CYBER UNIT
Cyber Security
Strategic Partner

www.cyberunit.tech

Impermax Core Smart Contract Audit

Date: January 18, 2020 – Updated February 8, 2021
Report for: Impermax Finance
By: CyberUnit.Tech

This document may contain confidential information about IT systems and the customer's intellectual property and information about potential vulnerabilities and exploitation methods.
The report contains confidential information. This information can be used internally by the customer. The customer can release the information after fixing all vulnerabilities.

Document

Name	Impermax Core
Platform	EVM
Link	https://github.com/Impermax-Finance/impermax-x-uniswapv2-core/commit/e899cd3354d25c717b3261fc963ce2053e857a66
Date	15/01/21

Table of contents

Introduction

This report presents the Customer` s smart contract's security assessment findings and its code review conducted between January 4 to January 15.

Scope

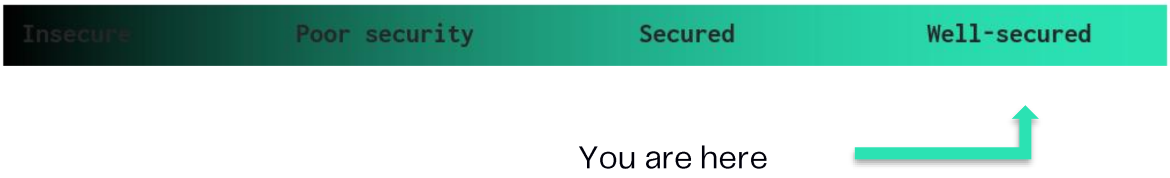
The scope of the project is Impermax-x-uniswap, Impermax core smart contracts.

We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the widely known vulnerabilities that are considered (the full list includes them but is not limited by them):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level

Executive Summary

According to the assessment, Customer' smart contracts are secured.



Our team performed an analysis of code functionality, manual audit, and automated checks with Slither and remix IDE (see Appendix B pic 1-2). All issues found during automated analysis were manually reviewed, and application vulnerabilities are presented in the Audit overview section. A general overview is shown in the AS-IS section, and all found issues can be found in the Audit overview section.

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to tokens loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also significantly not impact smart contract execution, e.g., public access to crucial functions.
Medium	Medium-level vulnerabilities are essential to fix; however, they can't lead to tokens loss.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc., code snippets that can't significantly impact execution.
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

AS-IS overview for Borrowable

Impermax x Uniswap V2 contract consists of the following smart contracts:

1. **Math.sol, SafeMath.sol, UQ112x112.sol** contracts - supporting libraries

2. **IBDeployer.sol, IBorrowTracker.sol, IBorrowable.sol, ICDeployer.sol, ICollateral.sol, IERC20.sol, IFactory.sol, IImpermaxCallee.sol, IPoolToken.sol, ISimpleUniswapOracle.sol, IUniswapV2Factory.sol, IUniswapV2Pair.sol** contracts - interfaces

3. **Impermax** contracts - **BAllowance.sol, BDeployer.sol, BInterestRateModel.sol, Borrowable.sol, BSetter.sol, BStorage.sol, CDeployer.sol, Collateral.sol, CSetter.sol, CStorage.sol, ImpermaxERC20.sol, PoolToken.sol, Factory.sol**

Contracts from point 1 were compared to original "Openzeppelin" and "Uniswap-v2-core" templates. No logic differences were found. They are considered secure.
Contracts from point 2 Impermax Interfaces - describe the actions that an object can perform.
Contracts from point 3 The Impermax classes implementing the "Impermax x Uniswap V2" protocol will be detailed in the report.

BDeployer: This contract is to deploy Borrowable.

BDeployer contract inherits interface IBDeployer and class **Borrowable**

deployBorrowable function was called with the following parameters:

- address (uniswapV2Pair)
- index (uint8)
- **latestVersion** was set to the moment of review.

CDeployer: This contract is to deploy Collateral.

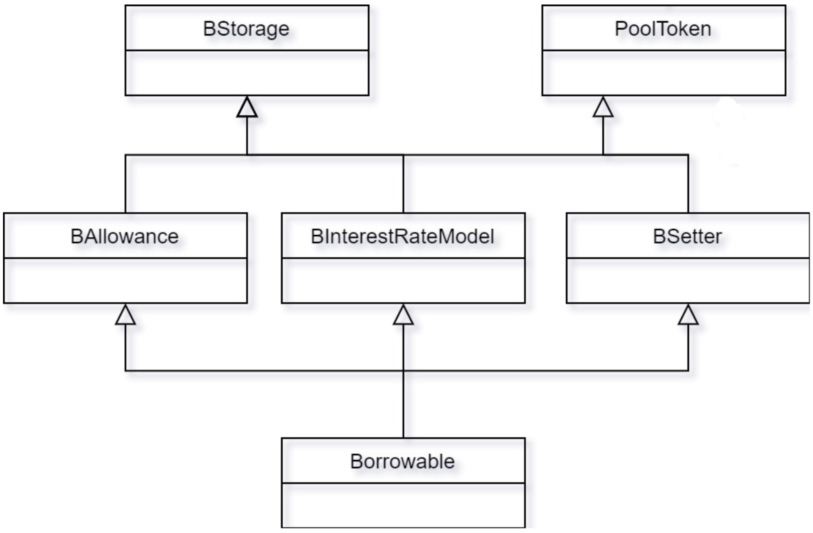
BDeployer contract inherits interface ICDeployer and class – **Collateral**

deployCollateral function was called with the following parameters:

- address (uniswapV2Pair)
- **latestVersion** was set to the moment of review.

BStorage and **CStorage**: Auxiliary contacts are helpers describing constants.

Borrowable:



Borrowable contract inherits the IBorrowable interface and the classes – PoolToken, BStorage, BSetter, BInterestRateModel, BAllowance

exchangeRate function was called with no following parameters

borrowBalance function was called with the following parameters:

- address(borrower)

That is the stored borrow balance.

borrow function was called with the following parameters:

- address(borrower)
- address(receiver)
- uint(borrowAmount)
- bytes(data)

liquidate function was called with the following parameters:

- address(borrower)

- address(liquidator)
- uint(declaredRepayAmount)
- bytes(data)

trackBorrow function was called with the following parameters:

- address(borrower)

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

Audit Borrowable overview

Critical

No critical severity vulnerabilities were found.

High

No high severity vulnerabilities were found.

Medium [Fixed]

1. ~~Divide before multiply (see Appendix A pic. 3 for evidence). Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.~~
2. ~~Borrowable Contract Failed Automatic Testing (see Appendix B pic. 5 for evidence). It is important to investigate the reasons for failed tests.~~

Low [Fixed]

3. ~~Reentrancy vulnerability is used in the redeem function (see Appendix A pic. 1 for evidence). Events may appear in the wrong order, which can create problems for third parties.~~
4. ~~Different versions of Solidity are used in Version used: ['=0.5.16', '>=0.5.0'] (see Appendix A pic. 2 for evidence)~~

Conclusion Borrowable

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. For the contract, a high-level description of functionality was presented in the report's As-is overview section.

The audit report contains all found security vulnerabilities and other issues in the reviewed code. Note that automatic testing of the Borrowable contract through the BorrowableHarness adapter fails.

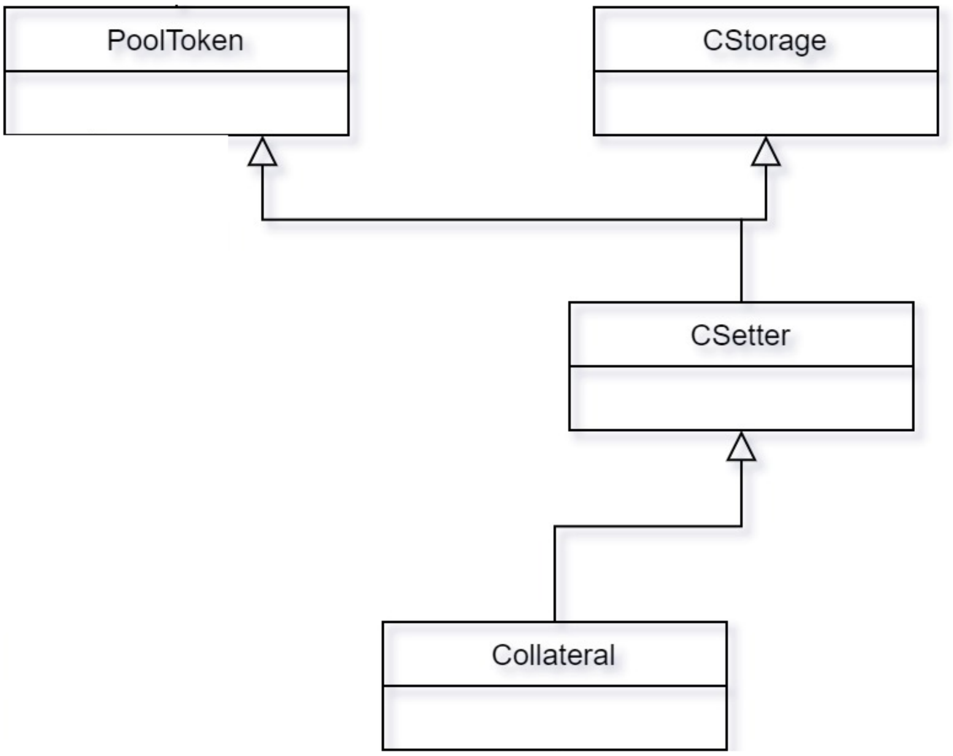
Security engineers found two medium and two low vulnerabilities.

AS-IS overview for Collateral:

This contract is basically UniswapV2ERC20 with small modifications.

Contract was compared to the original "Uniswap-v2-core" no logic differences were found. It is considered secure.

Collateral:



Collateral contract inherits the ICollateral interface and the classes - PoolToken, CStorage, CSetter and using UQ112x112

getPrices function was called with the following parameters:

- uint(price0)
- uint(price)

tokensUnlocked function was called with the following parametersaddress(from)

- uint(value)

accountLiquidityAmounts function was called with the following parameters:

- address(from)
- uint(value)

accountLiquidity function was called with the following parameters:

- address(borrower)

canBorrow function was called with the following parameters:

- address(borrower)
- address(borrowable)
- uint(accountBorrows)

seize function was called with the following parameters:

- address(liquidator)

- address(borrowable)
- uint(repayAmount)

- **flashRedeem** function was called with the following parameters:
- address(redeemer)
- uint(redeemAmount)
- bytes(data)

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

Audit Collateral overview

Critical

No critical severity vulnerabilities were found.

High

No high severity vulnerabilities were found.

Medium [Fixed]

1. ~~Divide before multiply (see Appendix A pic. 3 for evidence). Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.~~
2. ~~Collateral Contract Failed Automatic Testing (11 passing) (6 failings) (see Appendix B pic. 6 for evidence).~~

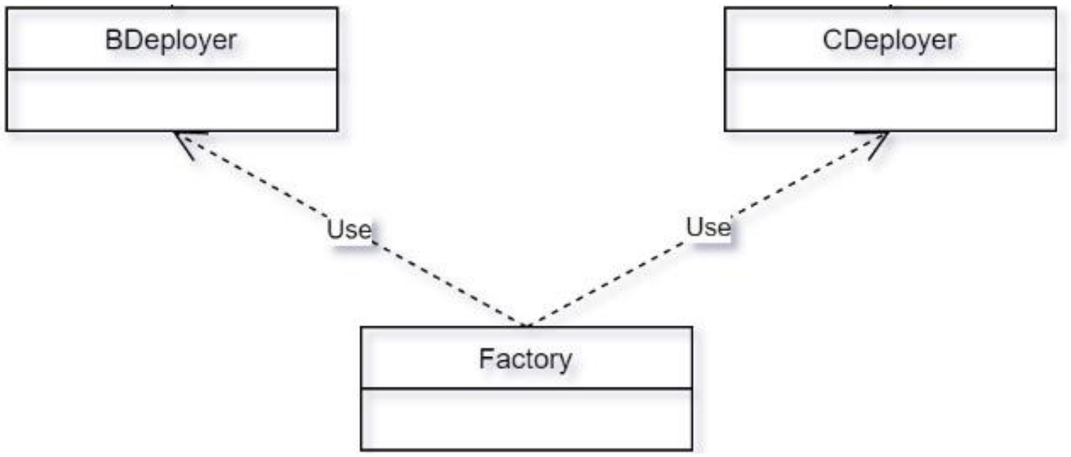
Low [Fixed]

3. ~~Different versions of Solidity are used in Version used: ['=0.5.16', '>=0.5.0'] (see Appendix A pic. 4 for evidence)~~

Conclusion Collateral

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. For the contract, a high-level description of functionality was presented in the report's As-is overview section. Note that automatic testing of the Collateral contract parts of tests ends fail.

AS-IS overview for Factory:



Factory contract inherits the IFactory interface.

PrestakingProvisioner contract **init** function was called with the following parameters:

- address(_admin)
- IBDeployer(_bDeployer)
- ICDeployer(_cDeployer)
- IUniswapV2Factory(_uniswapV2Factory)
- ISimpleUniswapOracle(_simpleUniswapOracle)

createCollateral function was called with the following parameters:

- address(uniswapV2Pair)

createBorrowable0 function was called with the following parameters:

- address(uniswapV2Pair)

createBorrowable1 function was called with the following parameters:

- address(uniswapV2Pair)

initializeLendingPool function was called with the following parameters:

- address(uniswapV2Pair)

Audit Factory overview

Critical

No critical severity vulnerabilities were found.

High

No high severity vulnerabilities were found.

Medium

No medium severity vulnerabilities were found.

Low

1. Different versions of Solidity are used in Version used: ['=0.5.16', '>=0.5.0'] (see Appendix A pic. 2 for evidence)

Disclaimers

Disclaimer

The smart contracts given for audit had been analyzed following the best industry practices at the date of this report, concerning: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

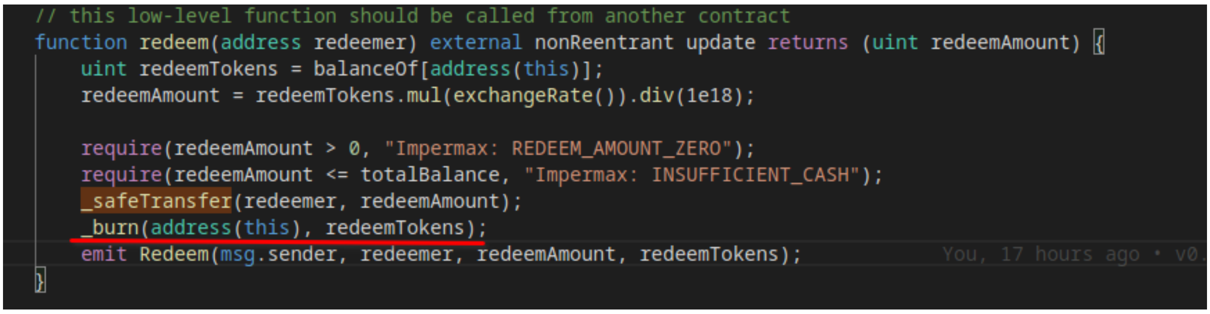
The audit makes no statements or warranties on the security of the code. It can also not be considered a sufficient assessment regarding the code's utility and safety, bug-free status, or any other contract statements. While we have done our best to conduct the analysis and produce this report, it is important to note that you should not rely on this report only – we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, programming language, and other software related to the smart contract can have their vulnerabilities leading to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.

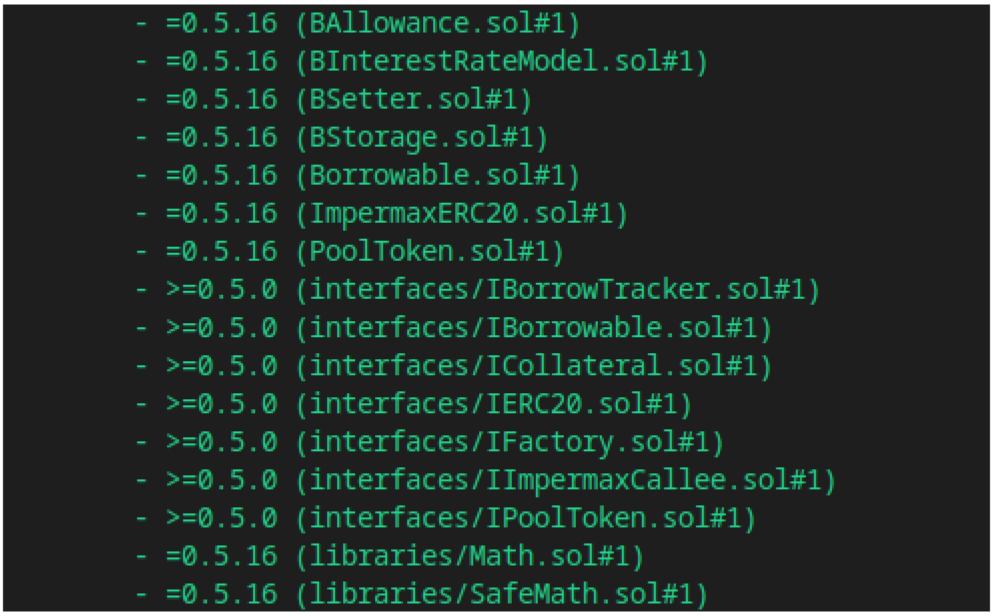
Appendix A. Automated tools report

Pic 1. Reentrancy Vulnerabilities:



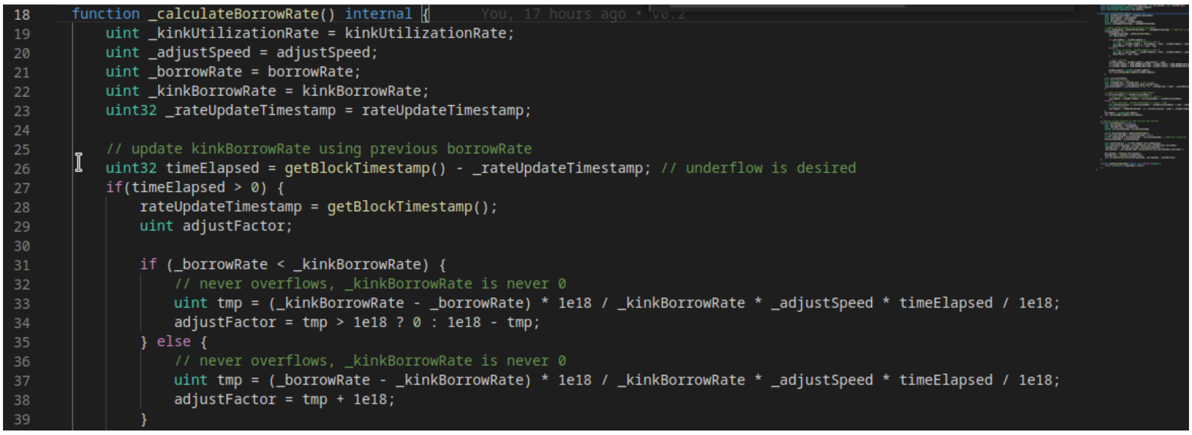
```
// this low-level function should be called from another contract
function redeem(address redeemer) external nonReentrant update returns (uint redeemAmount) {
    uint redeemTokens = balanceOf[address(this)];
    redeemAmount = redeemTokens.mul(exchangeRate()).div(1e18);

    require(redeemAmount > 0, "Impermax: REDEEM_AMOUNT_ZERO");
    require(redeemAmount <= totalBalance, "Impermax: INSUFFICIENT_CASH");
    _safeTransfer(redeemer, redeemAmount);
    _burn(address(this), redeemTokens);
    emit Redeem(msg.sender, redeemer, redeemAmount, redeemTokens);          You, 17 hours ago • v0.
}
```

Pic 2. Different pragma directives are used:



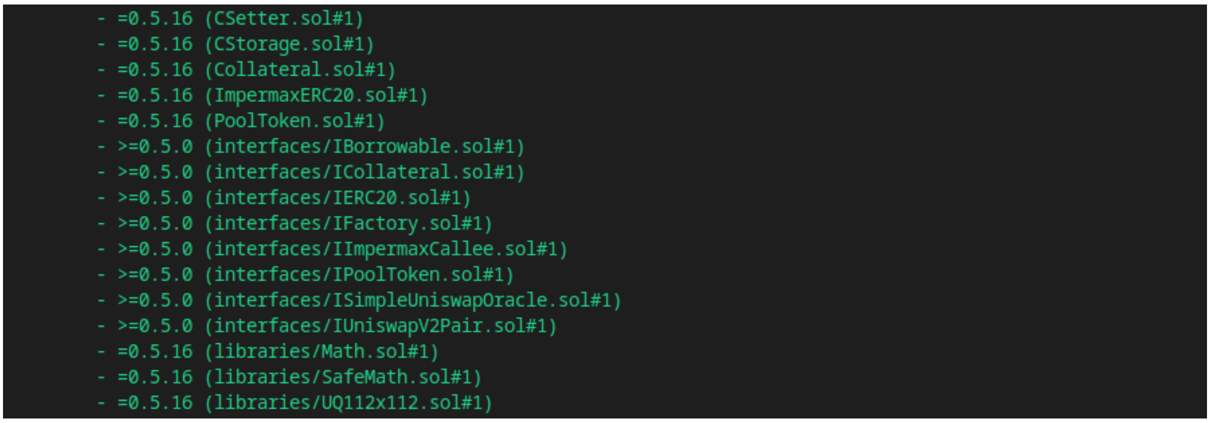
```
        - =0.5.16 (BAllowance.sol#1)
        - =0.5.16 (BInterestRateModel.sol#1)
        - =0.5.16 (BSetter.sol#1)
        - =0.5.16 (BStorage.sol#1)
        - =0.5.16 (Borrowable.sol#1)
        - =0.5.16 (ImpermaxERC20.sol#1)
        - =0.5.16 (PoolToken.sol#1)
        - >=0.5.0 (interfaces/IBorrowTracker.sol#1)
        - >=0.5.0 (interfaces/IBorrowable.sol#1)
        - >=0.5.0 (interfaces/ICollateral.sol#1)
        - >=0.5.0 (interfaces/IERC20.sol#1)
        - >=0.5.0 (interfaces/IFactory.sol#1)
        - >=0.5.0 (interfaces/IImpermaxCallee.sol#1)
        - >=0.5.0 (interfaces/IPoolToken.sol#1)
        - =0.5.16 (libraries/Math.sol#1)
        - =0.5.16 (libraries/SafeMath.sol#1)
```

Pic 3. Divide before multiply:



```
function _calculateBorrowRate() internal {      You, 17 hours ago • v0.2
    uint _kinkUtilizationRate = kinkUtilizationRate;
    uint _adjustSpeed = adjustSpeed;
    uint _borrowRate = borrowRate;
    uint _kinkBorrowRate = kinkBorrowRate;
    uint32 _rateUpdateTimestamp = rateUpdateTimestamp;

    // update kinkBorrowRate using previous borrowRate
    uint32 timeElapsed = getBlockTimestamp() - _rateUpdateTimestamp; // underflow is desired
    if(timeElapsed > 0) {
        rateUpdateTimestamp = getBlockTimestamp();
        uint adjustFactor;

        if (_borrowRate < _kinkBorrowRate) {
            // never overflows, _kinkBorrowRate is never 0
            uint tmp = (_kinkBorrowRate - _borrowRate) * 1e18 / _kinkBorrowRate * _adjustSpeed * timeElapsed / 1e18;
            adjustFactor = tmp > 1e18 ? 0 : 1e18 - tmp;
        } else {
            // never overflows, _kinkBorrowRate is never 0
            uint tmp = (_borrowRate - _kinkBorrowRate) * 1e18 / _kinkBorrowRate * _adjustSpeed * timeElapsed / 1e18;
            adjustFactor = tmp + 1e18;
        }
```

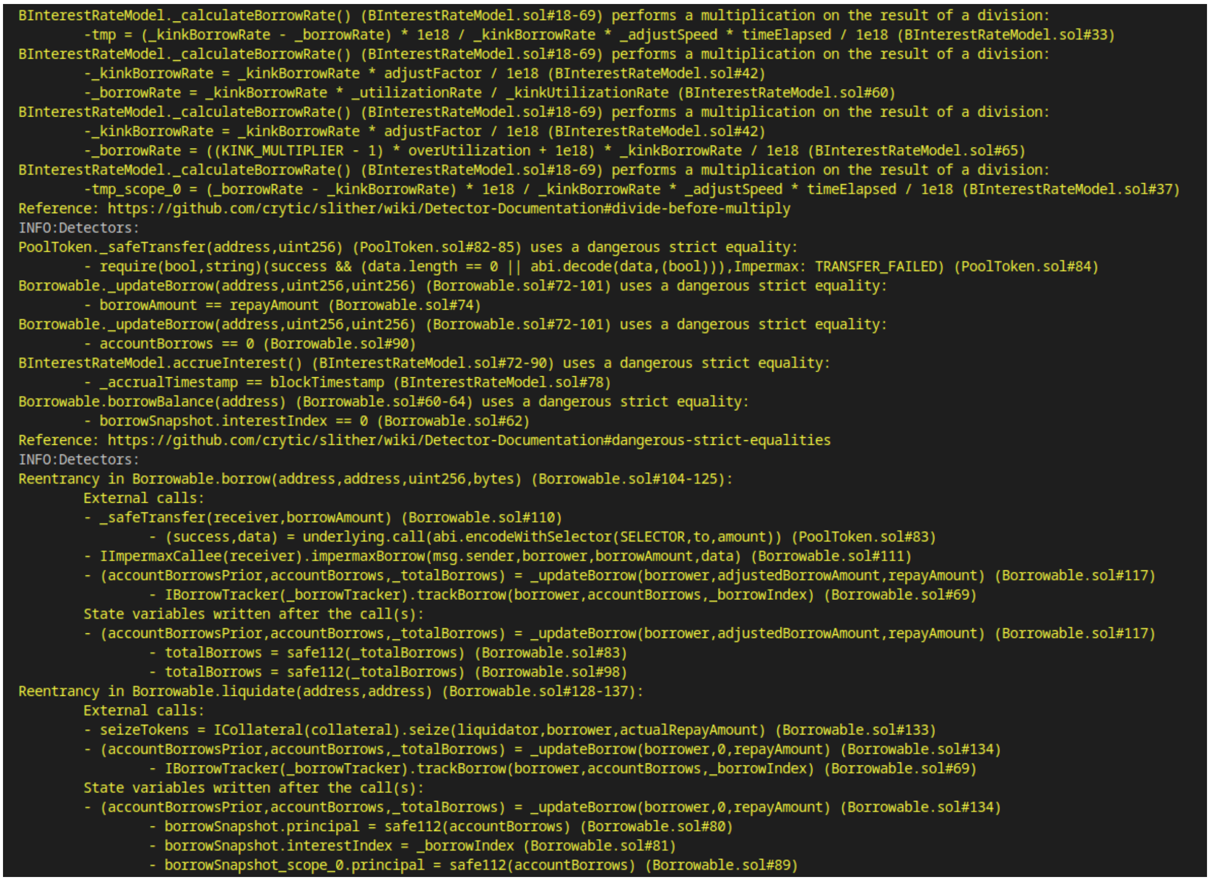
Pic 4. Different pragma directives are used:



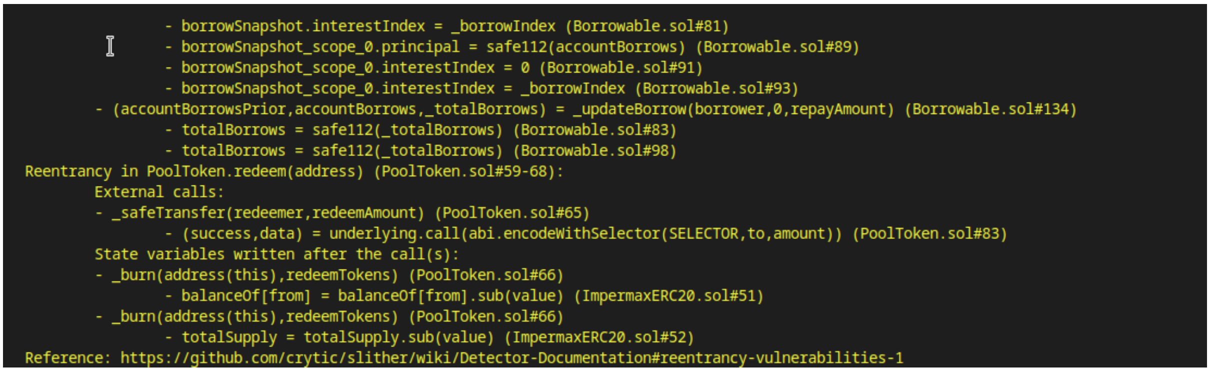
```
        - =0.5.16 (CSetter.sol#1)
        - =0.5.16 (CStorage.sol#1)
        - =0.5.16 (Collateral.sol#1)
        - =0.5.16 (ImpermaxERC20.sol#1)
        - =0.5.16 (PoolToken.sol#1)
        - >=0.5.0 (interfaces/IBorrowable.sol#1)
        - >=0.5.0 (interfaces/ICollateral.sol#1)
        - >=0.5.0 (interfaces/IERC20.sol#1)
        - >=0.5.0 (interfaces/IFactory.sol#1)
        - >=0.5.0 (interfaces/IImpermaxCallee.sol#1)
        - >=0.5.0 (interfaces/IPoolToken.sol#1)
        - >=0.5.0 (interfaces/ISimpleUniswapOracle.sol#1)
        - >=0.5.0 (interfaces/IUniswapV2Pair.sol#1)
        - =0.5.16 (libraries/Math.sol#1)
        - =0.5.16 (libraries/SafeMath.sol#1)
        - =0.5.16 (libraries/UQ112x112.sol#1)
```

Appendix B. Automated tools reports

Pic 1. BAllowance Slither automated report:

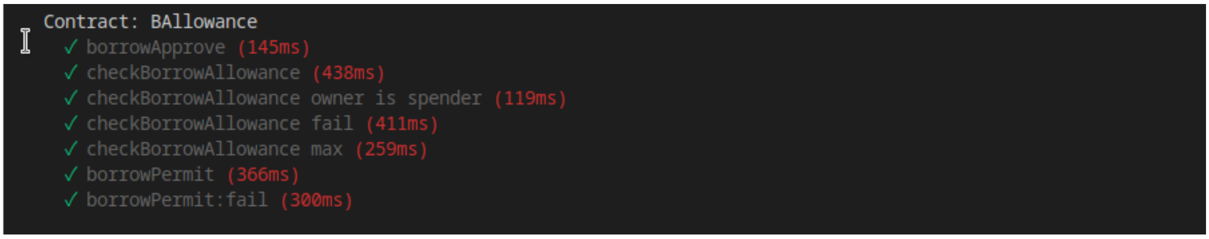


```
BInterestRateModel._calculateBorrowRate() (BInterestRateModel.sol#18-69) performs a multiplication on the result of a division:
        -tmp = (_kinkBorrowRate - _borrowRate) * 1e18 / _kinkBorrowRate * _adjustSpeed * timeElapsed / 1e18 (BInterestRateModel.sol#33)
BInterestRateModel._calculateBorrowRate() (BInterestRateModel.sol#18-69) performs a multiplication on the result of a division:
        -_kinkBorrowRate = _kinkBorrowRate * adjustFactor / 1e18 (BInterestRateModel.sol#42)
        -_borrowRate = _kinkBorrowRate * _utilizationRate / _kinkUtilizationRate (BInterestRateModel.sol#60)
BInterestRateModel._calculateBorrowRate() (BInterestRateModel.sol#18-69) performs a multiplication on the result of a division:
        -_kinkBorrowRate = _kinkBorrowRate * adjustFactor / 1e18 (BInterestRateModel.sol#42)
        -_borrowRate = ((KINK_MULTIPLIER - 1) * overUtilization + 1e18) * _kinkBorrowRate / 1e18 (BInterestRateModel.sol#65)
BInterestRateModel._calculateBorrowRate() (BInterestRateModel.sol#18-69) performs a multiplication on the result of a division:
        -tmp_scope_0 = (_borrowRate - _kinkBorrowRate) * 1e18 / _kinkBorrowRate * _adjustSpeed * timeElapsed / 1e18 (BInterestRateModel.sol#37)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
PoolToken._safeTransfer(address,uint256) (PoolToken.sol#82-85) uses a dangerous strict equality:
        - require(bool,string)(success && (data.length == 0 || abi.decode(data,(bool))),Impermax: TRANSFER_FAILED) (PoolToken.sol#84)
Borrowable._updateBorrow(address,uint256,uint256) (Borrowable.sol#72-101) uses a dangerous strict equality:
        - borrowAmount == repayAmount (Borrowable.sol#74)
Borrowable._updateBorrow(address,uint256,uint256) (Borrowable.sol#72-101) uses a dangerous strict equality:
        - accountBorrows == 0 (Borrowable.sol#90)
BInterestRateModel.accrueInterest() (BInterestRateModel.sol#72-90) uses a dangerous strict equality:
        - _accrualTimestamp == blockTimestamp (BInterestRateModel.sol#78)
Borrowable.borrowBalance(address) (Borrowable.sol#60-64) uses a dangerous strict equality:
        - borrowSnapshot.interestIndex == 0 (Borrowable.sol#62)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in Borrowable.borrow(address,address,uint256,bytes) (Borrowable.sol#104-125):
        External calls:
        - _safeTransfer(receiver,borrowAmount) (Borrowable.sol#110)
                - (success,data) = underlying.call(abi.encodeWithSelector(SELECTOR,to,amount)) (PoolToken.sol#83)
        - IImpermaxCallee(receiver).impermaxBorrow(msg.sender,borrower,borrowAmount,data) (Borrowable.sol#111)
        - (accountBorrowsPrior,accountBorrows,_totalBorrows) = _updateBorrow(borrower,adjustedBorrowAmount,repayAmount) (Borrowable.sol#117)
                - IBorrowTracker(_borrowTracker).trackBorrow(borrower,accountBorrows,_borrowIndex) (Borrowable.sol#69)
        State variables written after the call(s):
        - (accountBorrowsPrior,accountBorrows,_totalBorrows) = _updateBorrow(borrower,adjustedBorrowAmount,repayAmount) (Borrowable.sol#117)
                - totalBorrows = safe112(_totalBorrows) (Borrowable.sol#83)
                - totalBorrows = safe112(_totalBorrows) (Borrowable.sol#98)
Reentrancy in Borrowable.liquidate(address,address) (Borrowable.sol#128-137):
        External calls:
        - seizeTokens = ICollateral(collateral).seize(liquidator,borrower,actualRepayAmount) (Borrowable.sol#133)
        - (accountBorrowsPrior,accountBorrows,_totalBorrows) = _updateBorrow(borrower,0,repayAmount) (Borrowable.sol#134)
                - IBorrowTracker(_borrowTracker).trackBorrow(borrower,accountBorrows,_borrowIndex) (Borrowable.sol#69)
        State variables written after the call(s):
        - (accountBorrowsPrior,accountBorrows,_totalBorrows) = _updateBorrow(borrower,0,repayAmount) (Borrowable.sol#134)
                - borrowSnapshot.principal = safe112(accountBorrows) (Borrowable.sol#80)
                - borrowSnapshot.interestIndex = _borrowIndex (Borrowable.sol#81)
                - borrowSnapshot_scope_0.principal = safe112(accountBorrows) (Borrowable.sol#89)
```



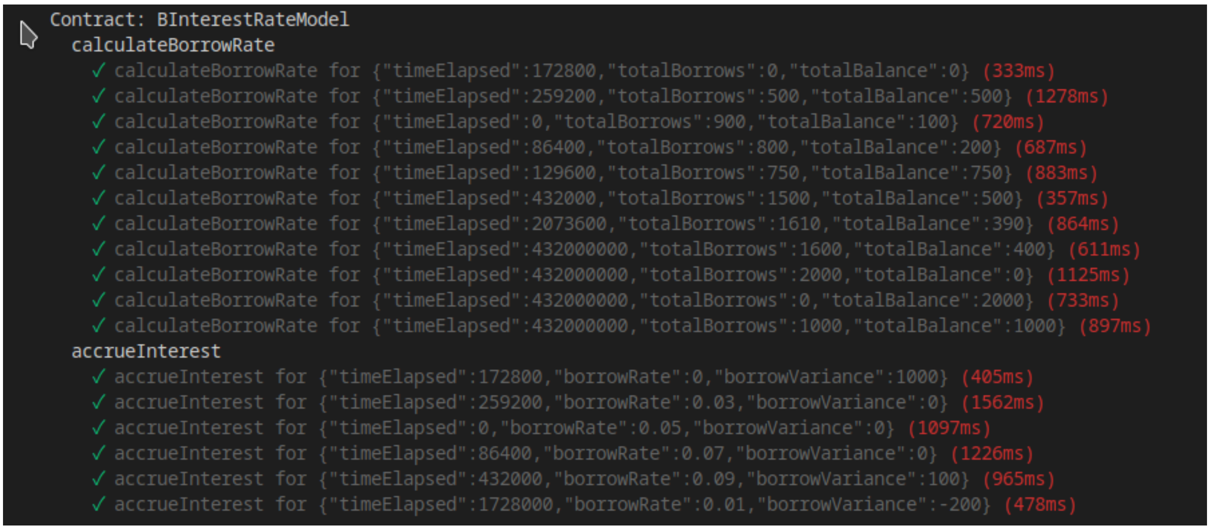
```
                - borrowSnapshot.interestIndex = _borrowIndex (Borrowable.sol#81)
                - borrowSnapshot_scope_0.principal = safe112(accountBorrows) (Borrowable.sol#89)
                - borrowSnapshot_scope_0.interestIndex = 0 (Borrowable.sol#91)
                - borrowSnapshot_scope_0.interestIndex = _borrowIndex (Borrowable.sol#93)
        - (accountBorrowsPrior,accountBorrows,_totalBorrows) = _updateBorrow(borrower,0,repayAmount) (Borrowable.sol#134)
                - totalBorrows = safe112(_totalBorrows) (Borrowable.sol#83)
                - totalBorrows = safe112(_totalBorrows) (Borrowable.sol#98)
Reentrancy in PoolToken.redeem(address) (PoolToken.sol#59-68):
        External calls:
        - _safeTransfer(redeemer,redeemAmount) (PoolToken.sol#65)
                - (success,data) = underlying.call(abi.encodeWithSelector(SELECTOR,to,amount)) (PoolToken.sol#83)
        State variables written after the call(s):
        - _burn(address(this),redeemTokens) (PoolToken.sol#66)
                - balanceOf[from] = balanceOf[from].sub(value) (ImpermaxERC20.sol#51)
        - _burn(address(this),redeemTokens) (PoolToken.sol#66)
                - totalSupply = totalSupply.sub(value) (ImpermaxERC20.sol#52)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
```

Pic 2. Impermax BAllowance truffle automated report:



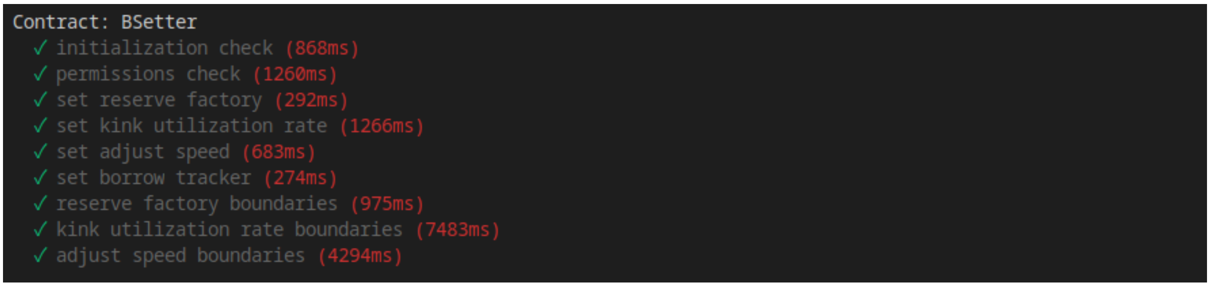
```
  Contract: BAllowance
    ✓ borrowApprove (145ms)
    ✓ checkBorrowAllowance (438ms)
    ✓ checkBorrowAllowance owner is spender (119ms)
    ✓ checkBorrowAllowance fail (411ms)
    ✓ checkBorrowAllowance max (259ms)
    ✓ borrowPermit (366ms)
    ✓ borrowPermit:fail (300ms)
```

Pic 3. Impermax BInterestRateModel truffle automated report:



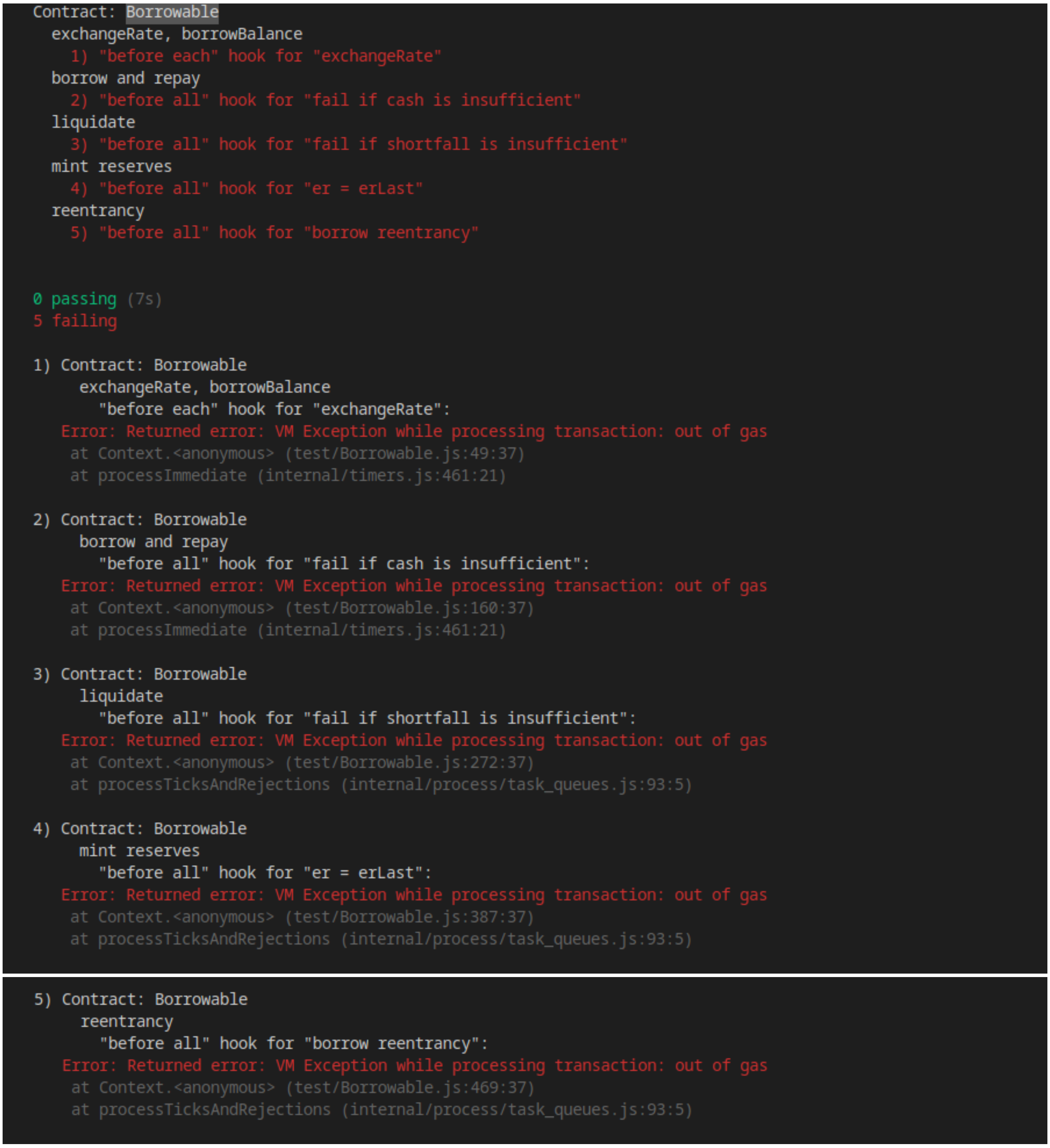
```
 Contract: BInterestRateModel
   calculateBorrowRate
     ✓ calculateBorrowRate for {"timeElapsed":172800,"totalBorrows":0,"totalBalance":0} (333ms)
     ✓ calculateBorrowRate for {"timeElapsed":259200,"totalBorrows":500,"totalBalance":500} (1278ms)
     ✓ calculateBorrowRate for {"timeElapsed":0,"totalBorrows":900,"totalBalance":100} (720ms)
     ✓ calculateBorrowRate for {"timeElapsed":86400,"totalBorrows":800,"totalBalance":200} (687ms)
     ✓ calculateBorrowRate for {"timeElapsed":129600,"totalBorrows":750,"totalBalance":750} (883ms)
     ✓ calculateBorrowRate for {"timeElapsed":432000,"totalBorrows":1500,"totalBalance":500} (357ms)
     ✓ calculateBorrowRate for {"timeElapsed":2073600,"totalBorrows":1610,"totalBalance":390} (864ms)
     ✓ calculateBorrowRate for {"timeElapsed":432000000,"totalBorrows":1600,"totalBalance":400} (611ms)
     ✓ calculateBorrowRate for {"timeElapsed":432000000,"totalBorrows":2000,"totalBalance":0} (1125ms)
     ✓ calculateBorrowRate for {"timeElapsed":432000000,"totalBorrows":0,"totalBalance":2000} (733ms)
     ✓ calculateBorrowRate for {"timeElapsed":432000000,"totalBorrows":1000,"totalBalance":1000} (897ms)
   accrueInterest
     ✓ accrueInterest for {"timeElapsed":172800,"borrowRate":0,"borrowVariance":1000} (405ms)
     ✓ accrueInterest for {"timeElapsed":259200,"borrowRate":0.03,"borrowVariance":0} (1562ms)
     ✓ accrueInterest for {"timeElapsed":0,"borrowRate":0.05,"borrowVariance":0} (1097ms)
     ✓ accrueInterest for {"timeElapsed":86400,"borrowRate":0.07,"borrowVariance":0} (1226ms)
     ✓ accrueInterest for {"timeElapsed":432000,"borrowRate":0.09,"borrowVariance":100} (965ms)
     ✓ accrueInterest for {"timeElapsed":1728000,"borrowRate":0.01,"borrowVariance":-200} (478ms)
```

Pic 4. Impermax BSetter truffle automated report:



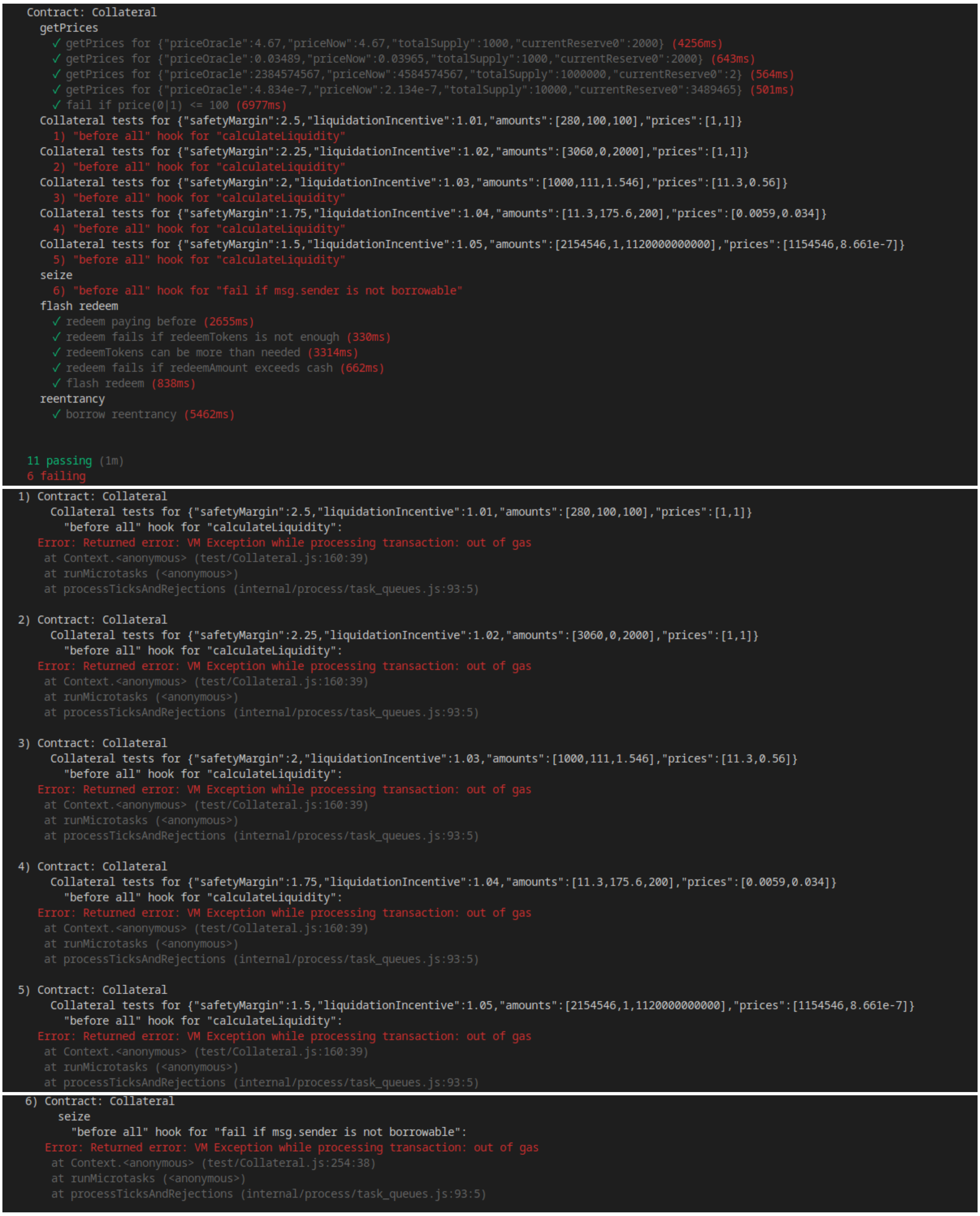
```
Contract: BSetter
  ✓ initialization check (868ms)
  ✓ permissions check (1260ms)
  ✓ set reserve factory (292ms)
  ✓ set kink utilization rate (1266ms)
  ✓ set adjust speed (683ms)
  ✓ set borrow tracker (274ms)
  ✓ reserve factory boundaries (975ms)
  ✓ kink utilization rate boundaries (7483ms)
  ✓ adjust speed boundaries (4294ms)
```

Pic 5. Impermax Borrowable truffle automated report:



```
  Contract: Borrowable
    exchangeRate, borrowBalance
      1) "before each" hook for "exchangeRate"
    borrow and repay
      2) "before all" hook for "fail if cash is insufficient"
    liquidate
      3) "before all" hook for "fail if shortfall is insufficient"
    mint reserves
      4) "before all" hook for "er = erLast"
    reentrancy
      5) "before all" hook for "borrow reentrancy"


  0 passing (7s)
  5 failing

  1) Contract: Borrowable
       exchangeRate, borrowBalance
         "before each" hook for "exchangeRate":
     Error: Returned error: VM Exception while processing transaction: out of gas
      at Context.<anonymous> (test/Borrowable.js:49:37)
      at processImmediate (internal/timers.js:461:21)

  2) Contract: Borrowable
       borrow and repay
         "before all" hook for "fail if cash is insufficient":
     Error: Returned error: VM Exception while processing transaction: out of gas
      at Context.<anonymous> (test/Borrowable.js:160:37)
      at processImmediate (internal/timers.js:461:21)

  3) Contract: Borrowable
       liquidate
         "before all" hook for "fail if shortfall is insufficient":
     Error: Returned error: VM Exception while processing transaction: out of gas
      at Context.<anonymous> (test/Borrowable.js:272:37)
      at processTicksAndRejections (internal/process/task_queues.js:93:5)

  4) Contract: Borrowable
       mint reserves
         "before all" hook for "er = erLast":
     Error: Returned error: VM Exception while processing transaction: out of gas
      at Context.<anonymous> (test/Borrowable.js:387:37)
      at processTicksAndRejections (internal/process/task_queues.js:93:5)

  5) Contract: Borrowable
       reentrancy
         "before all" hook for "borrow reentrancy":
     Error: Returned error: VM Exception while processing transaction: out of gas
      at Context.<anonymous> (test/Borrowable.js:469:37)
      at processTicksAndRejections (internal/process/task_queues.js:93:5)
```

Pic 6. Impermax Collateral truffle automated report:

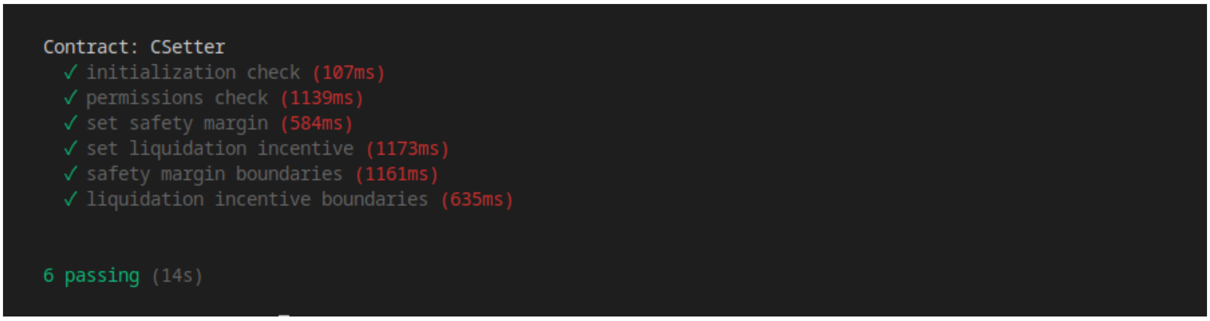


```
  Contract: Collateral
    getPrices
      ✓ getPrices for {"priceOracle":4.67,"priceNow":4.67,"totalSupply":1000,"currentReserve0":2000} (4256ms)
      ✓ getPrices for {"priceOracle":0.03489,"priceNow":0.03965,"totalSupply":1000,"currentReserve0":2000} (643ms)
      ✓ getPrices for {"priceOracle":2384574567,"priceNow":4584574567,"totalSupply":1000000,"currentReserve0":2} (564ms)
      ✓ getPrices for {"priceOracle":4.834e-7,"priceNow":2.134e-7,"totalSupply":10000,"currentReserve0":3489465} (501ms)
      ✓ fail if price(0|1) <= 100 (6977ms)
    Collateral tests for {"safetyMargin":2.5,"liquidationIncentive":1.01,"amounts":[280,100,100],"prices":[1,1]}
      1) "before all" hook for "calculateLiquidity"
    Collateral tests for {"safetyMargin":2.25,"liquidationIncentive":1.02,"amounts":[3060,0,2000],"prices":[1,1]}
      2) "before all" hook for "calculateLiquidity"
    Collateral tests for {"safetyMargin":2,"liquidationIncentive":1.03,"amounts":[1000,111,1.546],"prices":[11.3,0.56]}
      3) "before all" hook for "calculateLiquidity"
    Collateral tests for {"safetyMargin":1.75,"liquidationIncentive":1.04,"amounts":[11.3,175.6,200],"prices":[0.0059,0.034]}
      4) "before all" hook for "calculateLiquidity"
    Collateral tests for {"safetyMargin":1.5,"liquidationIncentive":1.05,"amounts":[2154546,1,1120000000000],"prices":[1154546,8.661e-7]}
      5) "before all" hook for "calculateLiquidity"
    seize
      6) "before all" hook for "fail if msg.sender is not borrowable"
    flash redeem
      ✓ redeem paying before (2655ms)
      ✓ redeem fails if redeemTokens is not enough (330ms)
      ✓ redeemTokens can be more than needed (3314ms)
      ✓ redeem fails if redeemAmount exceeds cash (662ms)
      ✓ flash redeem (838ms)
    reentrancy
      ✓ borrow reentrancy (5462ms)


  11 passing (1m)
  6 failing
```

```
 1) Contract: Collateral
      Collateral tests for {"safetyMargin":2.5,"liquidationIncentive":1.01,"amounts":[280,100,100],"prices":[1,1]}
        "before all" hook for "calculateLiquidity":
    Error: Returned error: VM Exception while processing transaction: out of gas
     at Context.<anonymous> (test/Collateral.js:160:39)
     at runMicrotasks (<anonymous>)
     at processTicksAndRejections (internal/process/task_queues.js:93:5)

 2) Contract: Collateral
      Collateral tests for {"safetyMargin":2.25,"liquidationIncentive":1.02,"amounts":[3060,0,2000],"prices":[1,1]}
        "before all" hook for "calculateLiquidity":
    Error: Returned error: VM Exception while processing transaction: out of gas
     at Context.<anonymous> (test/Collateral.js:160:39)
     at runMicrotasks (<anonymous>)
     at processTicksAndRejections (internal/process/task_queues.js:93:5)

 3) Contract: Collateral
      Collateral tests for {"safetyMargin":2,"liquidationIncentive":1.03,"amounts":[1000,111,1.546],"prices":[11.3,0.56]}
        "before all" hook for "calculateLiquidity":
    Error: Returned error: VM Exception while processing transaction: out of gas
     at Context.<anonymous> (test/Collateral.js:160:39)
     at runMicrotasks (<anonymous>)
     at processTicksAndRejections (internal/process/task_queues.js:93:5)

 4) Contract: Collateral
      Collateral tests for {"safetyMargin":1.75,"liquidationIncentive":1.04,"amounts":[11.3,175.6,200],"prices":[0.0059,0.034]}
        "before all" hook for "calculateLiquidity":
    Error: Returned error: VM Exception while processing transaction: out of gas
     at Context.<anonymous> (test/Collateral.js:160:39)
     at runMicrotasks (<anonymous>)
     at processTicksAndRejections (internal/process/task_queues.js:93:5)

 5) Contract: Collateral
      Collateral tests for {"safetyMargin":1.5,"liquidationIncentive":1.05,"amounts":[2154546,1,1120000000000],"prices":[1154546,8.661e-7]}
        "before all" hook for "calculateLiquidity":
    Error: Returned error: VM Exception while processing transaction: out of gas
     at Context.<anonymous> (test/Collateral.js:160:39)
     at runMicrotasks (<anonymous>)
     at processTicksAndRejections (internal/process/task_queues.js:93:5)
```

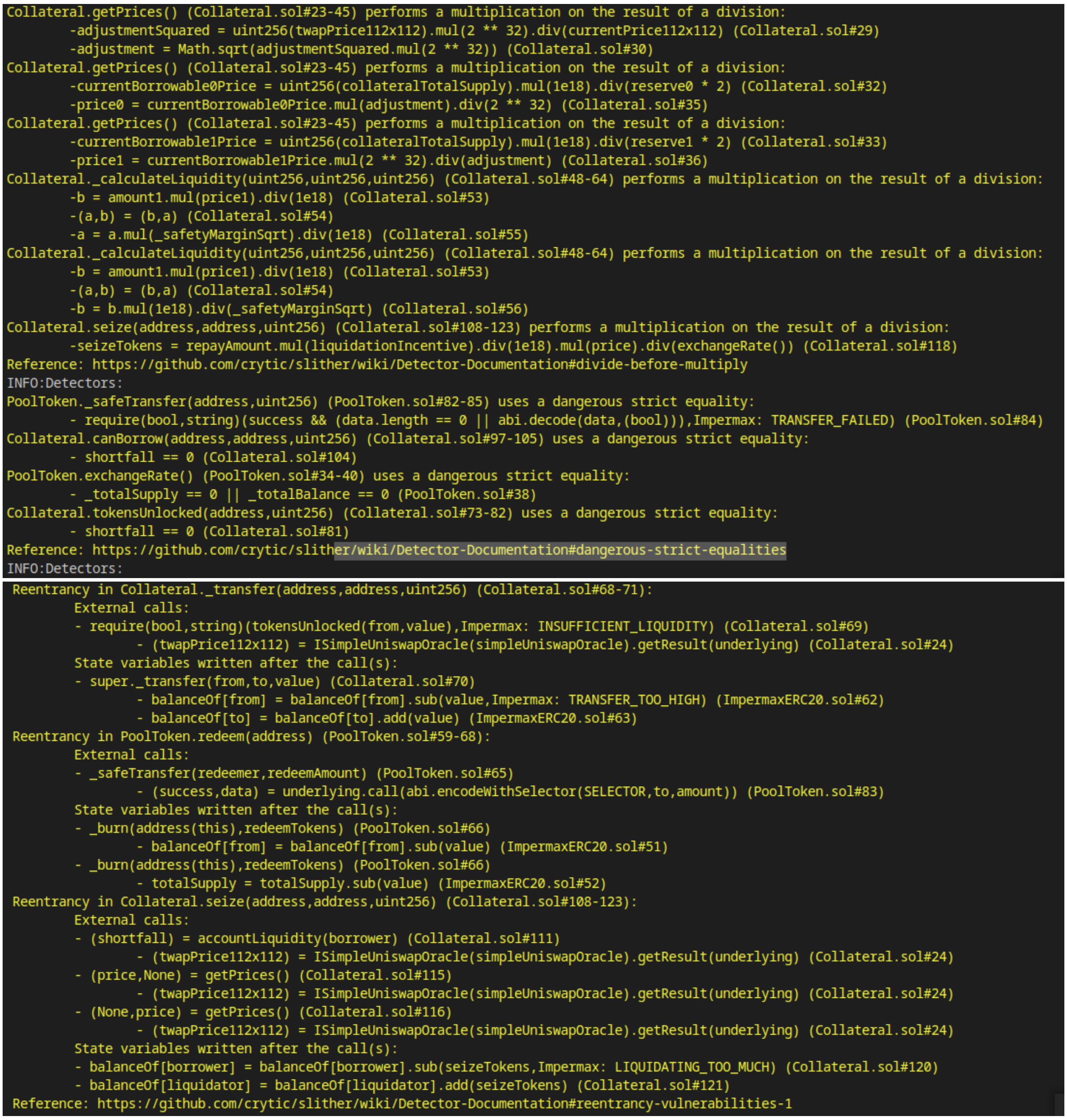
```
 6) Contract: Collateral
      seize
        "before all" hook for "fail if msg.sender is not borrowable":
    Error: Returned error: VM Exception while processing transaction: out of gas
     at Context.<anonymous> (test/Collateral.js:254:38)
     at runMicrotasks (<anonymous>)
     at processTicksAndRejections (internal/process/task_queues.js:93:5)
```

Pic 7. Impermax CSetter truffle automated report:



```
  Contract: CSetter
    ✓ initialization check (107ms)
    ✓ permissions check (1139ms)
    ✓ set safety margin (584ms)
    ✓ set liquidation incentive (1173ms)
    ✓ safety margin boundaries (1161ms)
    ✓ liquidation incentive boundaries (635ms)


  6 passing (14s)
```

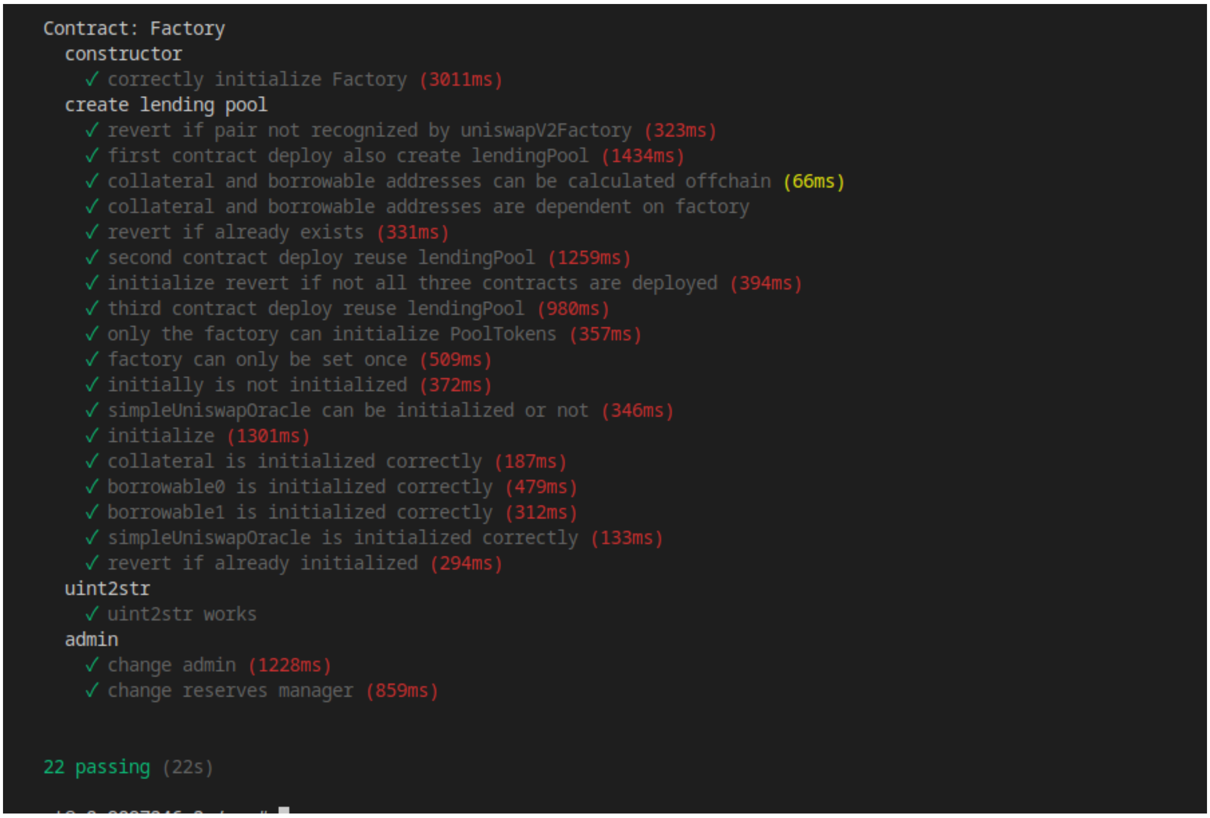
Pic 8. BAllowance Slither automated report:



```
Collateral.getPrices() (Collateral.sol#23-45) performs a multiplication on the result of a division:
        -adjustmentSquared = uint256(twapPrice112x112).mul(2 ** 32).div(currentPrice112x112) (Collateral.sol#29)
        -adjustment = Math.sqrt(adjustmentSquared.mul(2 ** 32)) (Collateral.sol#30)
Collateral.getPrices() (Collateral.sol#23-45) performs a multiplication on the result of a division:
        -currentBorrowable0Price = uint256(collateralTotalSupply).mul(1e18).div(reserve0 * 2) (Collateral.sol#32)
        -price0 = currentBorrowable0Price.mul(adjustment).div(2 ** 32) (Collateral.sol#35)
Collateral.getPrices() (Collateral.sol#23-45) performs a multiplication on the result of a division:
        -currentBorrowable1Price = uint256(collateralTotalSupply).mul(1e18).div(reserve1 * 2) (Collateral.sol#33)
        -price1 = currentBorrowable1Price.mul(2 ** 32).div(adjustment) (Collateral.sol#36)
Collateral._calculateLiquidity(uint256,uint256,uint256) (Collateral.sol#48-64) performs a multiplication on the result of a division:
        -b = amount1.mul(price1).div(1e18) (Collateral.sol#53)
        -(a,b) = (b,a) (Collateral.sol#54)
        -a = a.mul(_safetyMarginSqrt).div(1e18) (Collateral.sol#55)
Collateral._calculateLiquidity(uint256,uint256,uint256) (Collateral.sol#48-64) performs a multiplication on the result of a division:
        -b = amount1.mul(price1).div(1e18) (Collateral.sol#53)
        -(a,b) = (b,a) (Collateral.sol#54)
        -b = b.mul(1e18).div(_safetyMarginSqrt) (Collateral.sol#56)
Collateral.seize(address,address,uint256) (Collateral.sol#108-123) performs a multiplication on the result of a division:
        -seizeTokens = repayAmount.mul(liquidationIncentive).div(1e18).mul(price).div(exchangeRate()) (Collateral.sol#118)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
PoolToken._safeTransfer(address,uint256) (PoolToken.sol#82-85) uses a dangerous strict equality:
        - require(bool,string)(success && (data.length == 0 || abi.decode(data,(bool))),Impermax: TRANSFER_FAILED) (PoolToken.sol#84)
Collateral.canBorrow(address,address,uint256) (Collateral.sol#97-105) uses a dangerous strict equality:
        - shortfall == 0 (Collateral.sol#104)
PoolToken.exchangeRate() (PoolToken.sol#34-40) uses a dangerous strict equality:
        - _totalSupply == 0 || _totalBalance == 0 (PoolToken.sol#38)
Collateral.tokensUnlocked(address,uint256) (Collateral.sol#73-82) uses a dangerous strict equality:
        - shortfall == 0 (Collateral.sol#81)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
```

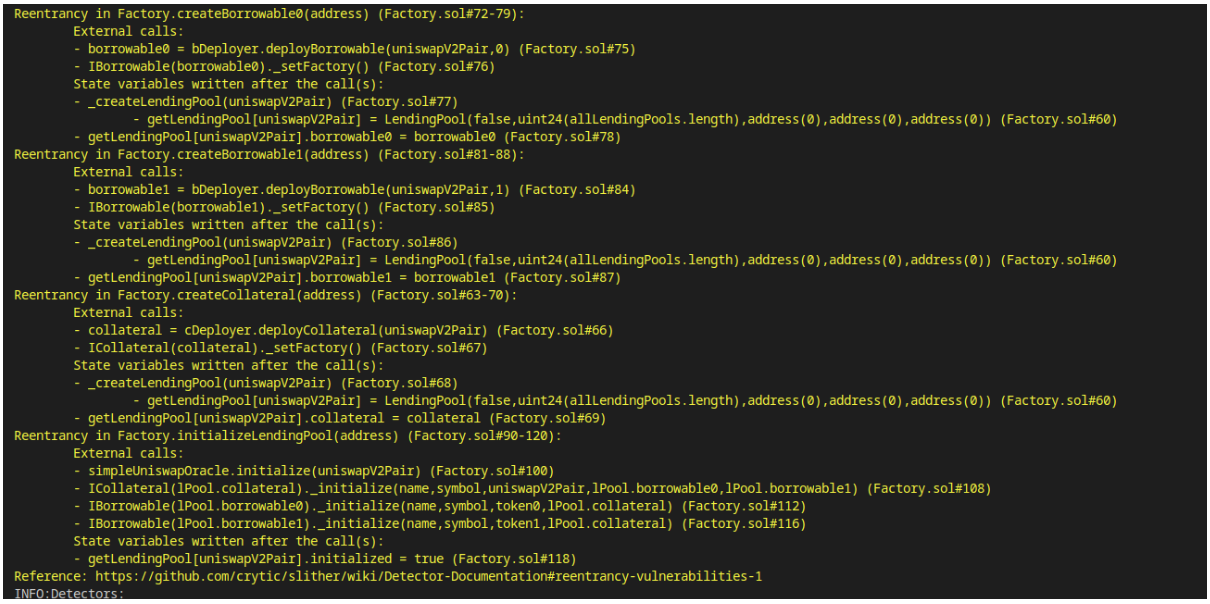
```
Reentrancy in Collateral._transfer(address,address,uint256) (Collateral.sol#68-71):
        External calls:
        - require(bool,string)(tokensUnlocked(from,value),Impermax: INSUFFICIENT_LIQUIDITY) (Collateral.sol#69)
                - (twapPrice112x112) = ISimpleUniswapOracle(simpleUniswapOracle).getResult(underlying) (Collateral.sol#24)
        State variables written after the call(s):
        - super._transfer(from,to,value) (Collateral.sol#70)
                - balanceOf[from] = balanceOf[from].sub(value,Impermax: TRANSFER_TOO_HIGH) (ImpermaxERC20.sol#62)
                - balanceOf[to] = balanceOf[to].add(value) (ImpermaxERC20.sol#63)
Reentrancy in PoolToken.redeem(address) (PoolToken.sol#59-68):
        External calls:
        - _safeTransfer(redeemer,redeemAmount) (PoolToken.sol#65)
                - (success,data) = underlying.call(abi.encodeWithSelector(SELECTOR,to,amount)) (PoolToken.sol#83)
        State variables written after the call(s):
        - _burn(address(this),redeemTokens) (PoolToken.sol#66)
                - balanceOf[from] = balanceOf[from].sub(value) (ImpermaxERC20.sol#51)
        - _burn(address(this),redeemTokens) (PoolToken.sol#66)
                - totalSupply = totalSupply.sub(value) (ImpermaxERC20.sol#52)
Reentrancy in Collateral.seize(address,address,uint256) (Collateral.sol#108-123):
        External calls:
        - (shortfall) = accountLiquidity(borrower) (Collateral.sol#111)
                - (twapPrice112x112) = ISimpleUniswapOracle(simpleUniswapOracle).getResult(underlying) (Collateral.sol#24)
        - (price,None) = getPrices() (Collateral.sol#115)
                - (twapPrice112x112) = ISimpleUniswapOracle(simpleUniswapOracle).getResult(underlying) (Collateral.sol#24)
        - (None,price) = getPrices() (Collateral.sol#116)
                - (twapPrice112x112) = ISimpleUniswapOracle(simpleUniswapOracle).getResult(underlying) (Collateral.sol#24)
        State variables written after the call(s):
        - balanceOf[borrower] = balanceOf[borrower].sub(seizeTokens,Impermax: LIQUIDATING_TOO_MUCH) (Collateral.sol#120)
        - balanceOf[liquidator] = balanceOf[liquidator].add(seizeTokens) (Collateral.sol#121)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
```

Pic 9. Impermax Factory truffle automated report:

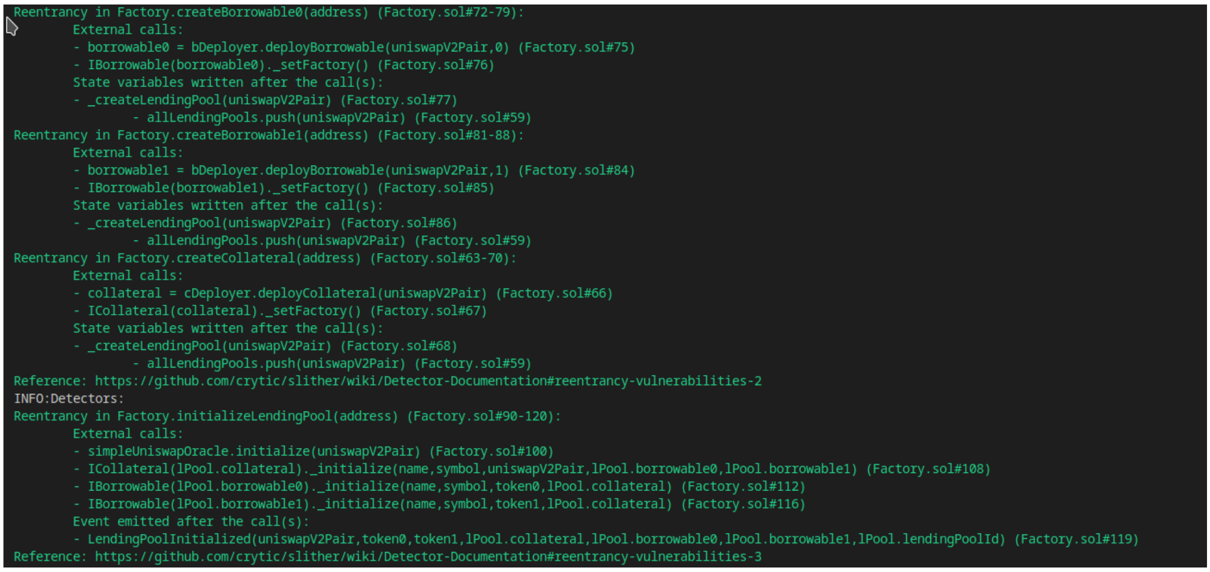


```
  Contract: Factory
    constructor
      ✓ correctly initialize Factory (3011ms)
    create lending pool
      ✓ revert if pair not recognized by uniswapV2Factory (323ms)
      ✓ first contract deploy also create lendingPool (1434ms)
      ✓ collateral and borrowable addresses can be calculated offchain (66ms)
      ✓ collateral and borrowable addresses are dependent on factory
      ✓ revert if already exists (331ms)
      ✓ second contract deploy reuse lendingPool (1259ms)
      ✓ initialize revert if not all three contracts are deployed (394ms)
      ✓ third contract deploy reuse lendingPool (980ms)
      ✓ only the factory can initialize PoolTokens (357ms)
      ✓ factory can only be set once (509ms)
      ✓ initially is not initialized (372ms)
      ✓ simpleUniswapOracle can be initialized or not (346ms)
      ✓ initialize (1301ms)
      ✓ collateral is initialized correctly (187ms)
      ✓ borrowable0 is initialized correctly (479ms)
      ✓ borrowable1 is initialized correctly (312ms)
      ✓ simpleUniswapOracle is initialized correctly (133ms)
      ✓ revert if already initialized (294ms)
    uint2str
      ✓ uint2str works
    admin
      ✓ change admin (1228ms)
      ✓ change reserves manager (859ms)


  22 passing (22s)
```

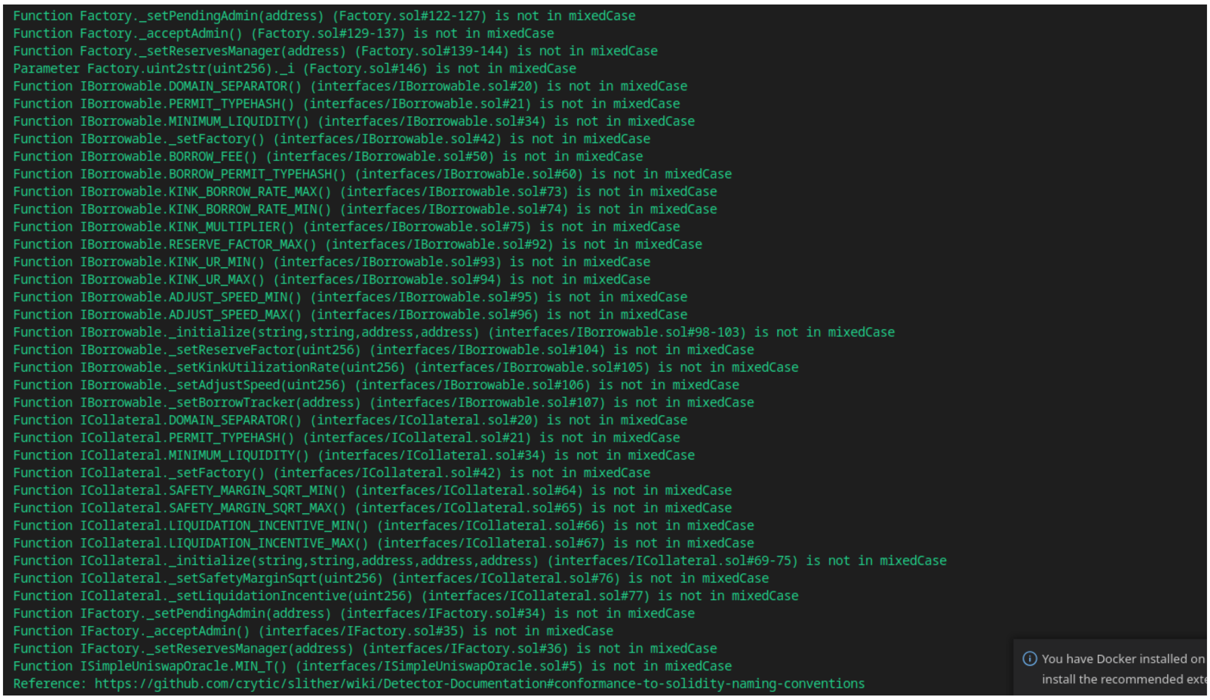
Pic 10. Factory Slither automated report:



```
Reentrancy in Factory.createBorrowable0(address) (Factory.sol#72-79):
        External calls:
        - borrowable0 = bDeployer.deployBorrowable(uniswapV2Pair,0) (Factory.sol#75)
        - IBorrowable(borrowable0)._setFactory() (Factory.sol#76)
        State variables written after the call(s):
        - _createLendingPool(uniswapV2Pair) (Factory.sol#77)
                - getLendingPool[uniswapV2Pair] = LendingPool(false,uint24(allLendingPools.length),address(0),address(0),address(0)) (Factory.sol#60)
        - getLendingPool[uniswapV2Pair].borrowable0 = borrowable0 (Factory.sol#78)
Reentrancy in Factory.createBorrowable1(address) (Factory.sol#81-88):
        External calls:
        - borrowable1 = bDeployer.deployBorrowable(uniswapV2Pair,1) (Factory.sol#84)
        - IBorrowable(borrowable1)._setFactory() (Factory.sol#85)
        State variables written after the call(s):
        - _createLendingPool(uniswapV2Pair) (Factory.sol#86)
                - getLendingPool[uniswapV2Pair] = LendingPool(false,uint24(allLendingPools.length),address(0),address(0),address(0)) (Factory.sol#60)
        - getLendingPool[uniswapV2Pair].borrowable1 = borrowable1 (Factory.sol#87)
Reentrancy in Factory.createCollateral(address) (Factory.sol#63-70):
        External calls:
        - collateral = cDeployer.deployCollateral(uniswapV2Pair) (Factory.sol#66)
        - ICollateral(collateral)._setFactory() (Factory.sol#67)
        State variables written after the call(s):
        - _createLendingPool(uniswapV2Pair) (Factory.sol#68)
                - getLendingPool[uniswapV2Pair] = LendingPool(false,uint24(allLendingPools.length),address(0),address(0),address(0)) (Factory.sol#60)
        - getLendingPool[uniswapV2Pair].collateral = collateral (Factory.sol#69)
Reentrancy in Factory.initializeLendingPool(address) (Factory.sol#90-120):
        External calls:
        - simpleUniswapOracle.initialize(uniswapV2Pair) (Factory.sol#100)
        - ICollateral(lPool.collateral)._initialize(name,symbol,uniswapV2Pair,lPool.borrowable0,lPool.borrowable1) (Factory.sol#108)
        - IBorrowable(lPool.borrowable0)._initialize(name,symbol,token0,lPool.collateral) (Factory.sol#112)
        - IBorrowable(lPool.borrowable1)._initialize(name,symbol,token1,lPool.collateral) (Factory.sol#116)
        State variables written after the call(s):
        - getLendingPool[uniswapV2Pair].initialized = true (Factory.sol#118)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
```



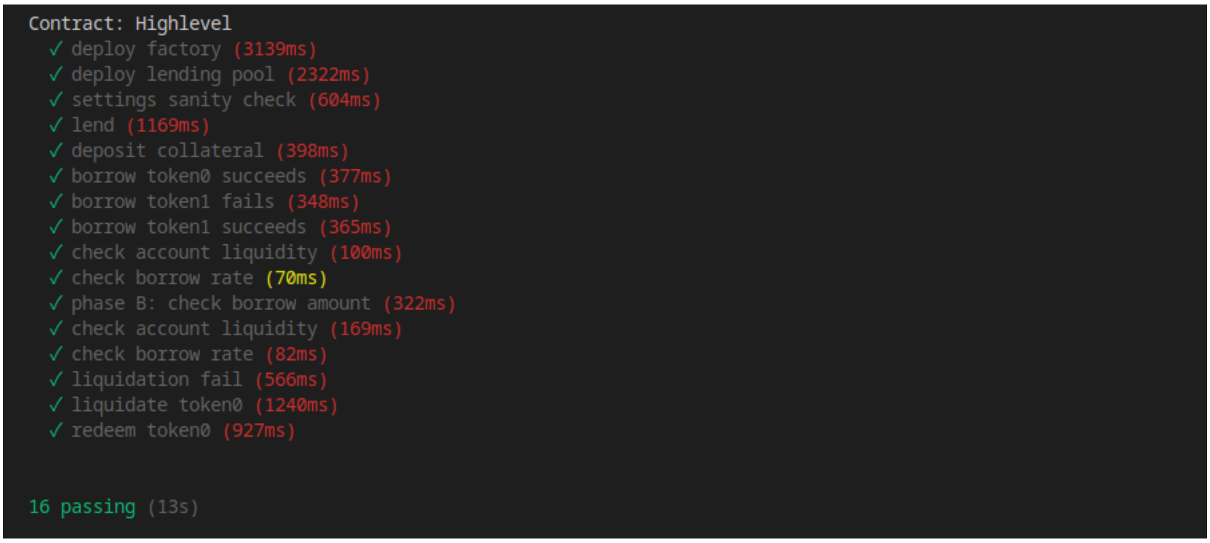
```
Reentrancy in Factory.createBorrowable0(address) (Factory.sol#72-79):
        External calls:
        - borrowable0 = bDeployer.deployBorrowable(uniswapV2Pair,0) (Factory.sol#75)
        - IBorrowable(borrowable0)._setFactory() (Factory.sol#76)
        State variables written after the call(s):
        - _createLendingPool(uniswapV2Pair) (Factory.sol#77)
                - allLendingPools.push(uniswapV2Pair) (Factory.sol#59)
Reentrancy in Factory.createBorrowable1(address) (Factory.sol#81-88):
        External calls:
        - borrowable1 = bDeployer.deployBorrowable(uniswapV2Pair,1) (Factory.sol#84)
        - IBorrowable(borrowable1)._setFactory() (Factory.sol#85)
        State variables written after the call(s):
        - _createLendingPool(uniswapV2Pair) (Factory.sol#86)
                - allLendingPools.push(uniswapV2Pair) (Factory.sol#59)
Reentrancy in Factory.createCollateral(address) (Factory.sol#63-70):
        External calls:
        - collateral = cDeployer.deployCollateral(uniswapV2Pair) (Factory.sol#66)
        - ICollateral(collateral)._setFactory() (Factory.sol#67)
        State variables written after the call(s):
        - _createLendingPool(uniswapV2Pair) (Factory.sol#68)
                - allLendingPools.push(uniswapV2Pair) (Factory.sol#59)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in Factory.initializeLendingPool(address) (Factory.sol#90-120):
        External calls:
        - simpleUniswapOracle.initialize(uniswapV2Pair) (Factory.sol#100)
        - ICollateral(lPool.collateral)._initialize(name,symbol,uniswapV2Pair,lPool.borrowable0,lPool.borrowable1) (Factory.sol#108)
        - IBorrowable(lPool.borrowable0)._initialize(name,symbol,token0,lPool.collateral) (Factory.sol#112)
        - IBorrowable(lPool.borrowable1)._initialize(name,symbol,token1,lPool.collateral) (Factory.sol#116)
        Event emitted after the call(s):
        - LendingPoolInitialized(uniswapV2Pair,token0,token1,lPool.collateral,lPool.borrowable0,lPool.borrowable1,lPool.lendingPoolId) (Factory.sol#119)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```



```
Function Factory._setPendingAdmin(address) (Factory.sol#122-127) is not in mixedCase
Function Factory._acceptAdmin() (Factory.sol#129-137) is not in mixedCase
Function Factory._setReservesManager(address) (Factory.sol#139-144) is not in mixedCase
Parameter Factory.uint2str(uint256)._i (Factory.sol#146) is not in mixedCase
Function IBorrowable.DOMAIN_SEPARATOR() (interfaces/IBorrowable.sol#20) is not in mixedCase
Function IBorrowable.PERMIT_TYPEHASH() (interfaces/IBorrowable.sol#21) is not in mixedCase
Function IBorrowable.MINIMUM_LIQUIDITY() (interfaces/IBorrowable.sol#34) is not in mixedCase
Function IBorrowable._setFactory() (interfaces/IBorrowable.sol#42) is not in mixedCase
Function IBorrowable.BORROW_FEE() (interfaces/IBorrowable.sol#50) is not in mixedCase
Function IBorrowable.BORROW_PERMIT_TYPEHASH() (interfaces/IBorrowable.sol#60) is not in mixedCase
Function IBorrowable.KINK_BORROW_RATE_MAX() (interfaces/IBorrowable.sol#73) is not in mixedCase
Function IBorrowable.KINK_BORROW_RATE_MIN() (interfaces/IBorrowable.sol#74) is not in mixedCase
Function IBorrowable.KINK_MULTIPLIER() (interfaces/IBorrowable.sol#75) is not in mixedCase
Function IBorrowable.RESERVE_FACTOR_MAX() (interfaces/IBorrowable.sol#92) is not in mixedCase
Function IBorrowable.KINK_UR_MIN() (interfaces/IBorrowable.sol#93) is not in mixedCase
Function IBorrowable.KINK_UR_MAX() (interfaces/IBorrowable.sol#94) is not in mixedCase
Function IBorrowable.ADJUST_SPEED_MIN() (interfaces/IBorrowable.sol#95) is not in mixedCase
Function IBorrowable.ADJUST_SPEED_MAX() (interfaces/IBorrowable.sol#96) is not in mixedCase
Function IBorrowable._initialize(string,string,address,address) (interfaces/IBorrowable.sol#98-103) is not in mixedCase
Function IBorrowable._setReserveFactor(uint256) (interfaces/IBorrowable.sol#104) is not in mixedCase
Function IBorrowable._setKinkUtilizationRate(uint256) (interfaces/IBorrowable.sol#105) is not in mixedCase
Function IBorrowable._setAdjustSpeed(uint256) (interfaces/IBorrowable.sol#106) is not in mixedCase
Function IBorrowable._setBorrowTracker(address) (interfaces/IBorrowable.sol#107) is not in mixedCase
Function ICollateral.DOMAIN_SEPARATOR() (interfaces/ICollateral.sol#20) is not in mixedCase
Function ICollateral.PERMIT_TYPEHASH() (interfaces/ICollateral.sol#21) is not in mixedCase
Function ICollateral.MINIMUM_LIQUIDITY() (interfaces/ICollateral.sol#34) is not in mixedCase
Function ICollateral._setFactory() (interfaces/ICollateral.sol#42) is not in mixedCase
Function ICollateral.SAFETY_MARGIN_SQRT_MIN() (interfaces/ICollateral.sol#64) is not in mixedCase
Function ICollateral.SAFETY_MARGIN_SQRT_MAX() (interfaces/ICollateral.sol#65) is not in mixedCase
Function ICollateral.LIQUIDATION_INCENTIVE_MIN() (interfaces/ICollateral.sol#66) is not in mixedCase
Function ICollateral.LIQUIDATION_INCENTIVE_MAX() (interfaces/ICollateral.sol#67) is not in mixedCase
Function ICollateral._initialize(string,string,address,address,address) (interfaces/ICollateral.sol#69-75) is not in mixedCase
Function ICollateral._setSafetyMarginSqrt(uint256) (interfaces/ICollateral.sol#76) is not in mixedCase
Function ICollateral._setLiquidationIncentive(uint256) (interfaces/ICollateral.sol#77) is not in mixedCase
Function IFactory._setPendingAdmin(address) (interfaces/IFactory.sol#34) is not in mixedCase
Function IFactory._acceptAdmin() (interfaces/IFactory.sol#35) is not in mixedCase
Function IFactory._setReservesManager(address) (interfaces/IFactory.sol#36) is not in mixedCase
Function ISimpleUniswapOracle.MIN_T() (interfaces/ISimpleUniswapOracle.sol#5) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

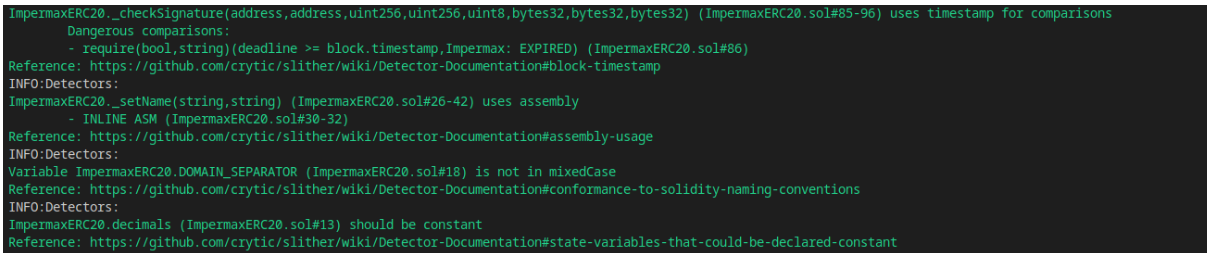
You have Docker installed on
install the recommended exte

Pic 11. Impermax Highlevel truffle automated report:



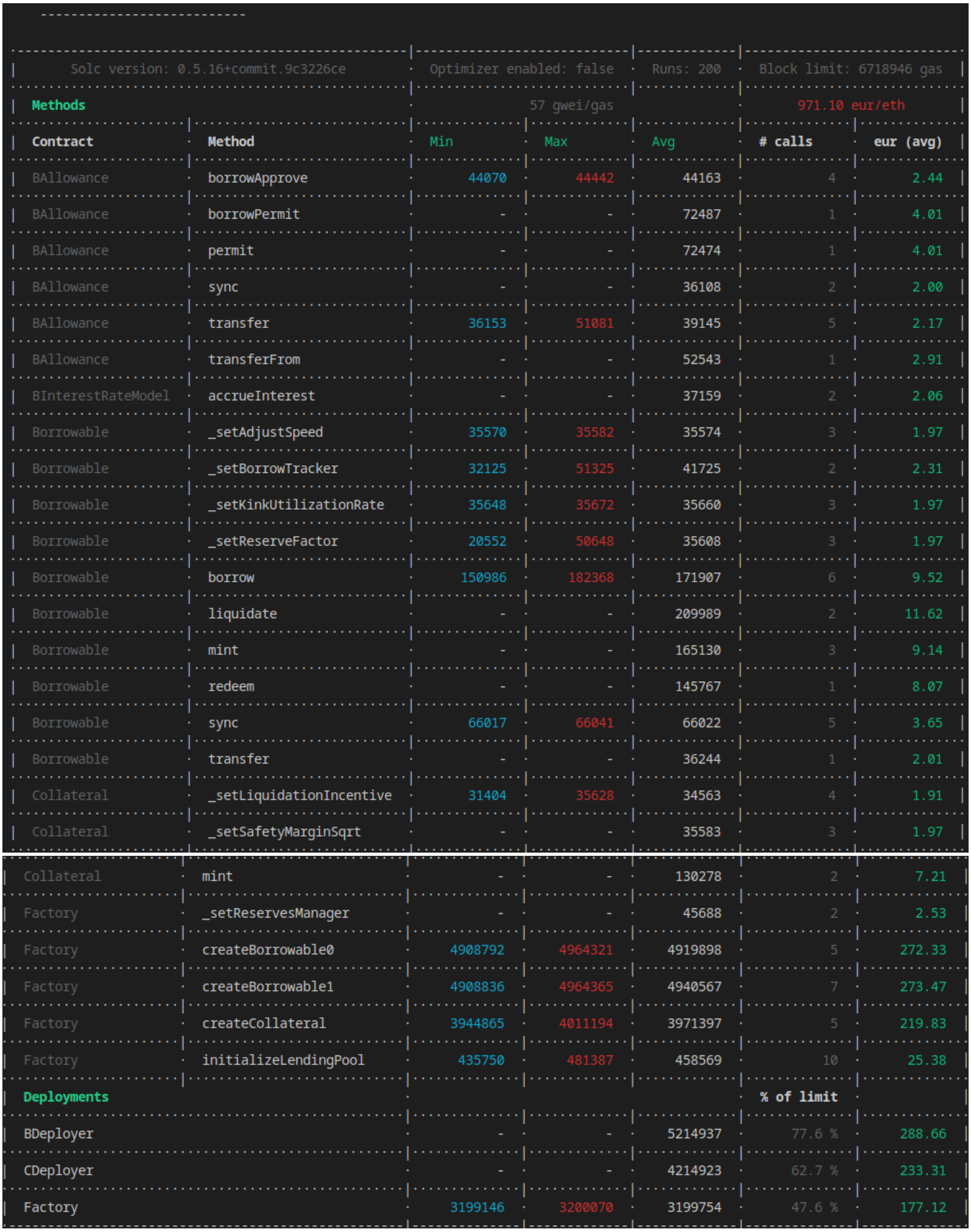
```
Contract: Highlevel
  ✓ deploy factory (3139ms)
  ✓ deploy lending pool (2322ms)
  ✓ settings sanity check (604ms)
  ✓ lend (1169ms)
  ✓ deposit collateral (398ms)
  ✓ borrow token0 succeeds (377ms)
  ✓ borrow token1 fails (348ms)
  ✓ borrow token1 succeeds (365ms)
  ✓ check account liquidity (100ms)
  ✓ check borrow rate (70ms)
  ✓ phase B: check borrow amount (322ms)
  ✓ check account liquidity (169ms)
  ✓ check borrow rate (82ms)
  ✓ liquidation fail (566ms)
  ✓ liquidate token0 (1240ms)
  ✓ redeem token0 (927ms)


16 passing (13s)
```

Pic 12. Factory Slither automated report:



```
ImpermaxERC20._checkSignature(address,address,uint256,uint256,uint8,bytes32,bytes32,bytes32) (ImpermaxERC20.sol#85-96) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(deadline >= block.timestamp,Impermax: EXPIRED) (ImpermaxERC20.sol#86)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
ImpermaxERC20._setName(string,string) (ImpermaxERC20.sol#26-42) uses assembly
        - INLINE ASM (ImpermaxERC20.sol#30-32)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Variable ImpermaxERC20.DOMAIN_SEPARATOR (ImpermaxERC20.sol#18) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
ImpermaxERC20.decimals (ImpermaxERC20.sol#13) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
```

Pic 13. gas usage automated report:



```
   --------------------------

·-----------------------------------------|---------------------------|-------------|------------------------------·
|      Solc version: 0.5.16+commit.9c3226ce  ·  Optimizer enabled: false  ·  Runs: 200  ·  Block limit: 6718946 gas  │
··········································|···························|·············|·······························
|  Methods                                ·                57 gwei/gas                ·       971.10 eur/eth       │
·····················|····················|·············|·············|·············|··············|···············
|  Contract          ·  Method                        ·  Min        ·  Max        ·  Avg        ·  # calls     ·  eur (avg)  │
·····················|····················|·············|·············|·············|··············|···············
|  BAllowance        ·  borrowApprove                 ·      44070  ·      44442  ·      44163  ·           4  ·       2.44  │
·····················|····················|·············|·············|·············|··············|···············
|  BAllowance        ·  borrowPermit                  ·          -  ·          -  ·      72487  ·           1  ·       4.01  │
·····················|····················|·············|·············|·············|··············|···············
|  BAllowance        ·  permit                        ·          -  ·          -  ·      72474  ·           1  ·       4.01  │
·····················|····················|·············|·············|·············|··············|···············
|  BAllowance        ·  sync                          ·          -  ·          -  ·      36108  ·           2  ·       2.00  │
·····················|····················|·············|·············|·············|··············|···············
|  BAllowance        ·  transfer                      ·      36153  ·      51081  ·      39145  ·           5  ·       2.17  │
·····················|····················|·············|·············|·············|··············|···············
|  BAllowance        ·  transferFrom                  ·          -  ·          -  ·      52543  ·           1  ·       2.91  │
·····················|····················|·············|·············|·············|··············|···············
|  BInterestRateModel  ·  accrueInterest              ·          -  ·          -  ·      37159  ·           2  ·       2.06  │
·····················|····················|·············|·············|·············|··············|···············
|  Borrowable        ·  _setAdjustSpeed               ·      35570  ·      35582  ·      35574  ·           3  ·       1.97  │
·····················|····················|·············|·············|·············|··············|···············
|  Borrowable        ·  _setBorrowTracker             ·      32125  ·      51325  ·      41725  ·           2  ·       2.31  │
·····················|····················|·············|·············|·············|··············|···············
|  Borrowable        ·  _setKinkUtilizationRate       ·      35648  ·      35672  ·      35660  ·           3  ·       1.97  │
·····················|····················|·············|·············|·············|··············|···············
|  Borrowable        ·  _setReserveFactor             ·      20552  ·      50648  ·      35608  ·           3  ·       1.97  │
·····················|····················|·············|·············|·············|··············|···············
|  Borrowable        ·  borrow                        ·     150986  ·     182368  ·     171907  ·           6  ·       9.52  │
·····················|····················|·············|·············|·············|··············|···············
|  Borrowable        ·  liquidate                     ·          -  ·          -  ·     209989  ·           2  ·      11.62  │
·····················|····················|·············|·············|·············|··············|···············
|  Borrowable        ·  mint                          ·          -  ·          -  ·     165130  ·           3  ·       9.14  │
·····················|····················|·············|·············|·············|··············|···············
|  Borrowable        ·  redeem                        ·          -  ·          -  ·     145767  ·           1  ·       8.07  │
·····················|····················|·············|·············|·············|··············|···············
|  Borrowable        ·  sync                          ·      66017  ·      66041  ·      66022  ·           5  ·       3.65  │
·····················|····················|·············|·············|·············|··············|···············
|  Borrowable        ·  transfer                      ·          -  ·          -  ·      36244  ·           1  ·       2.01  │
·····················|····················|·············|·············|·············|··············|···············
|  Collateral        ·  _setLiquidationIncentive      ·      31404  ·      35628  ·      34563  ·           4  ·       1.91  │
·····················|····················|·············|·············|·············|··············|···············
|  Collateral        ·  _setSafetyMarginSqrt          ·          -  ·          -  ·      35583  ·           3  ·       1.97  │
·····················|····················|·············|·············|·············|··············|···············
·····················|····················|·············|·············|·············|··············|···············
|  Collateral        ·  mint                          ·          -  ·          -  ·     130278  ·           2  ·       7.21  │
·····················|····················|·············|·············|·············|··············|···············
|  Factory           ·  _setReservesManager           ·          -  ·          -  ·      45688  ·           2  ·       2.53  │
·····················|····················|·············|·············|·············|··············|···············
|  Factory           ·  createBorrowable0             ·    4908792  ·    4964321  ·    4919898  ·           5  ·     272.33  │
·····················|····················|·············|·············|·············|··············|···············
|  Factory           ·  createBorrowable1             ·    4908836  ·    4964365  ·    4940567  ·           7  ·     273.47  │
·····················|····················|·············|·············|·············|··············|···············
|  Factory           ·  createCollateral              ·    3944865  ·    4011194  ·    3971397  ·           5  ·     219.83  │
·····················|····················|·············|·············|·············|··············|···············
|  Factory           ·  initializeLendingPool         ·     435750  ·     481387  ·     458569  ·          10  ·      25.38  │
·····················|····················|·············|·············|·············|··············|···············
|  Deployments                                        ·                                         ·  % of limit  ·             │
··················································|·············|·············|·············|··············|···············
|  BDeployer                                        ·          -  ·          -  ·    5214937  ·      77.6 %  ·     288.66  │
··················································|·············|·············|·············|··············|···············
|  CDeployer                                        ·          -  ·          -  ·    4214923  ·      62.7 %  ·     233.31  │
··················································|·············|·············|·············|··············|···············
|  Factory                                          ·    3199146  ·    3200070  ·    3199754  ·      47.6 %  ·     177.12  │
·-------------------------------------------------|-------------|-------------|-------------|--------------|--------------·
```