and claimEscrow() uses _sendOutTokenOrETH() function to transfer the ether. However, the removeSpigot() also uses _sendOutTokenOrETH() function, but it is not protected by the nonReentrant modifier, also it modifies the contract's state after transferring the ether.

Listing 51: Spigot.sol (Lines 279,280,282,294)

```
276 function removeSpigot(address revenueContract) external returns (bool) {
277         require(msg.sender == owner);
278 
279         address token = settings[revenueContract].token;
280         uint256 claimable = escrowed[token];
281         if(claimable > 0) {
282             require(_sendOutTokenOrETH(token, owner, claimable));
283             emit ClaimEscrow(token, claimable, owner);
284         }
285 
286         (bool success, bytes memory callData) = revenueContract.call(
287             abi.encodeWithSelector(
288                 settings[revenueContract].transferOwnerFunction,
289                 operator    // assume function only takes one param that is new owner address
290             )
291         );
292         require(success);
293 
294         delete settings[revenueContract];
295         emit RemoveSpigot(revenueContract, token);
296 
297         return true;
298     }
```

In the SpigotedLoan the sweep() function transfers ether, however, it is not protected by the nonReentrant modifier, also it modifies the contract's state after transferring the ether.

Listing 52: SpigotedLoan.sol (Lines 278,280,284)

```
261     * @notice - sends unused tokens to borrower if repaid or
 ↳ arbiter if liquidatable
262              -  doesnt send tokens out if loan is unpaid but
 ↳ healthy
263     * @dev     - callable by anyone
264     * @param token - token to take out
265    */
266     function sweep(address token) external returns (uint256) {
267         if (loanStatus == LoanLib.STATUS.REPAID) {
268             return _sweep(borrower, token);
269         }
270         if (loanStatus == LoanLib.STATUS.INSOLVENT) {
271             return _sweep(arbiter, token);
272         }
273
274         return 0;
275     }
276
277     function _sweep(address to, address token) internal returns (
 ↳ uint256 x) {
278         x = unusedTokens[token];
279         if (token == address(0)) {
280             payable(to).transfer(x);
281         } else {
282             require(IERC20(token).transfer(to, x));
283         }
284         delete unusedTokens[token];
285     }
```

Risk Level:

Likelihood - 1

Impact - 1

Recommendation:

It is recommended to apply the nonReentrant modifier or introduce check effects interaction pattern for functions mentioned above as a part of defense in depth security strategy.

AUTOMATED TESTING

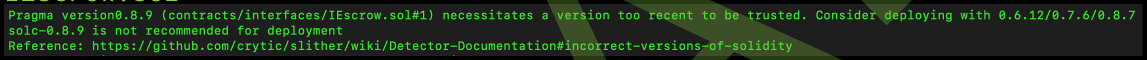
4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the scoped contracts. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their abi and binary formats, Slither was run on the all-scoped contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

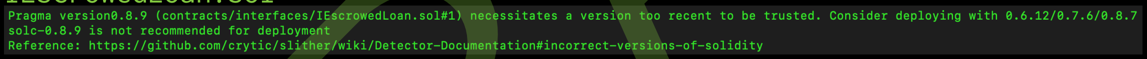
Slither results:

IEscrow.sol



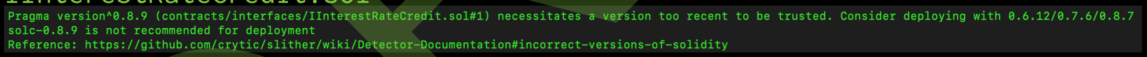
```
Pragma version0.8.9 (contracts/interfaces/IEscrow.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

IEscrowedLoan.sol



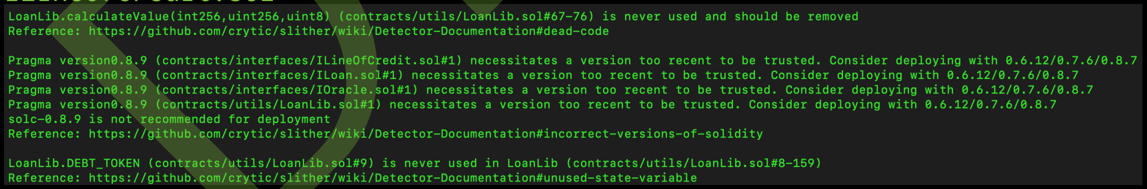
```
Pragma version0.8.9 (contracts/interfaces/IEscrowedLoan.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

IInterestRateCredit.sol



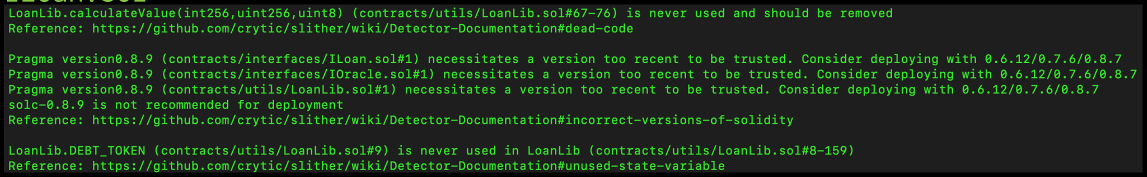
```
Pragma version^0.8.9 (contracts/interfaces/IInterestRateCredit.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

ILineOfCredit.sol



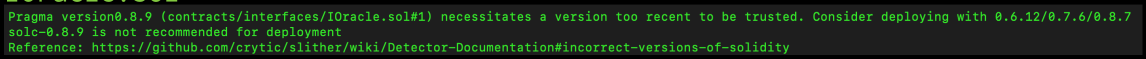
```
LoanLib.calculateValue(int256,uint256,uint8) (contracts/utils/LoanLib.sol#67-76) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version0.8.9 (contracts/interfaces/ILineOfCredit.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.9 (contracts/interfaces/ILoan.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.9 (contracts/interfaces/IOracle.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.9 (contracts/utils/LoanLib.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

LoanLib.DEBT_TOKEN (contracts/utils/LoanLib.sol#9) is never used in LoanLib (contracts/utils/LoanLib.sol#8-159)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable
```

ILoan.sol



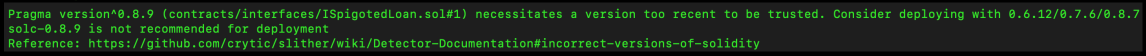
```
LoanLib.calculateValue(int256,uint256,uint8) (contracts/utils/LoanLib.sol#67-76) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version0.8.9 (contracts/interfaces/ILoan.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.9 (contracts/interfaces/IOracle.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.9 (contracts/utils/LoanLib.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

LoanLib.DEBT_TOKEN (contracts/utils/LoanLib.sol#9) is never used in LoanLib (contracts/utils/LoanLib.sol#8-159)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable
```

IOracle.sol



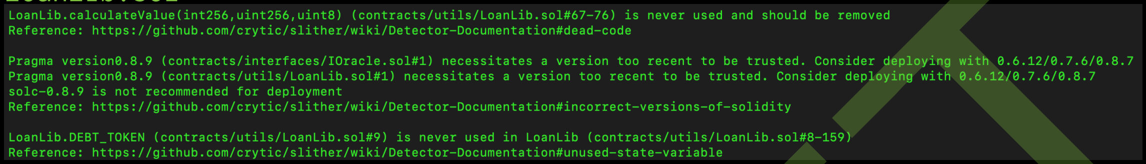
```
Pragma version0.8.9 (contracts/interfaces/IOracle.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

ISpigotedLoan.sol



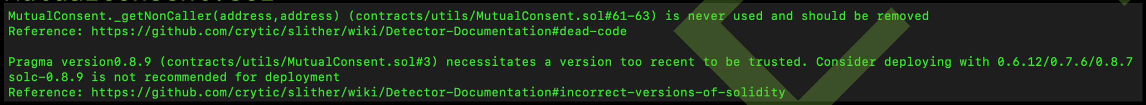
```
Pragma version^0.8.9 (contracts/interfaces/ISpigotedLoan.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

LoanLib.sol



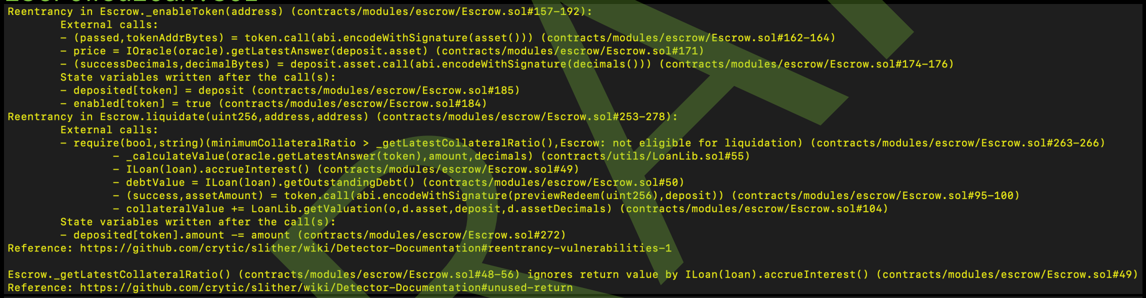
```
LoanLib.calculateValue(int256,uint256,uint8) (contracts/utils/LoanLib.sol#67-76) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version0.8.9 (contracts/interfaces/IOracle.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.9 (contracts/utils/LoanLib.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

LoanLib.DEBT_TOKEN (contracts/utils/LoanLib.sol#9) is never used in LoanLib (contracts/utils/LoanLib.sol#8-159)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable
```

MutualConsent.sol

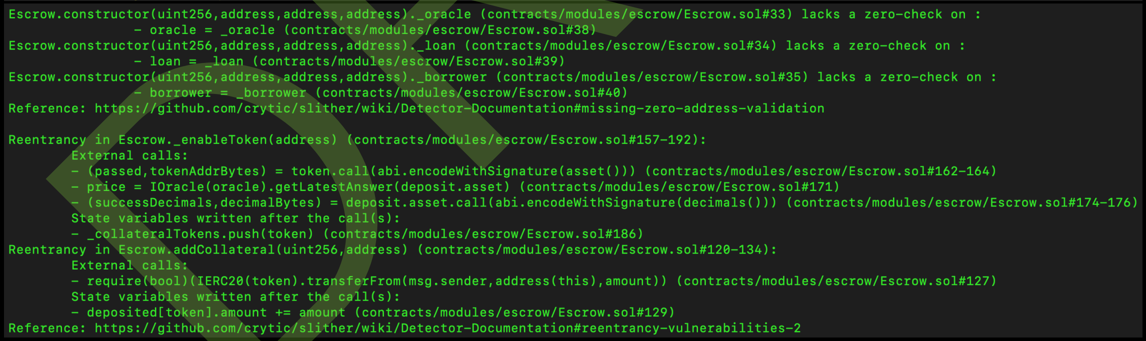


```
MutualConsent._getNonCaller(address,address) (contracts/utils/MutualConsent.sol#61-63) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version0.8.9 (contracts/utils/MutualConsent.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

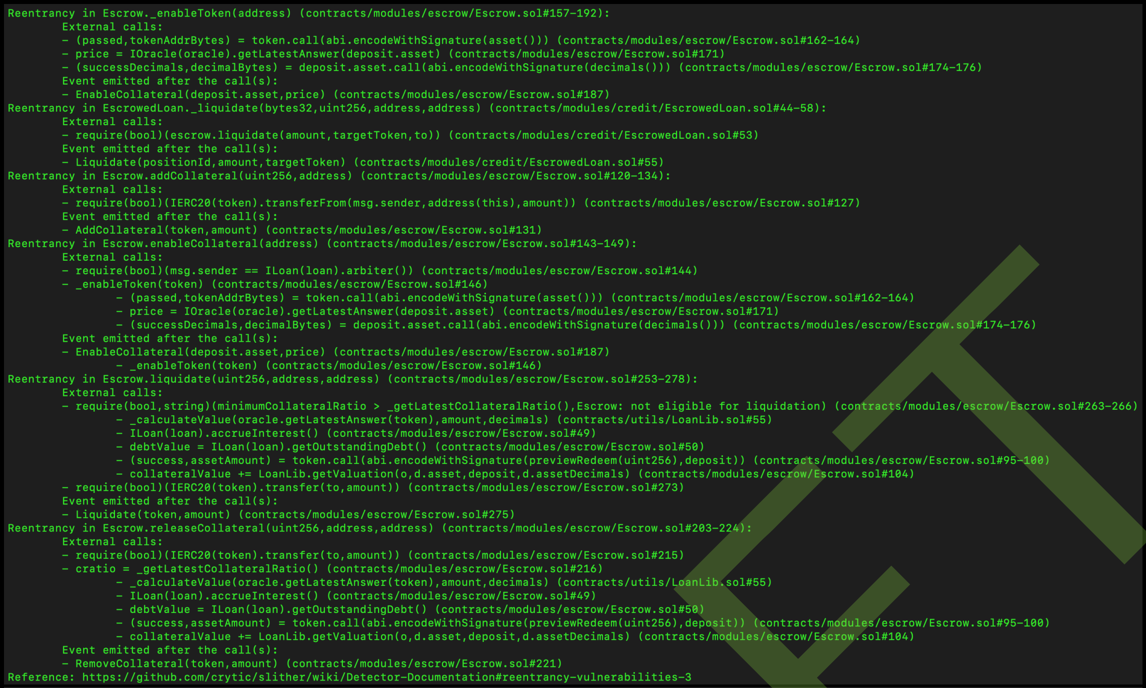
EscrowedLoan.sol



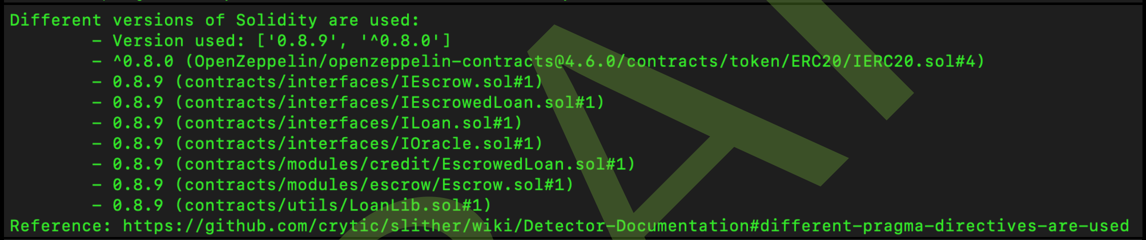
```
Reentrancy in Escrow._enableToken(address) (contracts/modules/escrow/Escrow.sol#157-192):
        External calls:
        - (passed,tokenAddrBytes) = token.call(abi.encodeWithSignature(asset())) (contracts/modules/escrow/Escrow.sol#162-164)
        - price = IOracle(oracle).getLatestAnswer(deposit.asset) (contracts/modules/escrow/Escrow.sol#171)
        - (successDecimals,decimalBytes) = deposit.asset.call(abi.encodeWithSignature(decimals())) (contracts/modules/escrow/Escrow.sol#174-176)
        State variables written after the call(s):
        - deposited[token] = deposit (contracts/modules/escrow/Escrow.sol#185)
        - enabled[token] = true (contracts/modules/escrow/Escrow.sol#184)
Reentrancy in Escrow.liquidate(uint256,address,address) (contracts/modules/escrow/Escrow.sol#253-278):
        External calls:
        - require(bool,string)(minimumCollateralRatio > _getLatestCollateralRatio(),Escrow: not eligible for liquidation) (contracts/modules/escrow/Escrow.sol#263-266)
                - _calculateValue(oracle.getLatestAnswer(token),amount,decimals) (contracts/utils/LoanLib.sol#55)
                - ILoan(loan).accrueInterest() (contracts/modules/escrow/Escrow.sol#49)
                - debtValue = ILoan(loan).getOutstandingDebt() (contracts/modules/escrow/Escrow.sol#50)
                - (success,assetAmount) = token.call(abi.encodeWithSignature(previewRedeem(uint256),deposit)) (contracts/modules/escrow/Escrow.sol#95-100)
                - collateralValue += LoanLib.getValuation(o,d.asset,deposit,d.assetDecimals) (contracts/modules/escrow/Escrow.sol#104)
        State variables written after the call(s):
        - deposited[token].amount -= amount (contracts/modules/escrow/Escrow.sol#272)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

Escrow._getLatestCollateralRatio() (contracts/modules/escrow/Escrow.sol#48-56) ignores return value by ILoan(loan).accrueInterest() (contracts/modules/escrow/Escrow.sol#49)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```



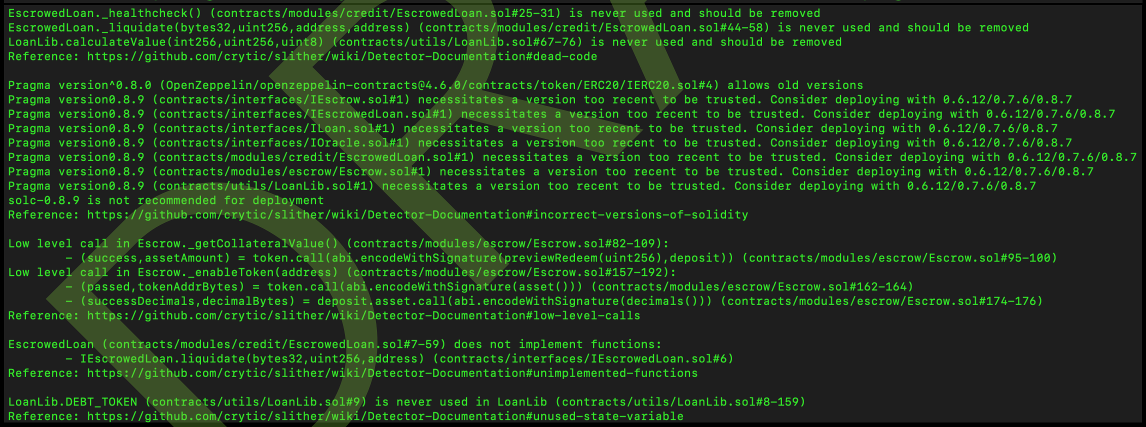
```
Escrow.constructor(uint256,address,address,address)._oracle (contracts/modules/escrow/Escrow.sol#33) lacks a zero-check on :
                - oracle = _oracle (contracts/modules/escrow/Escrow.sol#38)
Escrow.constructor(uint256,address,address,address)._loan (contracts/modules/escrow/Escrow.sol#34) lacks a zero-check on :
                - loan = _loan (contracts/modules/escrow/Escrow.sol#39)
Escrow.constructor(uint256,address,address,address)._borrower (contracts/modules/escrow/Escrow.sol#35) lacks a zero-check on :
                - borrower = _borrower (contracts/modules/escrow/Escrow.sol#40)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in Escrow._enableToken(address) (contracts/modules/escrow/Escrow.sol#157-192):
        External calls:
        - (passed,tokenAddrBytes) = token.call(abi.encodeWithSignature(asset())) (contracts/modules/escrow/Escrow.sol#162-164)
        - price = IOracle(oracle).getLatestAnswer(deposit.asset) (contracts/modules/escrow/Escrow.sol#171)
        - (successDecimals,decimalBytes) = deposit.asset.call(abi.encodeWithSignature(decimals())) (contracts/modules/escrow/Escrow.sol#174-176)
        State variables written after the call(s):
        - _collateralTokens.push(token) (contracts/modules/escrow/Escrow.sol#186)
Reentrancy in Escrow.addCollateral(uint256,address) (contracts/modules/escrow/Escrow.sol#120-134):
        External calls:
        - require(bool)(IERC20(token).transferFrom(msg.sender,address(this),amount)) (contracts/modules/escrow/Escrow.sol#127)
        State variables written after the call(s):
        - deposited[token].amount += amount (contracts/modules/escrow/Escrow.sol#129)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
```



```
Reentrancy in Escrow._enableToken(address) (contracts/modules/escrow/Escrow.sol#157-192):
        External calls:
        - (passed,tokenAddrBytes) = token.call(abi.encodeWithSignature(asset())) (contracts/modules/escrow/Escrow.sol#162-164)
        - price = IOracle(oracle).getLatestAnswer(deposit.asset) (contracts/modules/escrow/Escrow.sol#171)
        - (successDecimals,decimalBytes) = deposit.asset.call(abi.encodeWithSignature(decimals())) (contracts/modules/escrow/Escrow.sol#174-176)
        Event emitted after the call(s):
        - EnableCollateral(deposit.asset,price) (contracts/modules/escrow/Escrow.sol#187)
Reentrancy in EscrowedLoan._liquidate(bytes32,uint256,address,address) (contracts/modules/credit/EscrowedLoan.sol#44-58):
        External calls:
        - require(bool)(escrow.liquidate(amount,targetToken,to)) (contracts/modules/credit/EscrowedLoan.sol#53)
        Event emitted after the call(s):
        - Liquidate(positionId,amount,targetToken) (contracts/modules/credit/EscrowedLoan.sol#55)
Reentrancy in Escrow.addCollateral(uint256,address) (contracts/modules/escrow/Escrow.sol#120-134):
        External calls:
        - require(bool)(IERC20(token).transferFrom(msg.sender,address(this),amount)) (contracts/modules/escrow/Escrow.sol#127)
        Event emitted after the call(s):
        - AddCollateral(token,amount) (contracts/modules/escrow/Escrow.sol#131)
Reentrancy in Escrow.enableCollateral(address) (contracts/modules/escrow/Escrow.sol#143-149):
        External calls:
        - require(bool)(msg.sender == ILoan(loan).arbiter()) (contracts/modules/escrow/Escrow.sol#144)
        - _enableToken(token) (contracts/modules/escrow/Escrow.sol#146)
                - (passed,tokenAddrBytes) = token.call(abi.encodeWithSignature(asset())) (contracts/modules/escrow/Escrow.sol#162-164)
                - price = IOracle(oracle).getLatestAnswer(deposit.asset) (contracts/modules/escrow/Escrow.sol#171)
                - (successDecimals,decimalBytes) = deposit.asset.call(abi.encodeWithSignature(decimals())) (contracts/modules/escrow/Escrow.sol#174-176)
        Event emitted after the call(s):
        - EnableCollateral(deposit.asset,price) (contracts/modules/escrow/Escrow.sol#187)
                - _enableToken(token) (contracts/modules/escrow/Escrow.sol#146)
Reentrancy in Escrow.liquidate(uint256,address,address) (contracts/modules/escrow/Escrow.sol#253-278):
        External calls:
        - require(bool,string)(minimumCollateralRatio > _getLatestCollateralRatio(),Escrow: not eligible for liquidation) (contracts/modules/escrow/Escrow.sol#263-266)
                - _calculateValue(oracle.getLatestAnswer(token),amount,decimals) (contracts/utils/LoanLib.sol#55)
                - ILoan(loan).accrueInterest() (contracts/modules/escrow/Escrow.sol#49)
                - debtValue = ILoan(loan).getOutstandingDebt() (contracts/modules/escrow/Escrow.sol#50)
                - (success,assetAmount) = token.call(abi.encodeWithSignature(previewRedeem(uint256),deposit)) (contracts/modules/escrow/Escrow.sol#95-100)
                - collateralValue += LoanLib.getValuation(o,d.asset,deposit,d.assetDecimals) (contracts/modules/escrow/Escrow.sol#104)
        - require(bool)(IERC20(token).transfer(to,amount)) (contracts/modules/escrow/Escrow.sol#273)
        Event emitted after the call(s):
        - Liquidate(token,amount) (contracts/modules/escrow/Escrow.sol#275)
Reentrancy in Escrow.releaseCollateral(uint256,address,address) (contracts/modules/escrow/Escrow.sol#203-224):
        External calls:
        - require(bool)(IERC20(token).transfer(to,amount)) (contracts/modules/escrow/Escrow.sol#215)
        - cratio = _getLatestCollateralRatio() (contracts/modules/escrow/Escrow.sol#216)
                - _calculateValue(oracle.getLatestAnswer(token),amount,decimals) (contracts/utils/LoanLib.sol#55)
                - ILoan(loan).accrueInterest() (contracts/modules/escrow/Escrow.sol#49)
                - debtValue = ILoan(loan).getOutstandingDebt() (contracts/modules/escrow/Escrow.sol#50)
                - (success,assetAmount) = token.call(abi.encodeWithSignature(previewRedeem(uint256),deposit)) (contracts/modules/escrow/Escrow.sol#95-100)
                - collateralValue += LoanLib.getValuation(o,d.asset,deposit,d.assetDecimals) (contracts/modules/escrow/Escrow.sol#104)
        Event emitted after the call(s):
        - RemoveCollateral(token,amount) (contracts/modules/escrow/Escrow.sol#221)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

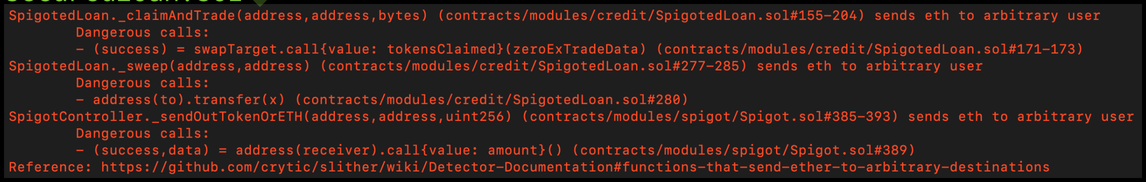


```
Different versions of Solidity are used:
        - Version used: ['0.8.9', '^0.8.0']
        - ^0.8.0 (OpenZeppelin/openzeppelin-contracts@4.6.0/contracts/token/ERC20/IERC20.sol#4)
        - 0.8.9 (contracts/interfaces/IEscrow.sol#1)
        - 0.8.9 (contracts/interfaces/IEscrowedLoan.sol#1)
        - 0.8.9 (contracts/interfaces/ILoan.sol#1)
        - 0.8.9 (contracts/interfaces/IOracle.sol#1)
        - 0.8.9 (contracts/modules/credit/EscrowedLoan.sol#1)
        - 0.8.9 (contracts/modules/escrow/Escrow.sol#1)
        - 0.8.9 (contracts/utils/LoanLib.sol#1)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
```

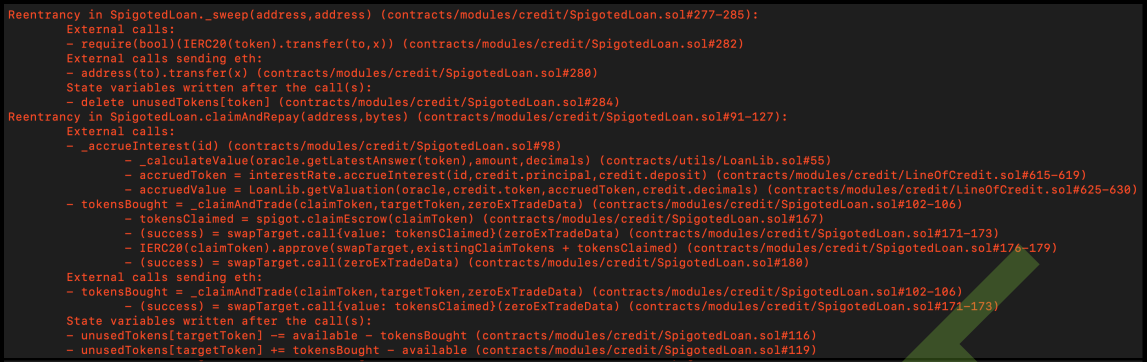


```
EscrowedLoan._healthcheck() (contracts/modules/credit/EscrowedLoan.sol#25-31) is never used and should be removed
EscrowedLoan._liquidate(bytes32,uint256,address,address) (contracts/modules/credit/EscrowedLoan.sol#44-58) is never used and should be removed
LoanLib.calculateValue(int256,uint256,uint8) (contracts/utils/LoanLib.sol#67-76) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (OpenZeppelin/openzeppelin-contracts@4.6.0/contracts/token/ERC20/IERC20.sol#4) allows old versions
Pragma version0.8.9 (contracts/interfaces/IEscrow.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.9 (contracts/interfaces/IEscrowedLoan.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.9 (contracts/interfaces/ILoan.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.9 (contracts/interfaces/IOracle.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.9 (contracts/modules/credit/EscrowedLoan.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.9 (contracts/modules/escrow/Escrow.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.9 (contracts/utils/LoanLib.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Escrow._getCollateralValue() (contracts/modules/escrow/Escrow.sol#82-109):
        - (success,assetAmount) = token.call(abi.encodeWithSignature(previewRedeem(uint256),deposit)) (contracts/modules/escrow/Escrow.sol#95-100)
Low level call in Escrow._enableToken(address) (contracts/modules/escrow/Escrow.sol#157-192):
        - (passed,tokenAddrBytes) = token.call(abi.encodeWithSignature(asset())) (contracts/modules/escrow/Escrow.sol#162-164)
        - (successDecimals,decimalBytes) = deposit.asset.call(abi.encodeWithSignature(decimals())) (contracts/modules/escrow/Escrow.sol#174-176)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

EscrowedLoan (contracts/modules/credit/EscrowedLoan.sol#7-59) does not implement functions:
        - IEscrowedLoan.liquidate(bytes32,uint256,address) (contracts/interfaces/IEscrowedLoan.sol#6)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unimplemented-functions

LoanLib.DEBT_TOKEN (contracts/utils/LoanLib.sol#9) is never used in LoanLib (contracts/utils/LoanLib.sol#8-159)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable
```

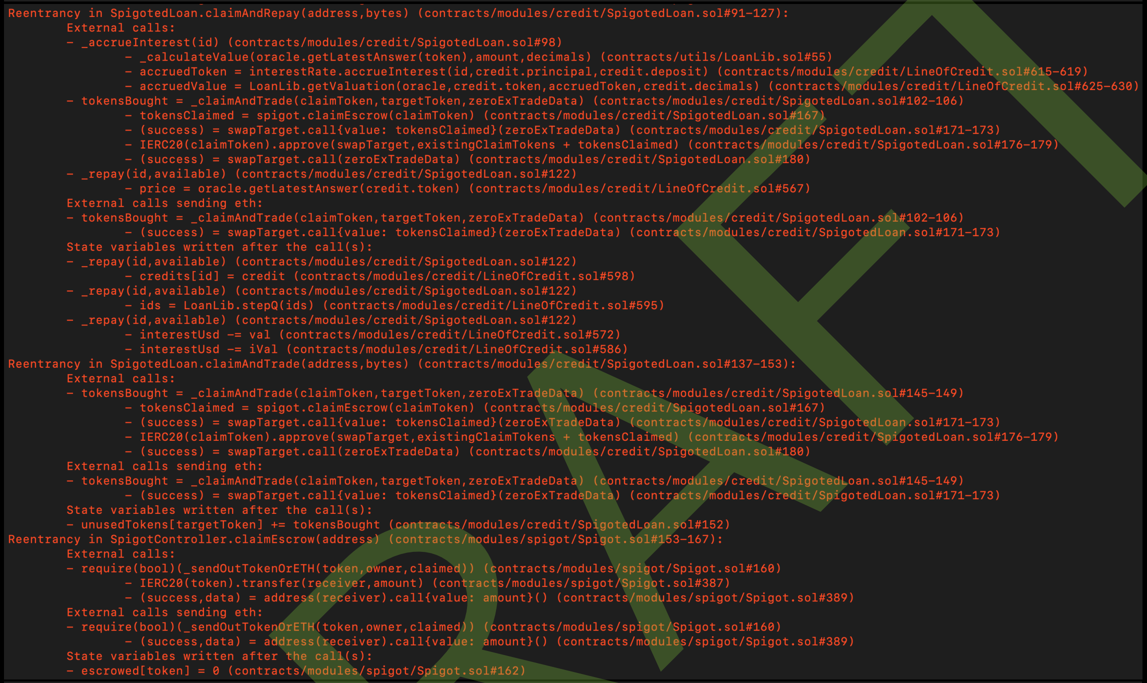
SecuredLoan.sol



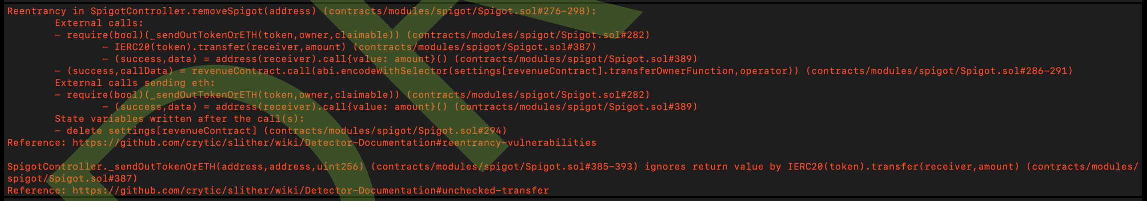
```
SpigotedLoan._claimAndTrade(address,address,bytes) (contracts/modules/credit/SpigotedLoan.sol#155-204) sends eth to arbitrary user
        Dangerous calls:
        - (success) = swapTarget.call{value: tokensClaimed}(zeroExTradeData) (contracts/modules/credit/SpigotedLoan.sol#171-173)
SpigotedLoan._sweep(address,address) (contracts/modules/credit/SpigotedLoan.sol#277-285) sends eth to arbitrary user
        Dangerous calls:
        - address(to).transfer(x) (contracts/modules/credit/SpigotedLoan.sol#280)
SpigotController._sendOutTokenOrETH(address,address,uint256) (contracts/modules/spigot/Spigot.sol#385-393) sends eth to arbitrary user
        Dangerous calls:
        - (success,data) = address(receiver).call{value: amount}() (contracts/modules/spigot/Spigot.sol#389)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
```



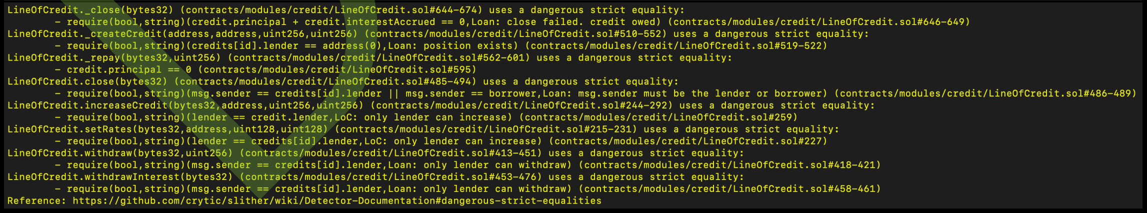
```
Reentrancy in SpigotedLoan._sweep(address,address) (contracts/modules/credit/SpigotedLoan.sol#277-285):
        External calls:
        - require(bool)(IERC20(token).transfer(to,x)) (contracts/modules/credit/SpigotedLoan.sol#282)
        External calls sending eth:
        - address(to).transfer(x) (contracts/modules/credit/SpigotedLoan.sol#280)
        State variables written after the call(s):
        - delete unusedTokens[token] (contracts/modules/credit/SpigotedLoan.sol#284)
Reentrancy in SpigotedLoan.claimAndRepay(address,bytes) (contracts/modules/credit/SpigotedLoan.sol#91-127):
        External calls:
        - _accrueInterest(id) (contracts/modules/credit/SpigotedLoan.sol#98)
                - _calculateValue(oracle.getLatestAnswer(token),amount,decimals) (contracts/utils/LoanLib.sol#55)
                - accruedToken = interestRate.accrueInterest(id,credit.principal,credit.deposit) (contracts/modules/credit/LineOfCredit.sol#615-619)
                - accruedValue = LoanLib.getValuation(oracle,credit.token,accruedToken,credit.decimals) (contracts/modules/credit/LineOfCredit.sol#625-630)
        - tokensBought = _claimAndTrade(claimToken,targetToken,zeroExTradeData) (contracts/modules/credit/SpigotedLoan.sol#102-106)
                - tokensClaimed = spigot.claimEscrow(claimToken) (contracts/modules/credit/SpigotedLoan.sol#167)
                - (success) = swapTarget.call{value: tokensClaimed}(zeroExTradeData) (contracts/modules/credit/SpigotedLoan.sol#171-173)
                - IERC20(claimToken).approve(swapTarget,existingClaimTokens + tokensClaimed) (contracts/modules/credit/SpigotedLoan.sol#176-179)
                - (success) = swapTarget.call(zeroExTradeData) (contracts/modules/credit/SpigotedLoan.sol#180)
        External calls sending eth:
        - tokensBought = _claimAndTrade(claimToken,targetToken,zeroExTradeData) (contracts/modules/credit/SpigotedLoan.sol#102-106)
                - (success) = swapTarget.call{value: tokensClaimed}(zeroExTradeData) (contracts/modules/credit/SpigotedLoan.sol#171-173)
        State variables written after the call(s):
        - unusedTokens[targetToken] -= available - tokensBought (contracts/modules/credit/SpigotedLoan.sol#116)
        - unusedTokens[targetToken] += tokensBought - available (contracts/modules/credit/SpigotedLoan.sol#119)
```



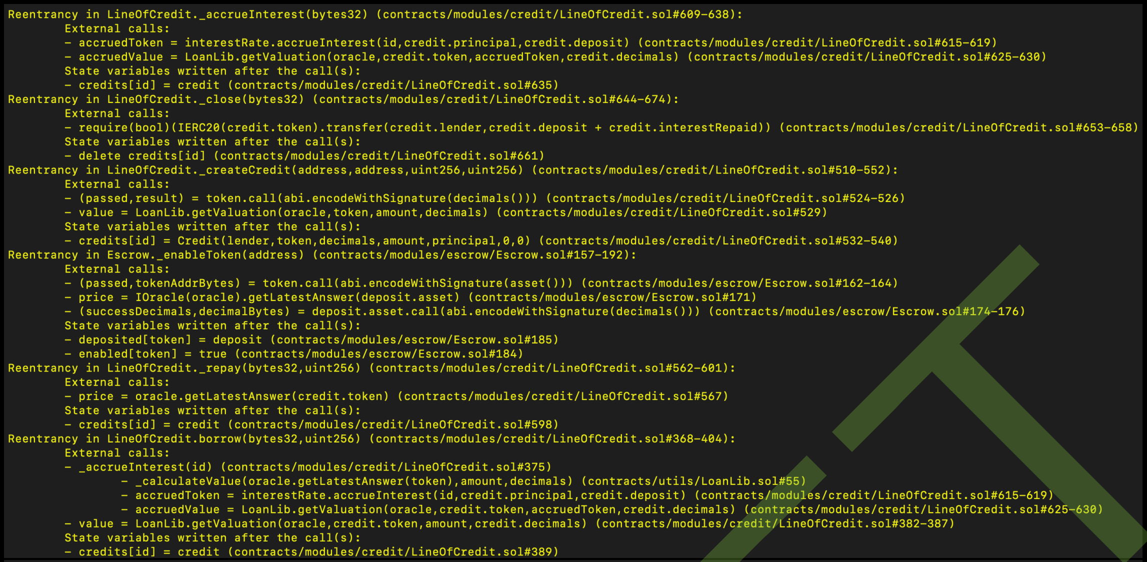
```
Reentrancy in SpigotedLoan.claimAndRepay(address,bytes) (contracts/modules/credit/SpigotedLoan.sol#91-127):
        External calls:
        - _accrueInterest(id) (contracts/modules/credit/SpigotedLoan.sol#98)
                - _calculateValue(oracle.getLatestAnswer(token),amount,decimals) (contracts/utils/LoanLib.sol#55)
                - accruedToken = interestRate.accrueInterest(id,credit.principal,credit.deposit) (contracts/modules/credit/LineOfCredit.sol#615-619)
                - accruedValue = LoanLib.getValuation(oracle,credit.token,accruedToken,credit.decimals) (contracts/modules/credit/LineOfCredit.sol#625-630)
        - tokensBought = _claimAndTrade(claimToken,targetToken,zeroExTradeData) (contracts/modules/credit/SpigotedLoan.sol#102-106)
                - tokensClaimed = spigot.claimEscrow(claimToken) (contracts/modules/credit/SpigotedLoan.sol#167)
                - (success) = swapTarget.call{value: tokensClaimed}(zeroExTradeData) (contracts/modules/credit/SpigotedLoan.sol#171-173)
                - IERC20(claimToken).approve(swapTarget,existingClaimTokens + tokensClaimed) (contracts/modules/credit/SpigotedLoan.sol#176-179)
                - (success) = swapTarget.call(zeroExTradeData) (contracts/modules/credit/SpigotedLoan.sol#180)
        - _repay(id,available) (contracts/modules/credit/SpigotedLoan.sol#122)
                - price = oracle.getLatestAnswer(credit.token) (contracts/modules/credit/LineOfCredit.sol#567)
        External calls sending eth:
        - tokensBought = _claimAndTrade(claimToken,targetToken,zeroExTradeData) (contracts/modules/credit/SpigotedLoan.sol#102-106)
                - (success) = swapTarget.call{value: tokensClaimed}(zeroExTradeData) (contracts/modules/credit/SpigotedLoan.sol#171-173)
        State variables written after the call(s):
        - _repay(id,available) (contracts/modules/credit/SpigotedLoan.sol#122)
                - credits[id] = credit (contracts/modules/credit/LineOfCredit.sol#598)
        - _repay(id,available) (contracts/modules/credit/SpigotedLoan.sol#122)
                - ids = LoanLib.stepQ(ids) (contracts/modules/credit/LineOfCredit.sol#595)
        - _repay(id,available) (contracts/modules/credit/SpigotedLoan.sol#122)
                - interestUsd -= val (contracts/modules/credit/LineOfCredit.sol#572)
                - interestUsd -= iVal (contracts/modules/credit/LineOfCredit.sol#586)
Reentrancy in SpigotedLoan.claimAndTrade(address,bytes) (contracts/modules/credit/SpigotedLoan.sol#137-153):
        External calls:
        - tokensBought = _claimAndTrade(claimToken,targetToken,zeroExTradeData) (contracts/modules/credit/SpigotedLoan.sol#145-149)
                - tokensClaimed = spigot.claimEscrow(claimToken) (contracts/modules/credit/SpigotedLoan.sol#167)
                - (success) = swapTarget.call{value: tokensClaimed}(zeroExTradeData) (contracts/modules/credit/SpigotedLoan.sol#171-173)
                - IERC20(claimToken).approve(swapTarget,existingClaimTokens + tokensClaimed) (contracts/modules/credit/SpigotedLoan.sol#176-179)
                - (success) = swapTarget.call(zeroExTradeData) (contracts/modules/credit/SpigotedLoan.sol#180)
        External calls sending eth:
        - tokensBought = _claimAndTrade(claimToken,targetToken,zeroExTradeData) (contracts/modules/credit/SpigotedLoan.sol#145-149)
                - (success) = swapTarget.call{value: tokensClaimed}(zeroExTradeData) (contracts/modules/credit/SpigotedLoan.sol#171-173)
        State variables written after the call(s):
        - unusedTokens[targetToken] += tokensBought (contracts/modules/credit/SpigotedLoan.sol#152)
Reentrancy in SpigotController.claimEscrow(address) (contracts/modules/spigot/Spigot.sol#153-167):
        External calls:
        - require(bool)(_sendOutTokenOrETH(token,owner,claimed)) (contracts/modules/spigot/Spigot.sol#160)
                - IERC20(token).transfer(receiver,amount) (contracts/modules/spigot/Spigot.sol#387)
                - (success,data) = address(receiver).call{value: amount}() (contracts/modules/spigot/Spigot.sol#389)
        External calls sending eth:
        - require(bool)(_sendOutTokenOrETH(token,owner,claimed)) (contracts/modules/spigot/Spigot.sol#160)
                - (success,data) = address(receiver).call{value: amount}() (contracts/modules/spigot/Spigot.sol#389)
        State variables written after the call(s):
        - escrowed[token] = 0 (contracts/modules/spigot/Spigot.sol#162)
```



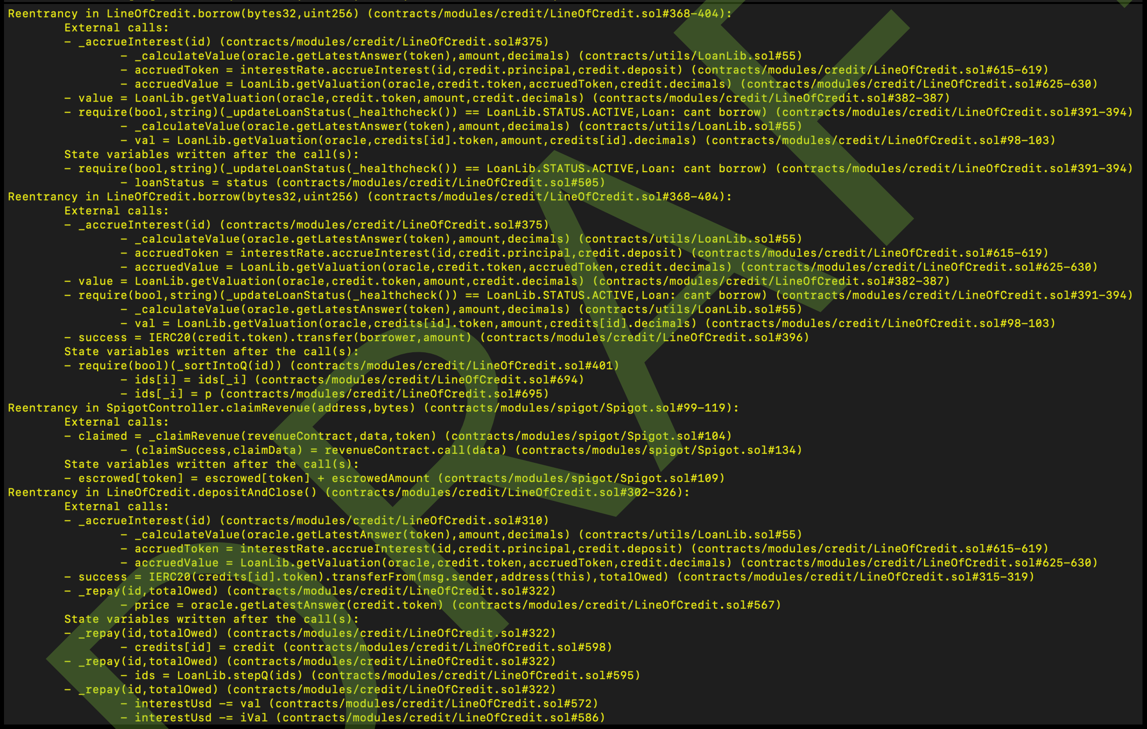
```
Reentrancy in SpigotController.removeSpigot(address) (contracts/modules/spigot/Spigot.sol#276-298):
        External calls:
        - require(bool)(_sendOutTokenOrETH(token,owner,claimable)) (contracts/modules/spigot/Spigot.sol#282)
                - IERC20(token).transfer(receiver,amount) (contracts/modules/spigot/Spigot.sol#387)
                - (success,data) = address(receiver).call{value: amount}() (contracts/modules/spigot/Spigot.sol#389)
        - (success,callData) = revenueContract.call(abi.encodeWithSelector(settings[revenueContract].transferOwnerFunction,operator)) (contracts/modules/spigot/Spigot.sol#286-291)
        External calls sending eth:
        - require(bool)(_sendOutTokenOrETH(token,owner,claimable)) (contracts/modules/spigot/Spigot.sol#282)
                - (success,data) = address(receiver).call{value: amount}() (contracts/modules/spigot/Spigot.sol#389)
        State variables written after the call(s):
        - delete settings[revenueContract] (contracts/modules/spigot/Spigot.sol#294)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities

SpigotController._sendOutTokenOrETH(address,address,uint256) (contracts/modules/spigot/Spigot.sol#385-393) ignores return value by IERC20(token).transfer(receiver,amount) (contracts/modules/
spigot/Spigot.sol#387)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
```



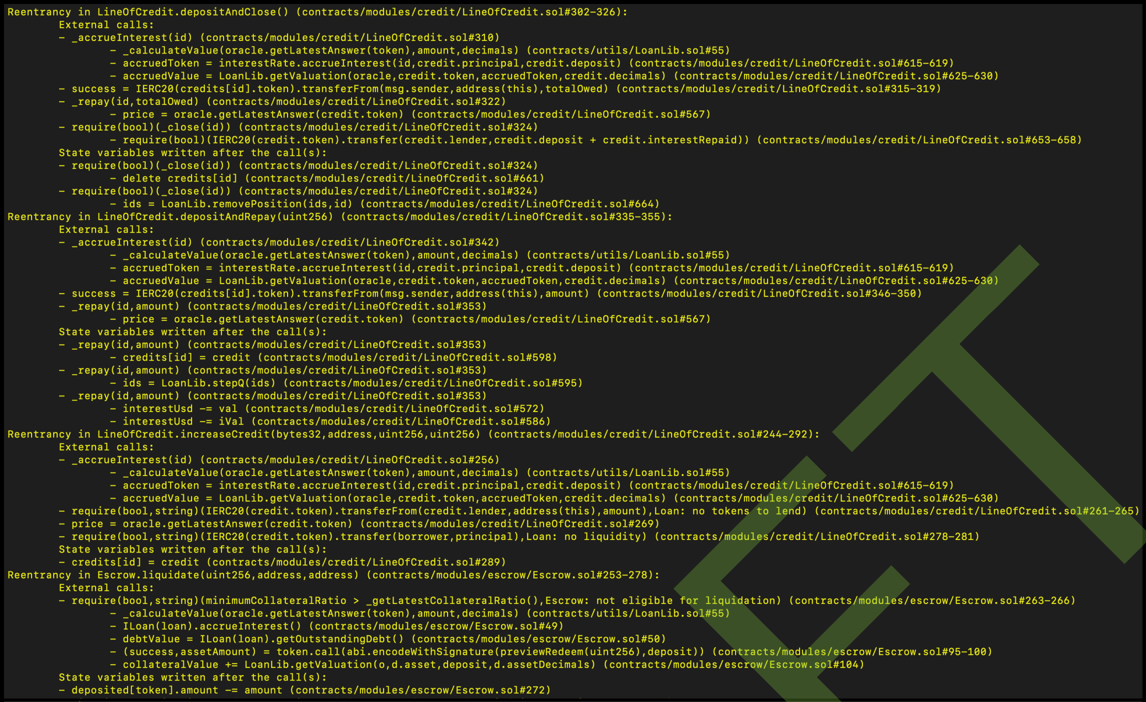
```
LineOfCredit._close(bytes32) (contracts/modules/credit/LineOfCredit.sol#644-674) uses a dangerous strict equality:
        - require(bool,string)(credit.principal + credit.interestAccrued == 0,Loan: close failed. credit owed) (contracts/modules/credit/LineOfCredit.sol#646-649)
LineOfCredit._createCredit(address,address,uint256,uint256) (contracts/modules/credit/LineOfCredit.sol#510-552) uses a dangerous strict equality:
        - require(bool,string)(credits[id].lender == address(0),Loan: position exists) (contracts/modules/credit/LineOfCredit.sol#519-522)
LineOfCredit._repay(bytes32,uint256) (contracts/modules/credit/LineOfCredit.sol#562-601) uses a dangerous strict equality:
        - credit.principal == 0 (contracts/modules/credit/LineOfCredit.sol#595)
LineOfCredit.close(bytes32) (contracts/modules/credit/LineOfCredit.sol#485-494) uses a dangerous strict equality:
        - require(bool,string)(msg.sender == credits[id].lender || msg.sender == borrower,Loan: msg.sender must be the lender or borrower) (contracts/modules/credit/LineOfCredit.sol#486-489)
LineOfCredit.increaseCredit(bytes32,address,uint256,uint256) (contracts/modules/credit/LineOfCredit.sol#244-292) uses a dangerous strict equality:
        - require(bool,string)(lender == credit.lender,LoC: only lender can increase) (contracts/modules/credit/LineOfCredit.sol#259)
LineOfCredit.setRates(bytes32,address,uint128,uint128) (contracts/modules/credit/LineOfCredit.sol#215-231) uses a dangerous strict equality:
        - require(bool,string)(lender == credits[id].lender,LoC: only lender can increase) (contracts/modules/credit/LineOfCredit.sol#227)
LineOfCredit.withdraw(bytes32,uint256) (contracts/modules/credit/LineOfCredit.sol#413-451) uses a dangerous strict equality:
        - require(bool,string)(msg.sender == credits[id].lender,Loan: only lender can withdraw) (contracts/modules/credit/LineOfCredit.sol#418-421)
LineOfCredit.withdrawInterest(bytes32) (contracts/modules/credit/LineOfCredit.sol#453-476) uses a dangerous strict equality:
        - require(bool,string)(msg.sender == credits[id].lender,Loan: only lender can withdraw) (contracts/modules/credit/LineOfCredit.sol#458-461)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
```



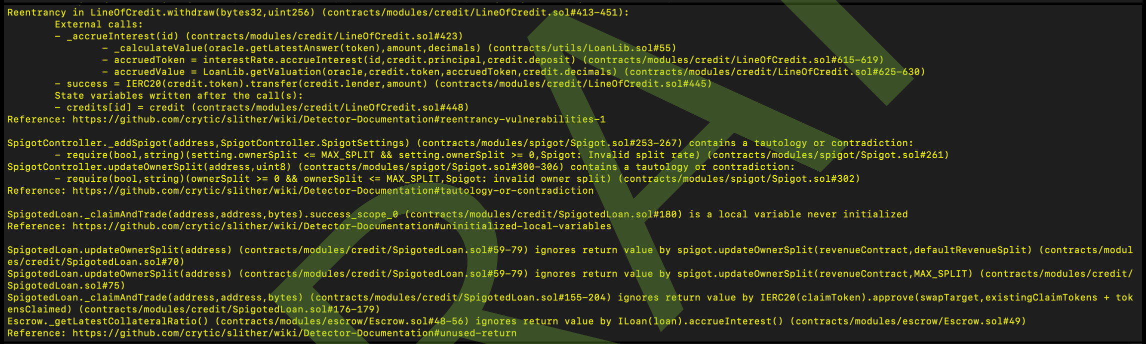
```
Reentrancy in LineOfCredit._accrueInterest(bytes32) (contracts/modules/credit/LineOfCredit.sol#609-638):
        External calls:
        - accruedToken = interestRate.accrueInterest(id,credit.principal,credit.deposit) (contracts/modules/credit/LineOfCredit.sol#615-619)
        - accruedValue = LoanLib.getValuation(oracle,credit.token,accruedToken,credit.decimals) (contracts/modules/credit/LineOfCredit.sol#625-630)
        State variables written after the call(s):
        - credits[id] = credit (contracts/modules/credit/LineOfCredit.sol#635)
Reentrancy in LineOfCredit._close(bytes32) (contracts/modules/credit/LineOfCredit.sol#644-674):
        External calls:
        - require(bool)(IERC20(credit.token).transfer(credit.lender,credit.deposit + credit.interestRepaid)) (contracts/modules/credit/LineOfCredit.sol#653-658)
        State variables written after the call(s):
        - delete credits[id] (contracts/modules/credit/LineOfCredit.sol#661)
Reentrancy in LineOfCredit._createCredit(address,address,uint256,uint256) (contracts/modules/credit/LineOfCredit.sol#510-552):
        External calls:
        - (passed,result) = token.call(abi.encodeWithSignature(decimals())) (contracts/modules/credit/LineOfCredit.sol#524-526)
        - value = LoanLib.getValuation(oracle,token,amount,decimals) (contracts/modules/credit/LineOfCredit.sol#529)
        State variables written after the call(s):
        - credits[id] = Credit(lender,token,decimals,amount,principal,0,0) (contracts/modules/credit/LineOfCredit.sol#532-540)
Reentrancy in Escrow._enableToken(address) (contracts/modules/escrow/Escrow.sol#157-192):
        External calls:
        - (passed,tokenAddrBytes) = token.call(abi.encodeWithSignature(asset())) (contracts/modules/escrow/Escrow.sol#162-164)
        - price = IOracle(oracle).getLatestAnswer(deposit.asset) (contracts/modules/escrow/Escrow.sol#171)
        - (successDecimals,decimalBytes) = deposit.asset.call(abi.encodeWithSignature(decimals())) (contracts/modules/escrow/Escrow.sol#174-176)
        State variables written after the call(s):
        - deposited[token] = deposit (contracts/modules/escrow/Escrow.sol#185)
        - enabled[token] = true (contracts/modules/escrow/Escrow.sol#184)
Reentrancy in LineOfCredit._repay(bytes32,uint256) (contracts/modules/credit/LineOfCredit.sol#562-601):
        External calls:
        - price = oracle.getLatestAnswer(credit.token) (contracts/modules/credit/LineOfCredit.sol#567)
        State variables written after the call(s):
        - credits[id] = credit (contracts/modules/credit/LineOfCredit.sol#598)
Reentrancy in LineOfCredit.borrow(bytes32,uint256) (contracts/modules/credit/LineOfCredit.sol#368-404):
        External calls:
        - _accrueInterest(id) (contracts/modules/credit/LineOfCredit.sol#375)
                - _calculateValue(oracle.getLatestAnswer(token),amount,decimals) (contracts/utils/LoanLib.sol#55)
                - accruedToken = interestRate.accrueInterest(id,credit.principal,credit.deposit) (contracts/modules/credit/LineOfCredit.sol#615-619)
                - accruedValue = LoanLib.getValuation(oracle,credit.token,accruedToken,credit.decimals) (contracts/modules/credit/LineOfCredit.sol#625-630)
        - value = LoanLib.getValuation(oracle,credit.token,amount,credit.decimals) (contracts/modules/credit/LineOfCredit.sol#382-387)
        State variables written after the call(s):
        - credits[id] = credit (contracts/modules/credit/LineOfCredit.sol#389)
```



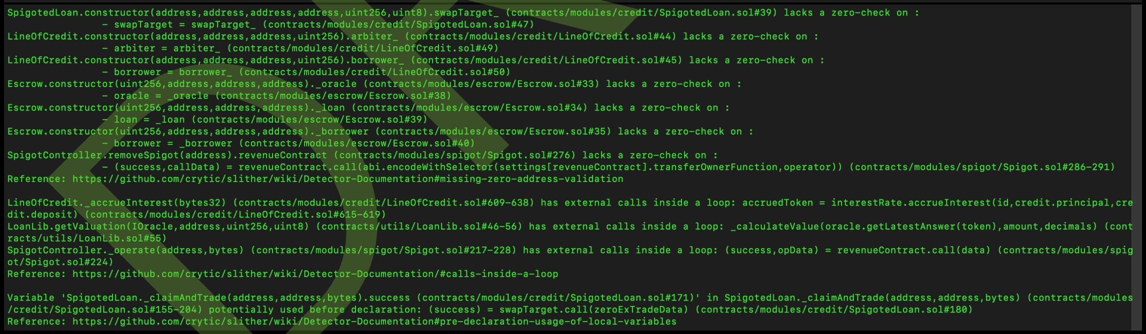
```
Reentrancy in LineOfCredit.borrow(bytes32,uint256) (contracts/modules/credit/LineOfCredit.sol#368-404):
        External calls:
        - _accrueInterest(id) (contracts/modules/credit/LineOfCredit.sol#375)
                - _calculateValue(oracle.getLatestAnswer(token),amount,decimals) (contracts/utils/LoanLib.sol#55)
                - accruedToken = interestRate.accrueInterest(id,credit.principal,credit.deposit) (contracts/modules/credit/LineOfCredit.sol#615-619)
                - accruedValue = LoanLib.getValuation(oracle,credit.token,accruedToken,credit.decimals) (contracts/modules/credit/LineOfCredit.sol#625-630)
        - value = LoanLib.getValuation(oracle,credit.token,amount,credit.decimals) (contracts/modules/credit/LineOfCredit.sol#382-387)
        - require(bool,string)(_updateLoanStatus(_healthcheck()) == LoanLib.STATUS.ACTIVE,Loan: cant borrow) (contracts/modules/credit/LineOfCredit.sol#391-394)
                - _calculateValue(oracle.getLatestAnswer(token),amount,decimals) (contracts/utils/LoanLib.sol#55)
                - val = LoanLib.getValuation(oracle,credits[id].token,amount,credits[id].decimals) (contracts/modules/credit/LineOfCredit.sol#98-103)
        State variables written after the call(s):
        - require(bool,string)(_updateLoanStatus(_healthcheck()) == LoanLib.STATUS.ACTIVE,Loan: cant borrow) (contracts/modules/credit/LineOfCredit.sol#391-394)
                - loanStatus = status (contracts/modules/credit/LineOfCredit.sol#505)
Reentrancy in LineOfCredit.borrow(bytes32,uint256) (contracts/modules/credit/LineOfCredit.sol#368-404):
        External calls:
        - _accrueInterest(id) (contracts/modules/credit/LineOfCredit.sol#375)
                - _calculateValue(oracle.getLatestAnswer(token),amount,decimals) (contracts/utils/LoanLib.sol#55)
                - accruedToken = interestRate.accrueInterest(id,credit.principal,credit.deposit) (contracts/modules/credit/LineOfCredit.sol#615-619)
                - accruedValue = LoanLib.getValuation(oracle,credit.token,accruedToken,credit.decimals) (contracts/modules/credit/LineOfCredit.sol#625-630)
        - value = LoanLib.getValuation(oracle,credit.token,amount,credit.decimals) (contracts/modules/credit/LineOfCredit.sol#382-387)
        - require(bool,string)(_updateLoanStatus(_healthcheck()) == LoanLib.STATUS.ACTIVE,Loan: cant borrow) (contracts/modules/credit/LineOfCredit.sol#391-394)
                - _calculateValue(oracle.getLatestAnswer(token),amount,decimals) (contracts/utils/LoanLib.sol#55)
                - val = LoanLib.getValuation(oracle,credits[id].token,amount,credits[id].decimals) (contracts/modules/credit/LineOfCredit.sol#98-103)
        - success = IERC20(credit.token).transfer(borrower,amount) (contracts/modules/credit/LineOfCredit.sol#396)
        State variables written after the call(s):
        - require(bool)(_sortIntoQ(id)) (contracts/modules/credit/LineOfCredit.sol#401)
                - ids[i] = ids[_i] (contracts/modules/credit/LineOfCredit.sol#694)
                - ids[_i] = p (contracts/modules/credit/LineOfCredit.sol#695)
Reentrancy in SpigotController.claimRevenue(address,bytes) (contracts/modules/spigot/Spigot.sol#99-119):
        External calls:
        - claimed = _claimRevenue(revenueContract,data,token) (contracts/modules/spigot/Spigot.sol#104)
                - (claimSuccess,claimData) = revenueContract.call(data) (contracts/modules/spigot/Spigot.sol#134)
        State variables written after the call(s):
        - escrowed[token] = escrowed[token] + escrowedAmount (contracts/modules/spigot/Spigot.sol#109)
Reentrancy in LineOfCredit.depositAndClose() (contracts/modules/credit/LineOfCredit.sol#302-326):
        External calls:
        - _accrueInterest(id) (contracts/modules/credit/LineOfCredit.sol#310)
                - _calculateValue(oracle.getLatestAnswer(token),amount,decimals) (contracts/utils/LoanLib.sol#55)
                - accruedToken = interestRate.accrueInterest(id,credit.principal,credit.deposit) (contracts/modules/credit/LineOfCredit.sol#615-619)
                - accruedValue = LoanLib.getValuation(oracle,credit.token,accruedToken,credit.decimals) (contracts/modules/credit/LineOfCredit.sol#625-630)
        - success = IERC20(credits[id].token).transferFrom(msg.sender,address(this),totalOwed) (contracts/modules/credit/LineOfCredit.sol#315-319)
        - _repay(id,totalOwed) (contracts/modules/credit/LineOfCredit.sol#322)
                - price = oracle.getLatestAnswer(credit.token) (contracts/modules/credit/LineOfCredit.sol#567)
        State variables written after the call(s):
        - _repay(id,totalOwed) (contracts/modules/credit/LineOfCredit.sol#322)
                - credits[id] = credit (contracts/modules/credit/LineOfCredit.sol#598)
        - _repay(id,totalOwed) (contracts/modules/credit/LineOfCredit.sol#322)
                - ids = LoanLib.stepQ(ids) (contracts/modules/credit/LineOfCredit.sol#595)
        - _repay(id,totalOwed) (contracts/modules/credit/LineOfCredit.sol#322)
                - interestUsd -= val (contracts/modules/credit/LineOfCredit.sol#572)
                - interestUsd -= iVal (contracts/modules/credit/LineOfCredit.sol#586)
```



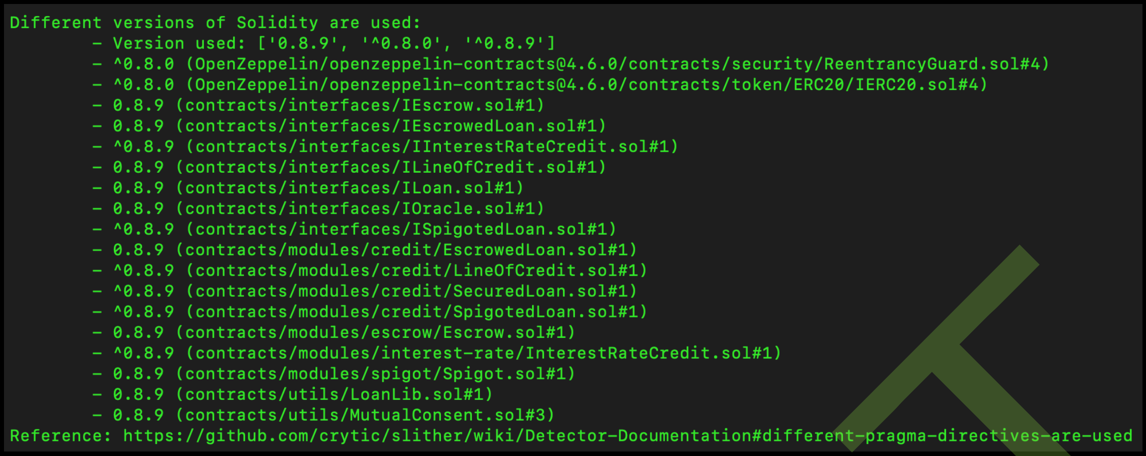
```
Reentrancy in LineOfCredit.depositAndClose() (contracts/modules/credit/LineOfCredit.sol#302-326):
	External calls:
	- _accrueInterest(id) (contracts/modules/credit/LineOfCredit.sol#310)
		- _calculateValue(oracle.getLatestAnswer(token),amount,decimals) (contracts/utils/LoanLib.sol#55)
		- accruedToken = interestRate.accrueInterest(id,credit.principal,credit.deposit) (contracts/modules/credit/LineOfCredit.sol#615-619)
		- accruedValue = LoanLib.getValuation(oracle,credit.token,accruedToken,credit.decimals) (contracts/modules/credit/LineOfCredit.sol#625-630)
	- success = IERC20(credits[id].token).transferFrom(msg.sender,address(this),totalOwed) (contracts/modules/credit/LineOfCredit.sol#315-319)
	- _repay(id,totalOwed) (contracts/modules/credit/LineOfCredit.sol#322)
		- price = oracle.getLatestAnswer(credit.token) (contracts/modules/credit/LineOfCredit.sol#567)
	- require(bool)(_close(id)) (contracts/modules/credit/LineOfCredit.sol#324)
		- require(bool)(IERC20(credit.token).transfer(credit.lender,credit.deposit + credit.interestRepaid)) (contracts/modules/credit/LineOfCredit.sol#653-658)
	State variables written after the call(s):
	- require(bool)(_close(id)) (contracts/modules/credit/LineOfCredit.sol#324)
		- delete credits[id] (contracts/modules/credit/LineOfCredit.sol#661)
	- require(bool)(_close(id)) (contracts/modules/credit/LineOfCredit.sol#324)
		- ids = LoanLib.removePosition(ids,id) (contracts/modules/credit/LineOfCredit.sol#664)
Reentrancy in LineOfCredit.depositAndRepay(uint256) (contracts/modules/credit/LineOfCredit.sol#335-355):
	External calls:
	- _accrueInterest(id) (contracts/modules/credit/LineOfCredit.sol#342)
		- _calculateValue(oracle.getLatestAnswer(token),amount,decimals) (contracts/utils/LoanLib.sol#55)
		- accruedToken = interestRate.accrueInterest(id,credit.principal,credit.deposit) (contracts/modules/credit/LineOfCredit.sol#615-619)
		- accruedValue = LoanLib.getValuation(oracle,credit.token,accruedToken,credit.decimals) (contracts/modules/credit/LineOfCredit.sol#625-630)
	- success = IERC20(credits[id].token).transferFrom(msg.sender,address(this),amount) (contracts/modules/credit/LineOfCredit.sol#346-350)
	- _repay(id,amount) (contracts/modules/credit/LineOfCredit.sol#353)
		- price = oracle.getLatestAnswer(credit.token) (contracts/modules/credit/LineOfCredit.sol#567)
	State variables written after the call(s):
	- _repay(id,amount) (contracts/modules/credit/LineOfCredit.sol#353)
		- credits[id] = credit (contracts/modules/credit/LineOfCredit.sol#598)
	- _repay(id,amount) (contracts/modules/credit/LineOfCredit.sol#353)
		- ids = LoanLib.stepQ(ids) (contracts/modules/credit/LineOfCredit.sol#595)
	- _repay(id,amount) (contracts/modules/credit/LineOfCredit.sol#353)
		- interestUsd -= val (contracts/modules/credit/LineOfCredit.sol#572)
		- interestUsd -= iVal (contracts/modules/credit/LineOfCredit.sol#586)
Reentrancy in LineOfCredit.increaseCredit(bytes32,address,uint256,uint256) (contracts/modules/credit/LineOfCredit.sol#244-292):
	External calls:
	- _accrueInterest(id) (contracts/modules/credit/LineOfCredit.sol#256)
		- _calculateValue(oracle.getLatestAnswer(token),amount,decimals) (contracts/utils/LoanLib.sol#55)
		- accruedToken = interestRate.accrueInterest(id,credit.principal,credit.deposit) (contracts/modules/credit/LineOfCredit.sol#615-619)
		- accruedValue = LoanLib.getValuation(oracle,credit.token,accruedToken,credit.decimals) (contracts/modules/credit/LineOfCredit.sol#625-630)
	- require(bool,string)(IERC20(credit.token).transferFrom(credit.lender,address(this),amount),Loan: no tokens to lend) (contracts/modules/credit/LineOfCredit.sol#261-265)
	- price = oracle.getLatestAnswer(credit.token) (contracts/modules/credit/LineOfCredit.sol#269)
	- require(bool,string)(IERC20(credit.token).transfer(borrower,principal),Loan: no liquidity) (contracts/modules/credit/LineOfCredit.sol#278-281)
	State variables written after the call(s):
	- credits[id] = credit (contracts/modules/credit/LineOfCredit.sol#289)
Reentrancy in Escrow.liquidate(uint256,address,address) (contracts/modules/escrow/Escrow.sol#253-278):
	External calls:
	- require(bool,string)(minimumCollateralRatio > _getLatestCollateralRatio(),Escrow: not eligible for liquidation) (contracts/modules/escrow/Escrow.sol#263-266)
		- _calculateValue(oracle.getLatestAnswer(token),amount,decimals) (contracts/utils/LoanLib.sol#55)
		- ILoan(loan).accrueInterest() (contracts/modules/escrow/Escrow.sol#49)
		- debtValue = ILoan(loan).getOutstandingDebt() (contracts/modules/escrow/Escrow.sol#50)
		- (success,assetAmount) = token.call(abi.encodeWithSignature(previewRedeem(uint256),deposit)) (contracts/modules/escrow/Escrow.sol#95-100)
		- collateralValue += LoanLib.getValuation(o,d.asset,deposit,d.assetDecimals) (contracts/modules/escrow/Escrow.sol#104)
	State variables written after the call(s):
	- deposited[token].amount -= amount (contracts/modules/escrow/Escrow.sol#272)
```



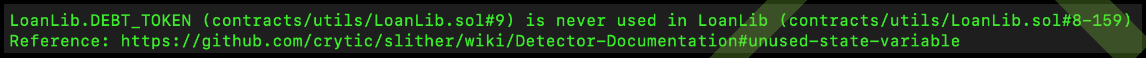
```
Reentrancy in LineOfCredit.withdraw(bytes32,uint256) (contracts/modules/credit/LineOfCredit.sol#413-451):
	External calls:
	- _accrueInterest(id) (contracts/modules/credit/LineOfCredit.sol#423)
		- _calculateValue(oracle.getLatestAnswer(token),amount,decimals) (contracts/utils/LoanLib.sol#55)
		- accruedToken = interestRate.accrueInterest(id,credit.principal,credit.deposit) (contracts/modules/credit/LineOfCredit.sol#615-619)
		- accruedValue = LoanLib.getValuation(oracle,credit.token,accruedToken,credit.decimals) (contracts/modules/credit/LineOfCredit.sol#625-630)
	- success = IERC20(credit.token).transfer(credit.lender,amount) (contracts/modules/credit/LineOfCredit.sol#445)
	State variables written after the call(s):
	- credits[id] = credit (contracts/modules/credit/LineOfCredit.sol#448)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

SpigotController._addSpigot(address,SpigotController.SpigotSettings) (contracts/modules/spigot/Spigot.sol#253-267) contains a tautology or contradiction:
	- require(bool,string)(setting.ownerSplit <= MAX_SPLIT && setting.ownerSplit >= 0,Spigot: Invalid split rate) (contracts/modules/spigot/Spigot.sol#261)
SpigotController.updateOwnerSplit(address,uint8) (contracts/modules/spigot/Spigot.sol#300-306) contains a tautology or contradiction:
	- require(bool,string)(ownerSplit >= 0 && ownerSplit <= MAX_SPLIT,Spigot: invalid owner split) (contracts/modules/spigot/Spigot.sol#302)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#tautology-or-contradiction

SpigotedLoan._claimAndTrade(address,address,bytes).success_scope_0 (contracts/modules/credit/SpigotedLoan.sol#180) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

SpigotedLoan.updateOwnerSplit(address) (contracts/modules/credit/SpigotedLoan.sol#59-79) ignores return value by spigot.updateOwnerSplit(revenueContract,defaultRevenueSplit) (contracts/modul
es/credit/SpigotedLoan.sol#70)
SpigotedLoan.updateOwnerSplit(address) (contracts/modules/credit/SpigotedLoan.sol#59-79) ignores return value by spigot.updateOwnerSplit(revenueContract,MAX_SPLIT) (contracts/modules/credit/
SpigotedLoan.sol#75)
SpigotedLoan._claimAndTrade(address,address,bytes) (contracts/modules/credit/SpigotedLoan.sol#155-204) ignores return value by IERC20(claimToken).approve(swapTarget,existingClaimTokens + tok
ensClaimed) (contracts/modules/credit/SpigotedLoan.sol#176-179)
Escrow._getLatestCollateralRatio() (contracts/modules/escrow/Escrow.sol#48-56) ignores return value by ILoan(loan).accrueInterest() (contracts/modules/escrow/Escrow.sol#49)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```



```
SpigotedLoan.constructor(address,address,address,address,uint256,uint8).swapTarget_ (contracts/modules/credit/SpigotedLoan.sol#39) lacks a zero-check on :
		- swapTarget = swapTarget_ (contracts/modules/credit/SpigotedLoan.sol#47)
LineOfCredit.constructor(address,address,address,uint256).arbiter_ (contracts/modules/credit/LineOfCredit.sol#44) lacks a zero-check on :
		- arbiter = arbiter_ (contracts/modules/credit/LineOfCredit.sol#49)
LineOfCredit.constructor(address,address,address,uint256).borrower_ (contracts/modules/credit/LineOfCredit.sol#45) lacks a zero-check on :
		- borrower = borrower_ (contracts/modules/credit/LineOfCredit.sol#50)
Escrow.constructor(uint256,address,address,address)._oracle (contracts/modules/escrow/Escrow.sol#33) lacks a zero-check on :
		- oracle = _oracle (contracts/modules/escrow/Escrow.sol#38)
Escrow.constructor(uint256,address,address,address)._loan (contracts/modules/escrow/Escrow.sol#34) lacks a zero-check on :
		- loan = _loan (contracts/modules/escrow/Escrow.sol#39)
Escrow.constructor(uint256,address,address,address)._borrower (contracts/modules/escrow/Escrow.sol#35) lacks a zero-check on :
		- borrower = _borrower (contracts/modules/escrow/Escrow.sol#40)
SpigotController.removeSpigot(address).revenueContract (contracts/modules/spigot/Spigot.sol#276) lacks a zero-check on :
		- (success,callData) = revenueContract.call(abi.encodeWithSelector(settings[revenueContract].transferOwnerFunction,operator)) (contracts/modules/spigot/Spigot.sol#286-291)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

LineOfCredit._accrueInterest(bytes32) (contracts/modules/credit/LineOfCredit.sol#609-638) has external calls inside a loop: accruedToken = interestRate.accrueInterest(id,credit.principal,cre
dit.deposit) (contracts/modules/credit/LineOfCredit.sol#615-619)
LoanLib.getValuation(IOracle,address,uint256,uint8) (contracts/utils/LoanLib.sol#46-56) has external calls inside a loop: _calculateValue(oracle.getLatestAnswer(token),amount,decimals) (cont
racts/utils/LoanLib.sol#55)
SpigotController._operate(address,bytes) (contracts/modules/spigot/Spigot.sol#217-228) has external calls inside a loop: (success,opData) = revenueContract.call(data) (contracts/modules/spig
ot/Spigot.sol#224)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

Variable 'SpigotedLoan._claimAndTrade(address,address,bytes).success (contracts/modules/credit/SpigotedLoan.sol#171)' in SpigotedLoan._claimAndTrade(address,address,bytes) (contracts/modules
/credit/SpigotedLoan.sol#155-204) potentially used before declaration: (success) = swapTarget.call(zeroExTradeData) (contracts/modules/credit/SpigotedLoan.sol#180)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables
```

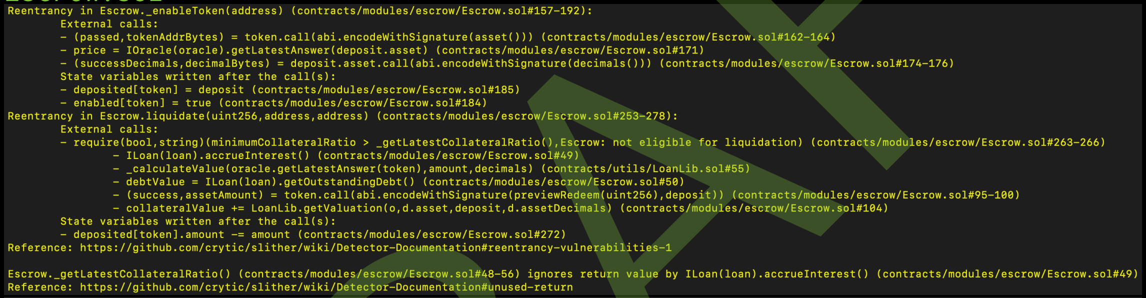


```
Different versions of Solidity are used:
        - Version used: ['0.8.9', '^0.8.0', '^0.8.9']
        - ^0.8.0 (OpenZeppelin/openzeppelin-contracts@4.6.0/contracts/security/ReentrancyGuard.sol#4)
        - ^0.8.0 (OpenZeppelin/openzeppelin-contracts@4.6.0/contracts/token/ERC20/IERC20.sol#4)
        - 0.8.9 (contracts/interfaces/IEscrow.sol#1)
        - 0.8.9 (contracts/interfaces/IEscrowedLoan.sol#1)
        - ^0.8.9 (contracts/interfaces/IInterestRateCredit.sol#1)
        - 0.8.9 (contracts/interfaces/ILineOfCredit.sol#1)
        - 0.8.9 (contracts/interfaces/ILoan.sol#1)
        - 0.8.9 (contracts/interfaces/IOracle.sol#1)
        - ^0.8.9 (contracts/interfaces/ISpigotedLoan.sol#1)
        - 0.8.9 (contracts/modules/credit/EscrowedLoan.sol#1)
        - ^0.8.9 (contracts/modules/credit/LineOfCredit.sol#1)
        - ^0.8.9 (contracts/modules/credit/SecuredLoan.sol#1)
        - ^0.8.9 (contracts/modules/credit/SpigotedLoan.sol#1)
        - 0.8.9 (contracts/modules/escrow/Escrow.sol#1)
        - ^0.8.9 (contracts/modules/interest-rate/InterestRateCredit.sol#1)
        - 0.8.9 (contracts/modules/spigot/Spigot.sol#1)
        - 0.8.9 (contracts/utils/LoanLib.sol#1)
        - 0.8.9 (contracts/utils/MutualConsent.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
```

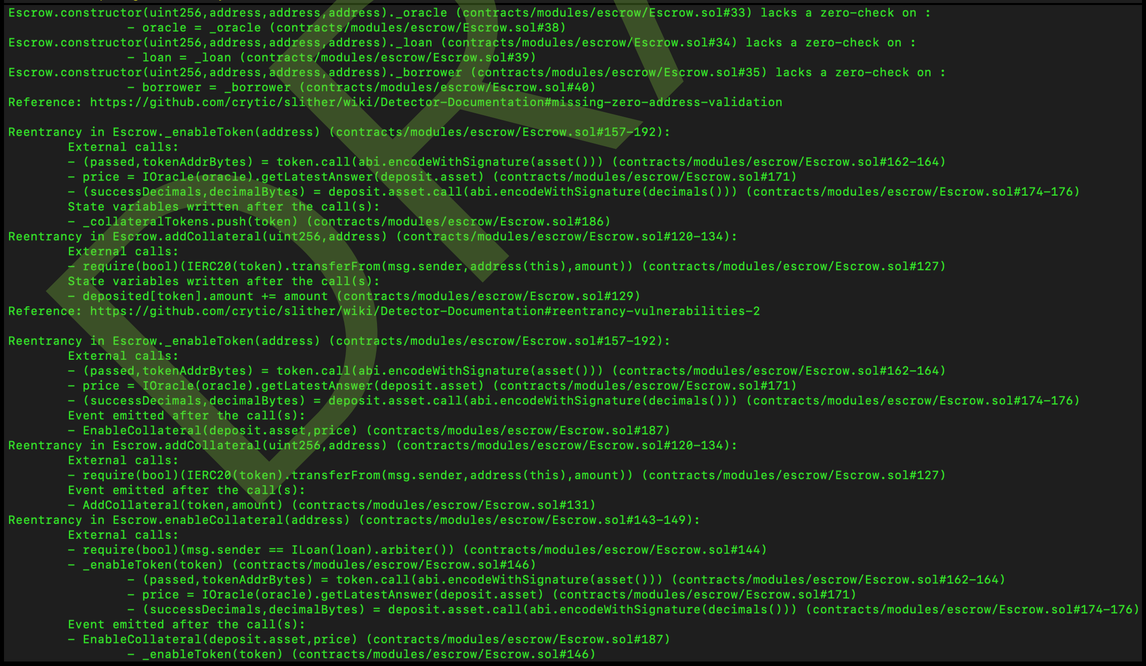


```
LoanLib.DEBT_TOKEN (contracts/utils/LoanLib.sol#9) is never used in LoanLib (contracts/utils/LoanLib.sol#8-159)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable
```

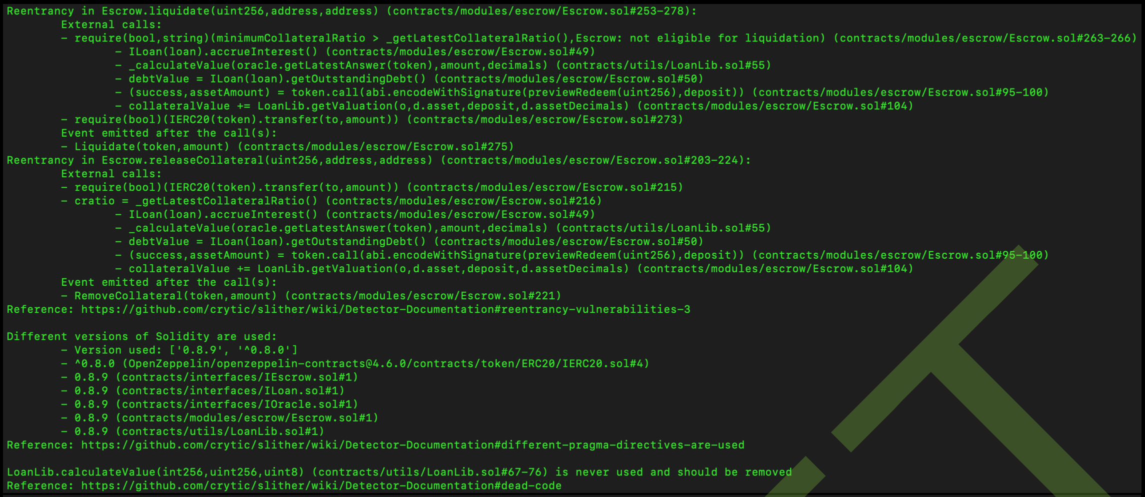
Escrow.sol



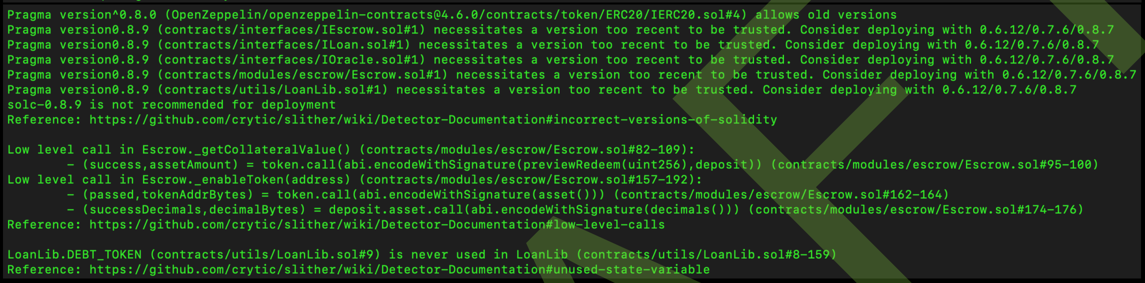
```
Reentrancy in Escrow._enableToken(address) (contracts/modules/escrow/Escrow.sol#157-192):
        External calls:
        - (passed,tokenAddrBytes) = token.call(abi.encodeWithSignature(asset())) (contracts/modules/escrow/Escrow.sol#162-164)
        - price = IOracle(oracle).getLatestAnswer(deposit.asset) (contracts/modules/escrow/Escrow.sol#171)
        - (successDecimals,decimalBytes) = deposit.asset.call(abi.encodeWithSignature(decimals())) (contracts/modules/escrow/Escrow.sol#174-176)
        State variables written after the call(s):
        - deposited[token] = deposit (contracts/modules/escrow/Escrow.sol#185)
        - enabled[token] = true (contracts/modules/escrow/Escrow.sol#184)
Reentrancy in Escrow.liquidate(uint256,address,address) (contracts/modules/escrow/Escrow.sol#253-278):
        External calls:
        - require(bool,string)(minimumCollateralRatio > _getLatestCollateralRatio(),Escrow: not eligible for liquidation) (contracts/modules/escrow/Escrow.sol#263-266)
                - ILoan(loan).accrueInterest() (contracts/modules/escrow/Escrow.sol#49)
                - _calculateValue(oracle.getLatestAnswer(token),amount,decimals) (contracts/utils/LoanLib.sol#55)
                - debtValue = ILoan(loan).getOutstandingDebt() (contracts/modules/escrow/Escrow.sol#50)
                - (success,assetAmount) = token.call(abi.encodeWithSignature(previewRedeem(uint256),deposit)) (contracts/modules/escrow/Escrow.sol#95-100)
                - collateralValue += LoanLib.getValuation(o,d.asset,deposit,d.assetDecimals) (contracts/modules/escrow/Escrow.sol#104)
        State variables written after the call(s):
        - deposited[token].amount -= amount (contracts/modules/escrow/Escrow.sol#272)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

Escrow._getLatestCollateralRatio() (contracts/modules/escrow/Escrow.sol#48-56) ignores return value by ILoan(loan).accrueInterest() (contracts/modules/escrow/Escrow.sol#49)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```



```
Escrow.constructor(uint256,address,address,address)._oracle (contracts/modules/escrow/Escrow.sol#33) lacks a zero-check on :
                - oracle = _oracle (contracts/modules/escrow/Escrow.sol#38)
Escrow.constructor(uint256,address,address,address)._loan (contracts/modules/escrow/Escrow.sol#34) lacks a zero-check on :
                - loan = _loan (contracts/modules/escrow/Escrow.sol#39)
Escrow.constructor(uint256,address,address,address)._borrower (contracts/modules/escrow/Escrow.sol#35) lacks a zero-check on :
                - borrower = _borrower (contracts/modules/escrow/Escrow.sol#40)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in Escrow._enableToken(address) (contracts/modules/escrow/Escrow.sol#157-192):
        External calls:
        - (passed,tokenAddrBytes) = token.call(abi.encodeWithSignature(asset())) (contracts/modules/escrow/Escrow.sol#162-164)
        - price = IOracle(oracle).getLatestAnswer(deposit.asset) (contracts/modules/escrow/Escrow.sol#171)
        - (successDecimals,decimalBytes) = deposit.asset.call(abi.encodeWithSignature(decimals())) (contracts/modules/escrow/Escrow.sol#174-176)
        State variables written after the call(s):
        - _collateralTokens.push(token) (contracts/modules/escrow/Escrow.sol#186)
Reentrancy in Escrow.addCollateral(uint256,address) (contracts/modules/escrow/Escrow.sol#120-134):
        External calls:
        - require(bool)(IERC20(token).transferFrom(msg.sender,address(this),amount)) (contracts/modules/escrow/Escrow.sol#127)
        State variables written after the call(s):
        - deposited[token].amount += amount (contracts/modules/escrow/Escrow.sol#129)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Reentrancy in Escrow._enableToken(address) (contracts/modules/escrow/Escrow.sol#157-192):
        External calls:
        - (passed,tokenAddrBytes) = token.call(abi.encodeWithSignature(asset())) (contracts/modules/escrow/Escrow.sol#162-164)
        - price = IOracle(oracle).getLatestAnswer(deposit.asset) (contracts/modules/escrow/Escrow.sol#171)
        - (successDecimals,decimalBytes) = deposit.asset.call(abi.encodeWithSignature(decimals())) (contracts/modules/escrow/Escrow.sol#174-176)
        Event emitted after the call(s):
        - EnableCollateral(deposit.asset,price) (contracts/modules/escrow/Escrow.sol#187)
Reentrancy in Escrow.addCollateral(uint256,address) (contracts/modules/escrow/Escrow.sol#120-134):
        External calls:
        - require(bool)(IERC20(token).transferFrom(msg.sender,address(this),amount)) (contracts/modules/escrow/Escrow.sol#127)
        Event emitted after the call(s):
        - AddCollateral(token,amount) (contracts/modules/escrow/Escrow.sol#131)
Reentrancy in Escrow.enableCollateral(address) (contracts/modules/escrow/Escrow.sol#143-149):
        External calls:
        - require(bool)(msg.sender == ILoan(loan).arbiter()) (contracts/modules/escrow/Escrow.sol#144)
        - _enableToken(token) (contracts/modules/escrow/Escrow.sol#146)
                - (passed,tokenAddrBytes) = token.call(abi.encodeWithSignature(asset())) (contracts/modules/escrow/Escrow.sol#162-164)
                - price = IOracle(oracle).getLatestAnswer(deposit.asset) (contracts/modules/escrow/Escrow.sol#171)
                - (successDecimals,decimalBytes) = deposit.asset.call(abi.encodeWithSignature(decimals())) (contracts/modules/escrow/Escrow.sol#174-176)
        Event emitted after the call(s):
        - EnableCollateral(deposit.asset,price) (contracts/modules/escrow/Escrow.sol#187)
                - _enableToken(token) (contracts/modules/escrow/Escrow.sol#146)
```

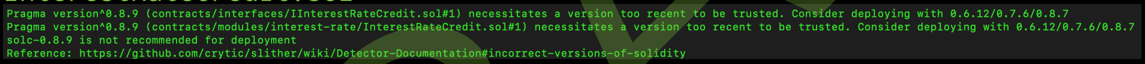


```
Reentrancy in Escrow.liquidate(uint256,address,address) (contracts/modules/escrow/Escrow.sol#253-278):
	External calls:
	- require(bool,string)(minimumCollateralRatio > _getLatestCollateralRatio(),Escrow: not eligible for liquidation) (contracts/modules/escrow/Escrow.sol#263-266)
		- ILoan(loan).accrueInterest() (contracts/modules/escrow/Escrow.sol#49)
		- _calculateValue(oracle.getLatestAnswer(token),amount,decimals) (contracts/utils/LoanLib.sol#55)
		- debtValue = ILoan(loan).getOutstandingDebt() (contracts/modules/escrow/Escrow.sol#50)
		- (success,assetAmount) = token.call(abi.encodeWithSignature(previewRedeem(uint256),deposit)) (contracts/modules/escrow/Escrow.sol#95-100)
		- collateralValue += LoanLib.getValuation(o,d.asset,deposit,d.assetDecimals) (contracts/modules/escrow/Escrow.sol#104)
	- require(bool)(IERC20(token).transfer(to,amount)) (contracts/modules/escrow/Escrow.sol#273)
	Event emitted after the call(s):
	- Liquidate(token,amount) (contracts/modules/escrow/Escrow.sol#275)
Reentrancy in Escrow.releaseCollateral(uint256,address,address) (contracts/modules/escrow/Escrow.sol#203-224):
	External calls:
	- require(bool)(IERC20(token).transfer(to,amount)) (contracts/modules/escrow/Escrow.sol#215)
	- cratio = _getLatestCollateralRatio() (contracts/modules/escrow/Escrow.sol#216)
		- ILoan(loan).accrueInterest() (contracts/modules/escrow/Escrow.sol#49)
		- _calculateValue(oracle.getLatestAnswer(token),amount,decimals) (contracts/utils/LoanLib.sol#55)
		- debtValue = ILoan(loan).getOutstandingDebt() (contracts/modules/escrow/Escrow.sol#50)
		- (success,assetAmount) = token.call(abi.encodeWithSignature(previewRedeem(uint256),deposit)) (contracts/modules/escrow/Escrow.sol#95-100)
		- collateralValue += LoanLib.getValuation(o,d.asset,deposit,d.assetDecimals) (contracts/modules/escrow/Escrow.sol#104)
	Event emitted after the call(s):
	- RemoveCollateral(token,amount) (contracts/modules/escrow/Escrow.sol#221)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Different versions of Solidity are used:
	- Version used: ['0.8.9', '^0.8.0']
	- ^0.8.0 (OpenZeppelin/openzeppelin-contracts@4.6.0/contracts/token/ERC20/IERC20.sol#4)
	- 0.8.9 (contracts/interfaces/IEscrow.sol#1)
	- 0.8.9 (contracts/interfaces/ILoan.sol#1)
	- 0.8.9 (contracts/interfaces/IOracle.sol#1)
	- 0.8.9 (contracts/modules/escrow/Escrow.sol#1)
	- 0.8.9 (contracts/utils/LoanLib.sol#1)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

LoanLib.calculateValue(int256,uint256,uint8) (contracts/utils/LoanLib.sol#67-76) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```



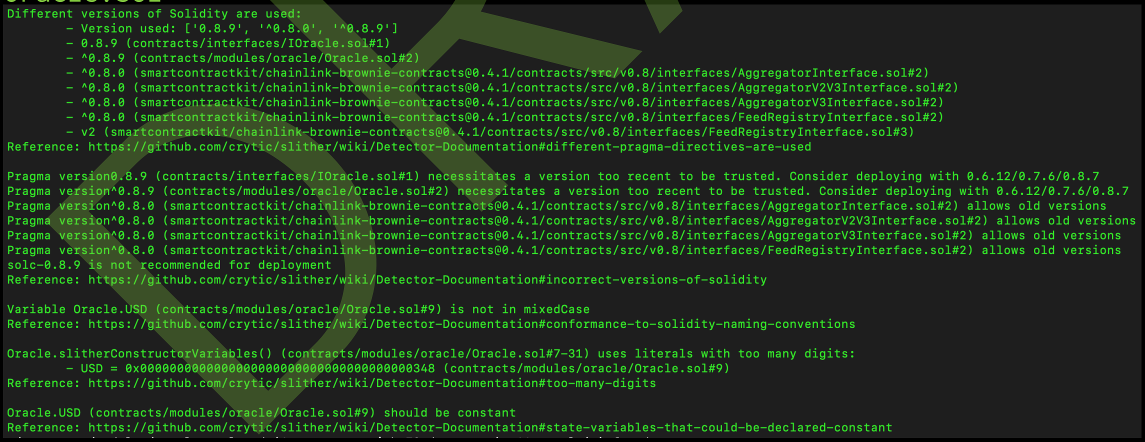
```
Pragma version^0.8.0 (OpenZeppelin/openzeppelin-contracts@4.6.0/contracts/token/ERC20/IERC20.sol#4) allows old versions
Pragma version0.8.9 (contracts/interfaces/IEscrow.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.9 (contracts/interfaces/ILoan.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.9 (contracts/interfaces/IOracle.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.9 (contracts/modules/escrow/Escrow.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.9 (contracts/utils/LoanLib.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Escrow._getCollateralValue() (contracts/modules/escrow/Escrow.sol#82-109):
	- (success,assetAmount) = token.call(abi.encodeWithSignature(previewRedeem(uint256),deposit)) (contracts/modules/escrow/Escrow.sol#95-100)
Low level call in Escrow._enableToken(address) (contracts/modules/escrow/Escrow.sol#157-192):
	- (passed,tokenAddrBytes) = token.call(abi.encodeWithSignature(asset())) (contracts/modules/escrow/Escrow.sol#162-164)
	- (successDecimals,decimalBytes) = deposit.asset.call(abi.encodeWithSignature(decimals())) (contracts/modules/escrow/Escrow.sol#174-176)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

LoanLib.DEBT_TOKEN (contracts/utils/LoanLib.sol#9) is never used in LoanLib (contracts/utils/LoanLib.sol#8-159)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable
```

InterestRateCredit.sol



```
Pragma version^0.8.9 (contracts/interfaces/IInterestRateCredit.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version^0.8.9 (contracts/modules/interest-rate/InterestRateCredit.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

Oracle.sol



```
Different versions of Solidity are used:
	- Version used: ['0.8.9', '^0.8.0', '^0.8.9']
	- 0.8.9 (contracts/interfaces/IOracle.sol#1)
	- ^0.8.9 (contracts/modules/oracle/Oracle.sol#2)
	- ^0.8.0 (smartcontractkit/chainlink-brownie-contracts@0.4.1/contracts/src/v0.8/interfaces/AggregatorInterface.sol#2)
	- ^0.8.0 (smartcontractkit/chainlink-brownie-contracts@0.4.1/contracts/src/v0.8/interfaces/AggregatorV2V3Interface.sol#2)
	- ^0.8.0 (smartcontractkit/chainlink-brownie-contracts@0.4.1/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol#2)
	- ^0.8.0 (smartcontractkit/chainlink-brownie-contracts@0.4.1/contracts/src/v0.8/interfaces/FeedRegistryInterface.sol#2)
	- v2 (smartcontractkit/chainlink-brownie-contracts@0.4.1/contracts/src/v0.8/interfaces/FeedRegistryInterface.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Pragma version0.8.9 (contracts/interfaces/IOracle.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version^0.8.9 (contracts/modules/oracle/Oracle.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version^0.8.0 (smartcontractkit/chainlink-brownie-contracts@0.4.1/contracts/src/v0.8/interfaces/AggregatorInterface.sol#2) allows old versions
Pragma version^0.8.0 (smartcontractkit/chainlink-brownie-contracts@0.4.1/contracts/src/v0.8/interfaces/AggregatorV2V3Interface.sol#2) allows old versions
Pragma version^0.8.0 (smartcontractkit/chainlink-brownie-contracts@0.4.1/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol#2) allows old versions
Pragma version^0.8.0 (smartcontractkit/chainlink-brownie-contracts@0.4.1/contracts/src/v0.8/interfaces/FeedRegistryInterface.sol#2) allows old versions
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Variable Oracle.USD (contracts/modules/oracle/Oracle.sol#9) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Oracle.slitherConstructorVariables() (contracts/modules/oracle/Oracle.sol#7-31) uses literals with too many digits:
	- USD = 0x0000000000000000000000000000000000000348 (contracts/modules/oracle/Oracle.sol#9)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

Oracle.USD (contracts/modules/oracle/Oracle.sol#9) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
```

- The usage of solc-0.8.9 is false-positive.
- The different version of Solidity usage instances are false-positives.
- The unused code was reported in UNUSED CODE finding.
- The _getNonCaller function is used in mutualConsent modifier.

- The reentrancy issues in EscrowedLoan, LineOfCredit are false-positives.
- Two instances of possible reentrancy issues are reported in LACK OF CHECK EFFECTS INTERACTIONS PATTERN OR REENTRENCY GUARD finding.
- The sending ether to arbitrary user instances are false-positives.
- The ignoring return value instances are false-positives.
- The low-level call instances are false-positives.
- The external calls inside a loop instances are false-positives.
- The usage of dangerous strict equality instances are false positives.
- The usage of literals with too many digits findings are false positives.
- The lacks of zero-check of contract address is considered no more as an issue. Since Solidity, 0.5.0 msg.data length is being checked.
- The some of identified issues are related to OppenZepplin's libraries.
- The scans of LineOfCredit.sol, SpigotedLoan.sol and Spigot.sol were included in scan of SecuredLoan.sol.
- No major issues were found by Slither.

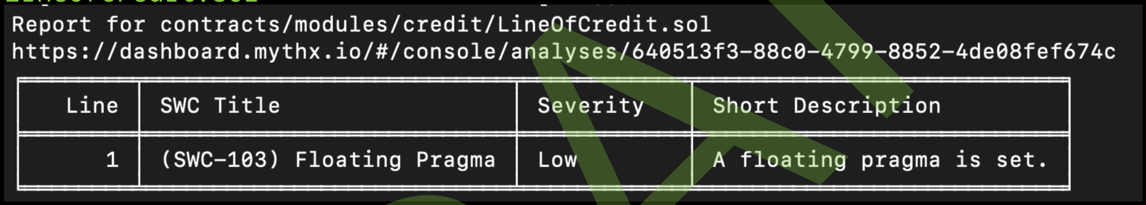
4.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruits on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on all the contracts and sent the compiled results to the analyzers to locate any vulnerabilities.

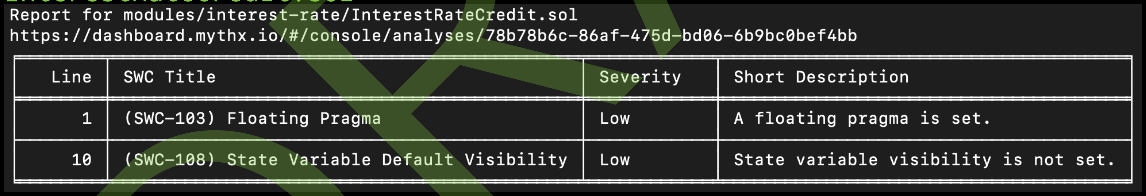
MythX results:

LineOfCredit.sol



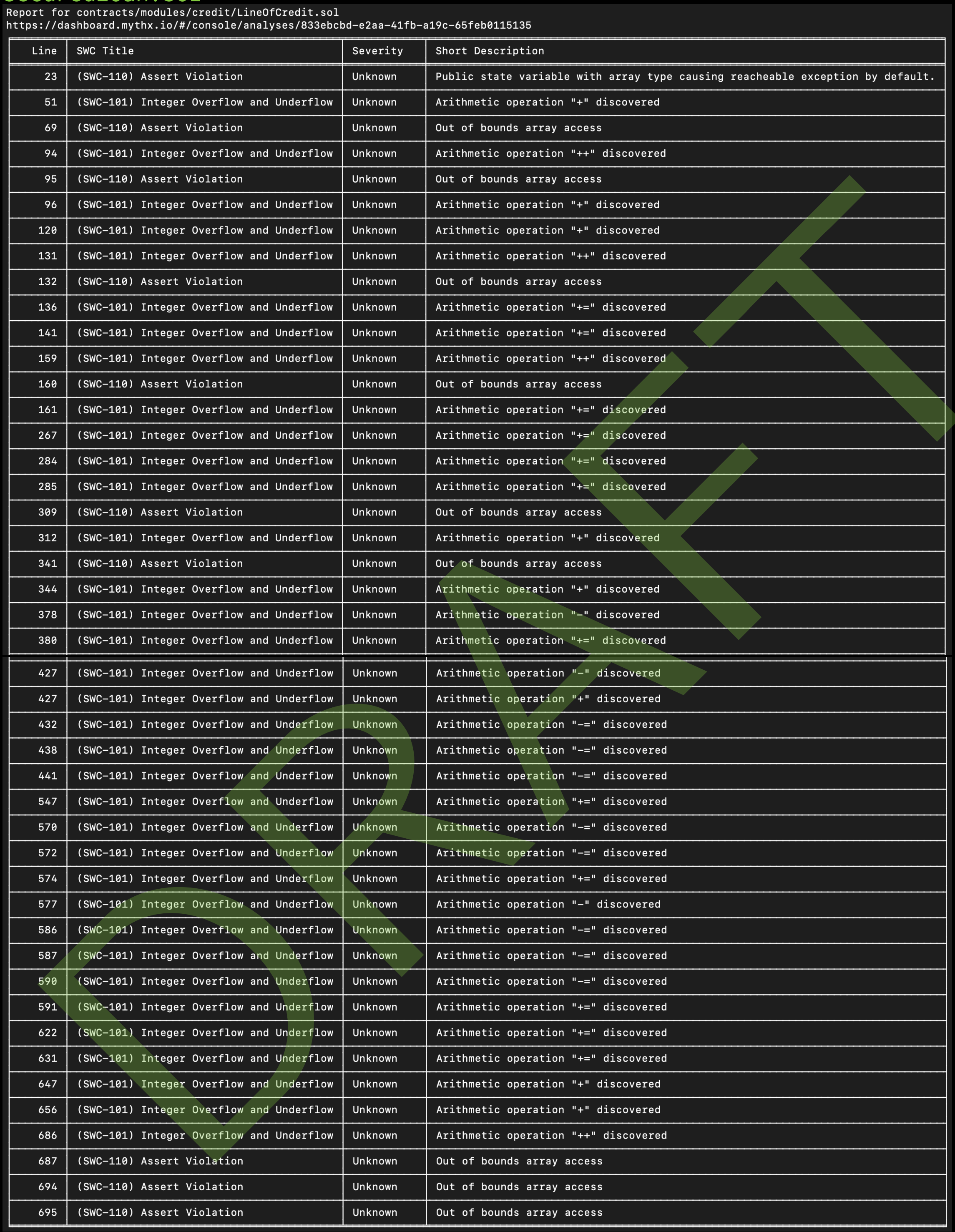
```
Report for contracts/modules/credit/LineOfCredit.sol
https://dashboard.mythx.io/#/console/analyses/640513f3-88c0-4799-8852-4de08fef674c

| Line | SWC Title                  | Severity | Short Description         |
|    1 | (SWC-103) Floating Pragma  | Low      | A floating pragma is set. |
```

InterestRateCredit.sol

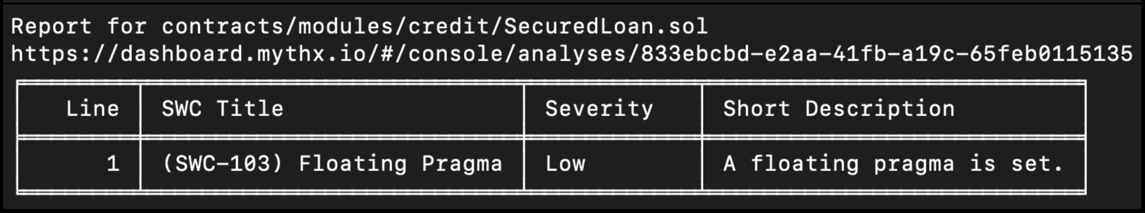


```
Report for modules/interest-rate/InterestRateCredit.sol
https://dashboard.mythx.io/#/console/analyses/78b78b6c-86af-475d-bd06-6b9bc0bef4bb

| Line | SWC Title                                  | Severity | Short Description                      |
|    1 | (SWC-103) Floating Pragma                  | Low      | A floating pragma is set.              |
|   10 | (SWC-108) State Variable Default Visibility | Low      | State variable visibility is not set.  |
```

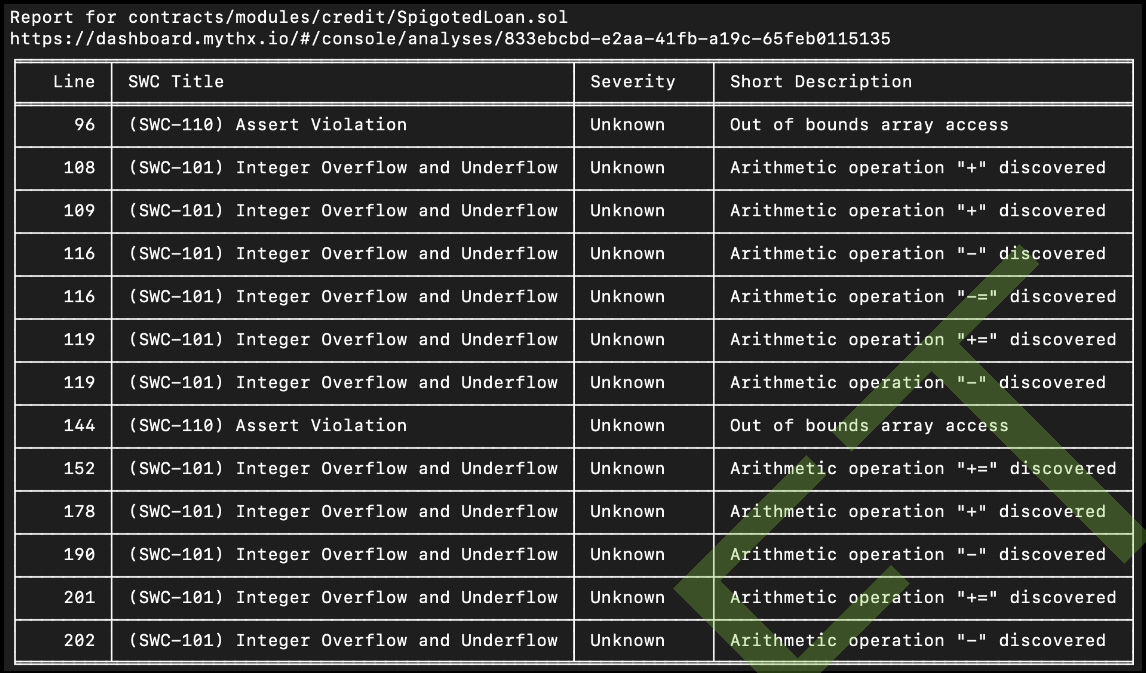
SecuredLoan.sol



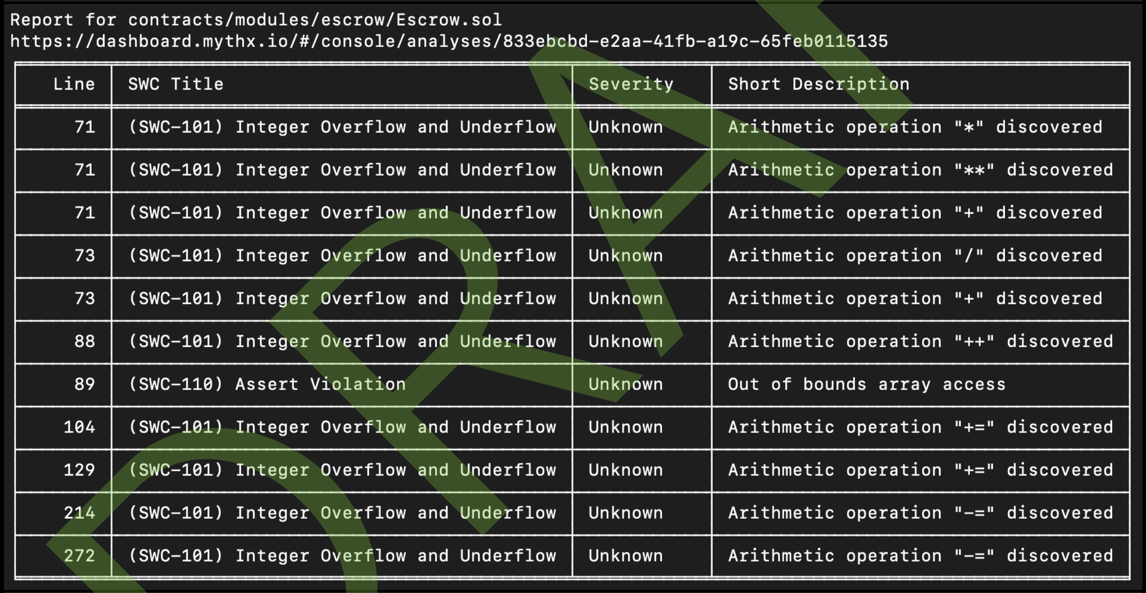
```
Report for contracts/modules/credit/LineOfCredit.sol
https://dashboard.mythx.io/#/console/analyses/833ebcbd-e2aa-41fb-a19c-65feb0115135

| Line | SWC Title                                  | Severity | Short Description                                                          |
|------|--------------------------------------------|----------|----------------------------------------------------------------------------|
|   23 | (SWC-110) Assert Violation                 | Unknown  | Public state variable with array type causing reacheable exception by default. |
|   51 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "+" discovered                                        |
|   69 | (SWC-110) Assert Violation                 | Unknown  | Out of bounds array access                                                 |
|   94 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "++" discovered                                       |
|   95 | (SWC-110) Assert Violation                 | Unknown  | Out of bounds array access                                                 |
|   96 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "+" discovered                                        |
|  120 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "+" discovered                                        |
|  131 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "++" discovered                                       |
|  132 | (SWC-110) Assert Violation                 | Unknown  | Out of bounds array access                                                 |
|  136 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "+=" discovered                                       |
|  141 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "+=" discovered                                       |
|  159 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "++" discovered                                       |
|  160 | (SWC-110) Assert Violation                 | Unknown  | Out of bounds array access                                                 |
|  161 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "+=" discovered                                       |
|  267 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "+=" discovered                                       |
|  284 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "+=" discovered                                       |
|  285 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "+=" discovered                                       |
|  309 | (SWC-110) Assert Violation                 | Unknown  | Out of bounds array access                                                 |
|  312 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "+" discovered                                        |
|  341 | (SWC-110) Assert Violation                 | Unknown  | Out of bounds array access                                                 |
|  344 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "+" discovered                                        |
|  378 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "-" discovered                                        |
|  380 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "+=" discovered                                       |
|  427 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "-" discovered                                        |
|  427 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "+" discovered                                        |
|  432 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "-=" discovered                                       |
|  438 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "-=" discovered                                       |
|  441 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "-=" discovered                                       |
|  547 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "+=" discovered                                       |
|  570 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "-=" discovered                                       |
|  572 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "-=" discovered                                       |
|  574 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "+=" discovered                                       |
|  577 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "-" discovered                                        |
|  586 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "-=" discovered                                       |
|  587 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "-=" discovered                                       |
|  590 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "-=" discovered                                       |
|  591 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "+=" discovered                                       |
|  622 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "+=" discovered                                       |
|  631 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "+=" discovered                                       |
|  647 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "+" discovered                                        |
|  656 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "+" discovered                                        |
|  686 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "++" discovered                                       |
|  687 | (SWC-110) Assert Violation                 | Unknown  | Out of bounds array access                                                 |
|  694 | (SWC-110) Assert Violation                 | Unknown  | Out of bounds array access                                                 |
|  695 | (SWC-110) Assert Violation                 | Unknown  | Out of bounds array access                                                 |
```



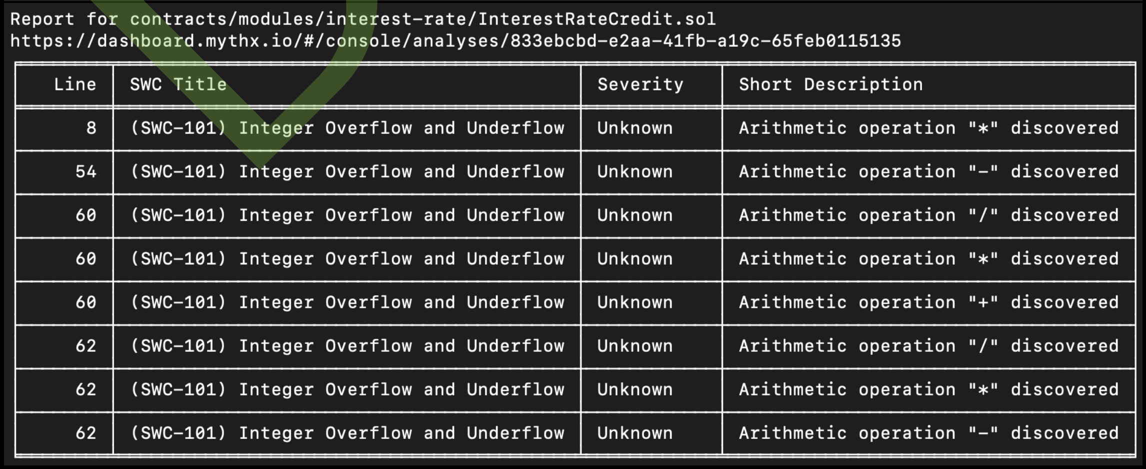
```
Report for contracts/modules/credit/SecuredLoan.sol
https://dashboard.mythx.io/#/console/analyses/833ebcbd-e2aa-41fb-a19c-65feb0115135

| Line | SWC Title                   | Severity | Short Description          |
|------|-----------------------------|----------|----------------------------|
|    1 | (SWC-103) Floating Pragma   | Low      | A floating pragma is set.  |
```



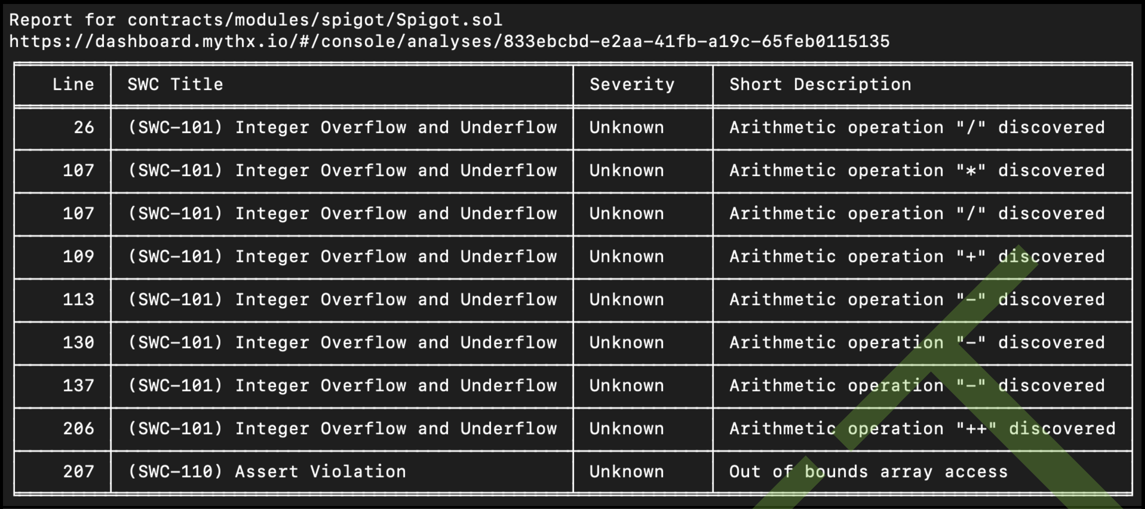
```
Report for contracts/modules/credit/SpigotedLoan.sol
https://dashboard.mythx.io/#/console/analyses/833ebcbd-e2aa-41fb-a19c-65feb0115135

| Line | SWC Title                                 | Severity | Short Description                  |
|   96 | (SWC-110) Assert Violation                | Unknown  | Out of bounds array access         |
|  108 | (SWC-101) Integer Overflow and Underflow  | Unknown  | Arithmetic operation "+" discovered  |
|  109 | (SWC-101) Integer Overflow and Underflow  | Unknown  | Arithmetic operation "+" discovered  |
|  116 | (SWC-101) Integer Overflow and Underflow  | Unknown  | Arithmetic operation "-" discovered  |
|  116 | (SWC-101) Integer Overflow and Underflow  | Unknown  | Arithmetic operation "-=" discovered |
|  119 | (SWC-101) Integer Overflow and Underflow  | Unknown  | Arithmetic operation "+=" discovered |
|  119 | (SWC-101) Integer Overflow and Underflow  | Unknown  | Arithmetic operation "-" discovered  |
|  144 | (SWC-110) Assert Violation                | Unknown  | Out of bounds array access         |
|  152 | (SWC-101) Integer Overflow and Underflow  | Unknown  | Arithmetic operation "+=" discovered |
|  178 | (SWC-101) Integer Overflow and Underflow  | Unknown  | Arithmetic operation "+" discovered  |
|  190 | (SWC-101) Integer Overflow and Underflow  | Unknown  | Arithmetic operation "-" discovered  |
|  201 | (SWC-101) Integer Overflow and Underflow  | Unknown  | Arithmetic operation "+=" discovered |
|  202 | (SWC-101) Integer Overflow and Underflow  | Unknown  | Arithmetic operation "-" discovered  |
```



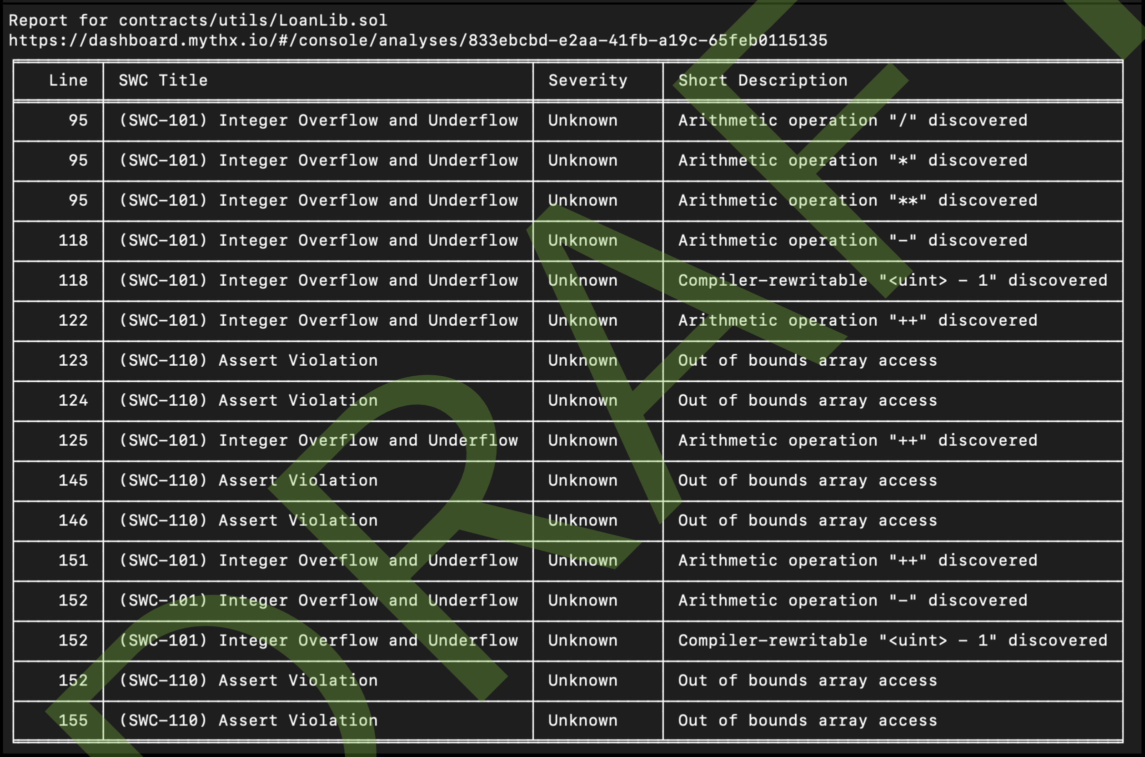
```
Report for contracts/modules/escrow/Escrow.sol
https://dashboard.mythx.io/#/console/analyses/833ebcbd-e2aa-41fb-a19c-65feb0115135

| Line | SWC Title                                 | Severity | Short Description                  |
|   71 | (SWC-101) Integer Overflow and Underflow  | Unknown  | Arithmetic operation "*" discovered  |
|   71 | (SWC-101) Integer Overflow and Underflow  | Unknown  | Arithmetic operation "**" discovered |
|   71 | (SWC-101) Integer Overflow and Underflow  | Unknown  | Arithmetic operation "+" discovered  |
|   73 | (SWC-101) Integer Overflow and Underflow  | Unknown  | Arithmetic operation "/" discovered  |
|   73 | (SWC-101) Integer Overflow and Underflow  | Unknown  | Arithmetic operation "+" discovered  |
|   88 | (SWC-101) Integer Overflow and Underflow  | Unknown  | Arithmetic operation "++" discovered |
|   89 | (SWC-110) Assert Violation                | Unknown  | Out of bounds array access         |
|  104 | (SWC-101) Integer Overflow and Underflow  | Unknown  | Arithmetic operation "+=" discovered |
|  129 | (SWC-101) Integer Overflow and Underflow  | Unknown  | Arithmetic operation "+=" discovered |
|  214 | (SWC-101) Integer Overflow and Underflow  | Unknown  | Arithmetic operation "-=" discovered |
|  272 | (SWC-101) Integer Overflow and Underflow  | Unknown  | Arithmetic operation "-=" discovered |
```



```
Report for contracts/modules/interest-rate/InterestRateCredit.sol
https://dashboard.mythx.io/#/console/analyses/833ebcbd-e2aa-41fb-a19c-65feb0115135

| Line | SWC Title                                 | Severity | Short Description                  |
|    8 | (SWC-101) Integer Overflow and Underflow  | Unknown  | Arithmetic operation "*" discovered  |
|   54 | (SWC-101) Integer Overflow and Underflow  | Unknown  | Arithmetic operation "-" discovered  |
|   60 | (SWC-101) Integer Overflow and Underflow  | Unknown  | Arithmetic operation "/" discovered  |
|   60 | (SWC-101) Integer Overflow and Underflow  | Unknown  | Arithmetic operation "*" discovered  |
|   60 | (SWC-101) Integer Overflow and Underflow  | Unknown  | Arithmetic operation "+" discovered  |
|   62 | (SWC-101) Integer Overflow and Underflow  | Unknown  | Arithmetic operation "/" discovered  |
|   62 | (SWC-101) Integer Overflow and Underflow  | Unknown  | Arithmetic operation "*" discovered  |
|   62 | (SWC-101) Integer Overflow and Underflow  | Unknown  | Arithmetic operation "-" discovered  |
```

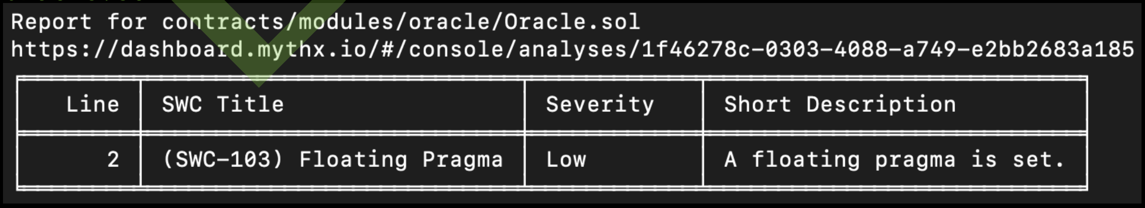


```
Report for contracts/modules/spigot/Spigot.sol
https://dashboard.mythx.io/#/console/analyses/833ebcbd-e2aa-41fb-a19c-65feb0115135

| Line | SWC Title                                | Severity | Short Description                      |
|------|------------------------------------------|----------|----------------------------------------|
|   26 | (SWC-101) Integer Overflow and Underflow | Unknown  | Arithmetic operation "/" discovered    |
|  107 | (SWC-101) Integer Overflow and Underflow | Unknown  | Arithmetic operation "*" discovered    |
|  107 | (SWC-101) Integer Overflow and Underflow | Unknown  | Arithmetic operation "/" discovered    |
|  109 | (SWC-101) Integer Overflow and Underflow | Unknown  | Arithmetic operation "+" discovered    |
|  113 | (SWC-101) Integer Overflow and Underflow | Unknown  | Arithmetic operation "-" discovered    |
|  130 | (SWC-101) Integer Overflow and Underflow | Unknown  | Arithmetic operation "-" discovered    |
|  137 | (SWC-101) Integer Overflow and Underflow | Unknown  | Arithmetic operation "-" discovered    |
|  206 | (SWC-101) Integer Overflow and Underflow | Unknown  | Arithmetic operation "++" discovered   |
|  207 | (SWC-110) Assert Violation               | Unknown  | Out of bounds array access             |
```



```
Report for contracts/utils/LoanLib.sol
https://dashboard.mythx.io/#/console/analyses/833ebcbd-e2aa-41fb-a19c-65feb0115135

| Line | SWC Title                                | Severity | Short Description                             |
|------|------------------------------------------|----------|-----------------------------------------------|
|   95 | (SWC-101) Integer Overflow and Underflow | Unknown  | Arithmetic operation "/" discovered           |
|   95 | (SWC-101) Integer Overflow and Underflow | Unknown  | Arithmetic operation "*" discovered           |
|   95 | (SWC-101) Integer Overflow and Underflow | Unknown  | Arithmetic operation "**" discovered          |
|  118 | (SWC-101) Integer Overflow and Underflow | Unknown  | Arithmetic operation "-" discovered           |
|  118 | (SWC-101) Integer Overflow and Underflow | Unknown  | Compiler-rewritable "<uint> - 1" discovered   |
|  122 | (SWC-101) Integer Overflow and Underflow | Unknown  | Arithmetic operation "++" discovered          |
|  123 | (SWC-110) Assert Violation               | Unknown  | Out of bounds array access                    |
|  124 | (SWC-110) Assert Violation               | Unknown  | Out of bounds array access                    |
|  125 | (SWC-101) Integer Overflow and Underflow | Unknown  | Arithmetic operation "++" discovered          |
|  145 | (SWC-110) Assert Violation               | Unknown  | Out of bounds array access                    |
|  146 | (SWC-110) Assert Violation               | Unknown  | Out of bounds array access                    |
|  151 | (SWC-101) Integer Overflow and Underflow | Unknown  | Arithmetic operation "++" discovered          |
|  152 | (SWC-101) Integer Overflow and Underflow | Unknown  | Arithmetic operation "-" discovered           |
|  152 | (SWC-101) Integer Overflow and Underflow | Unknown  | Compiler-rewritable "<uint> - 1" discovered   |
|  152 | (SWC-110) Assert Violation               | Unknown  | Out of bounds array access                    |
|  155 | (SWC-110) Assert Violation               | Unknown  | Out of bounds array access                    |
```

Oracle.sol



```
Report for contracts/modules/oracle/Oracle.sol
https://dashboard.mythx.io/#/console/analyses/1f46278c-0303-4088-a749-e2bb2683a185

| Line | SWC Title                  | Severity | Short Description           |
|------|----------------------------|----------|-----------------------------|
|    2 | (SWC-103) Floating Pragma  | Low      | A floating pragma is set.   |
```

- The floating pragma findings are false positives.
- The state variable default visibility instance is false positive.
- The Integer Overflow and Underflow findings are false positives.

- The scan of EscrowedLoan.sol, Escrow.sol, Loanlib.sol, and MutualConsent.sol yielded not results.
- The scans of LineOfCredit.sol, SpigotedLoan.sol, and Spigot.sol were included in the scan of SecuredLoan.sol.
- No major issues were discovered by Mythx software.

THANK YOU FOR CHOOSING

// HALBORN