SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for: Kairos
Prepared by: Sherlock
Lead Security Expert: 0x52
Dates Audited: March 20 - March 25, 2023
Prepared on: April 19, 2023

Introduction

Kairos Loan is a new type of lending protocol allowing you to borrow instantly using an NFT as collateral and with the best rates on the market.

Scope

kairos-contracts @ b2fd98d62cf0f25ee1db2bd551cd7b4606a5a988

- kairos-contracts/src/AdminFacet.sol
- kairos-contracts/src/AuctionFacet.sol
- kairos-contracts/src/BorrowFacet.sol
- kairos-contracts/src/BorrowLogic/BorrowCheckers.sol
- kairos-contracts/src/BorrowLogic/BorrowHandlers.sol
- kairos-contracts/src/ClaimFacet.sol
- kairos-contracts/src/DataStructure/Errors.sol
- kairos-contracts/src/DataStructure/Global.sol
- kairos-contracts/src/DataStructure/Objects.sol
- kairos-contracts/src/DataStructure/Storage.sol
- kairos-contracts/src/Initializer.sol
- kairos-contracts/src/ProtocolFacet.sol
- kairos-contracts/src/RepayFacet.sol
- kairos-contracts/src/Signature.sol
- kairos-contracts/src/SupplyPositionFacet.sol
- kairos-contracts/src/utils/RayMath.sol
- kairos-contracts/src/utils/Erc20CheckedTransfer.sol
- kairos-contracts/src/utils/NFTokenUtils.sol
- kairos-contracts/src/SupplyPositionLogic/SafeMint.sol

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.

- High issues are directly exploitable security vulnerabilities that need to be fixed.

Issues found

Medium	High
5	0

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

0x52
GimelSec
Inspex
jekapi
deadrxsezzz
Vagner
chaduke
Koolex

Issue M-1: Denial-of-Service in the liquidation flow results in the collateral NTF will be stuck in the contract.

Source: https://github.com/sherlock-audit/2023-02-kairos-judging/issues/76

Found by

Inspex, jekapi

Summary

If the `loanTovalue` value of the offer is extremely high, the liquidation flow will be reverted, causing the collateral NTF to persist in the contract forever.

Vulnerability Detail

The platform allows users to sign offers and provide funds to those who need to borrow assets.

In the first scenario, the lender provided an offer that the `loanTovalue` as high as the result of the `shareMatched` is `0`. For example, if the borrowed amount was `1e40` and the offer had a `loanTovalue` equal to `1e68`, the share would be `0`.

https://github.com/sherlock-audit/2023-02-kairos/blob/main/kairos-contracts/src/BorrowLogic/BorrowHandlers.sol#L47

As a result, an arithmetic error (`Division or modulo by 0`) will occur in the `price()` function at line 50 during the liquidation process.

https://github.com/sherlock-audit/2023-02-kairos/blob/main/kairos-contracts/src/AuctionFacet.sol#L34-L55

In the second scenario, if the lender's share exceeds `0`, but the offer's `loanToValue` is extremely high, the `price()` function at line 54 may encounter an arithmetic error(`Arithmetic over/underflow`) during the `estimatedValue` calculation.

https://github.com/sherlock-audit/2023-02-kairos/blob/main/kairos-contracts/src/AuctionFacet.sol#L54

Poof of Concept

kairos-contracts/test/BorrowBorrow.t.sol

```
function testBorrowOverflow() public {
    uint256 borrowAmount = 1e40;
    BorrowArg[] memory borrowArgs = new BorrowArg[](1);
    (, ,uint256 loanId , ) = kairos.getParameters();
    loanId += 1;
```

```
    Offer memory offer = Offer({
            assetToLend: money,
            loanToValue: 1e61,
            duration: 1,
            expirationDate: block.timestamp + 2 hours,
            tranche: 0,
            collateral: getNft()
        });
    uint256 currentTokenId;

    getFlooz(signer, money, getOfferArg(offer).amount);

    {
        OfferArg[] memory offerArgs = new OfferArg[](1);
        currentTokenId = getJpeg(BORROWER, nft);
        offer.collateral.id = currentTokenId;
        offerArgs[0] = OfferArg({
            signature: getSignature(offer),
            amount: borrowAmount,
            offer: offer
        });
        borrowArgs[0] = BorrowArg({nft: NFToken({id: currentTokenId, implem:
↪   nft}), args: offerArgs});
    }

    vm.prank(BORROWER);
    kairos.borrow(borrowArgs);

    assertEq(nft.balanceOf(BORROWER), 0);
    assertEq(money.balanceOf(BORROWER), borrowAmount);
    assertEq(nft.balanceOf(address(kairos)), 1);

    vm.warp(block.timestamp + 1);
    Loan memory loan = kairos.getLoan(loanId);
    console.log("price of loanId", kairos.price(loanId));
}
```

Impact

The loan position will not be liquidated, which will result in the collateral NTF being permanently frozen in the contract.

Code Snippet

https://github.com/sherlock-audit/2023-02-kairos/blob/main/kairos-contracts/src/BorrowLogic/BorrowHandlers.sol#L47

https://github.com/sherlock-audit/2023-02-kairos/blob/main/kairos-contracts/src/AuctionFacet.sol#L50

https://github.com/sherlock-audit/2023-02-kairos/blob/main/kairos-contracts/src/AuctionFacet.sol#L54

Tool used

Manual Review

Recommendation

We recommend adding the mechanism during the borrowing process to restrict the maximum `loanToValue` limit and ensure that the lender's share is always greater than zero. This will prevent arithmetic errors.

Discussion

npasquie

similar to #34

npasquie

fixed here https://github.com/kairos-loan/kairos-contracts/pull/52

Issue M-2: useLoan doesn't allow liqudator to specifiy maximum price

Source: https://github.com/sherlock-audit/2023-02-kairos-judging/issues/25

Found by

0x52

Summary

useLoan doesn't allow the liquidator to specify a max price they are will to pay for the collateral they are liquidating. On the surface this doesn't seem like an issue because the price is always decreasing due to the dutch auction. However this can be problematic if the chain the contracts are deployed suffers a reorg attack. This can place the transaction earlier than anticipated and therefore charge the user more than they meant to pay. On Ethereum this is unlikely but this is meant to be deployed on any compatible EVM chain many of which are frequently reorganized.

Vulnerability Detail

See summary.

Impact

Liquidator can be charged more than intended

Code Snippet

https://github.com/sherlock-audit/2023-02-kairos/blob/main/kairos-contracts/src/AuctionFacet.sol#L59-L73

Tool used

Manual Review

Recommendation

Allow liquidator to specify a max acceptable price to pay

Discussion

npasquie

fixed here https://github.com/kairos-loan/kairos-contracts/pull/50

Issue M-3: Adversary can utilize a large number of their own loans to cheat other lenders out of interest

Source: https://github.com/sherlock-audit/2023-02-kairos-judging/issues/24

Found by

0x52, GimelSec

Summary

The minimal interest paid by a loan is scaled by the number of provisions that make up the loan. By inflating the number of provisions with their own provisions then can cause legitimate lenders to receive a much lower interest rate than intended.

Vulnerability Detail

ClaimFacet.sol#L94-L106

```
function sendInterests(Loan storage loan, Provision storage provision) internal
 ↪  returns (uint256 sent) {
    uint256 interests = loan.payment.paid - loan.lent;
    if (interests == loan.payment.minInterestsToRepay) {
        // this is the case if the loan is repaid shortly after issuance
        // each lender gets its minimal interest, as an anti ddos measure to
         ↪  spam offer
        sent = provision.amount + (interests / loan.nbOfPositions);
    } else {
        /* provision.amount / lent = share of the interests belonging to the
         ↪  lender. The parenthesis make the
        calculus in the order that maximizes precison */
        sent = provision.amount + (interests * (provision.amount)) / loan.lent;
    }
    loan.assetLent.checkedTransfer(msg.sender, sent);
}
```

If a loan is paid back before the minimal interest rate has been reached then each provision will receive the unweighted minimal interest amount. This can be abused to take loans that pay legitimate lenders a lower APR than expected, cheating them of their yield.

Example: A user wishes to borrow 1000 USDC at 10% APR. Assume the minimal interest per provision is 10 USDC and minimum borrow amount is 20 USDC. After 1 year the user would owe 100 USDC in interest. A user can abuse the minimum to pay legitimate lenders much lower than 10% APR. The attacker will find a legitimate offer to claim 820 USDC. This will create an offer for themselves and borrow 20

USDC from it 9 times. This creates a total of 10 provisions each owed a minimum of 10 USDC or 100 USDC total. Now after 1 year they owe 100 USDC on their loan and the repay the loan. Since 100 USDC is the minimum, each of the 10 provisions will get their minimal interest. 90 USDC will go to their provisions and 10 will go to the legitimate user who loaned them a majority of the USDC. Their APR is ~1.2% which is ~1/9th of what they specified.

Impact

Legitimate users can be cheated out of interest owed

Code Snippet

ClaimFacet.sol#L94-L106

Tool used

Manual Review

Recommendation

The relative size of the provisions should be enforced so that one is not much larger than any other one

Discussion

npasquie

similar to #66

npasquie

fixed by https://github.com/kairos-loan/kairos-contracts/pull/51 the fix restricts the nb of offer/provision per loan to 1 eliminating the vulnerability

Issue M-4: minOfferCost can be bypassed in certain scenarios

Source: https://github.com/sherlock-audit/2023-02-kairos-judging/issues/23

Found by

0x52

Summary

minOfferCost is designed to prevent spam loan requests that can cause the lender to have positions that cost more gas to claim than interest. Due to how interest is calculated right after this minimum is passed it is still possible for the lender to receive less than the minimum.

Vulnerability Detail

ClaimFacet.sol#L94-L106

```
function sendInterests(Loan storage loan, Provision storage provision) internal
 ↪  returns (uint256 sent) {
    uint256 interests = loan.payment.paid - loan.lent;
    if (interests == loan.payment.minInterestsToRepay) {
        // this is the case if the loan is repaid shortly after issuance
        // each lender gets its minimal interest, as an anti ddos measure to
         ↪  spam offer
        sent = provision.amount + (interests / loan.nbOfPositions);
    } else {
        /* provision.amount / lent = share of the interests belonging to the
         ↪  lender. The parenthesis make the
        calculus in the order that maximizes precison */
        sent = provision.amount + (interests * (provision.amount)) / loan.lent;
         ↪  <- audit-issue minimal interest isn't guaranteed
    }
    loan.assetLent.checkedTransfer(msg.sender, sent);
}
```

When a loan has generated more than the minimum interest amount the method for calculating the interest paid is different and depending on the size of the provisions it may lead to provisions that are under the guaranteed minimum.

Example: Assume the minimum interest is 1e18. A loan is filled with 2 provisions. The first provision is 25% and the second is 75%. Since there are two loans the total minimum interest for the loan is 2e18. After some time the paid interest reaches 2.001e18 and the loan is paid back. Since it is above the minimum interest

rate, it is paid out proportionally. This gives 0.5e18 to the first provision and 1.5e18 to the second provision. This violates the minimum guaranteed interest amount.

Impact

Minimum interest guarantee can be violated

Code Snippet

https://github.com/sherlock-audit/2023-02-kairos/blob/main/kairos-contracts/src/ClaimFacet.sol#L94-L106

Tool used

Manual Review

Recommendation

Minimum interest should be set based on the percentage of the lowest provision and provision shouldn't be allowed to be lower than some amount. Since this problem occurs when the percentage is less than 1/n (where n is the number of provisions), any single provision should be allowed to be lower than 1/(2n).

Discussion

npasquie

fixed by https://github.com/kairos-loan/kairos-contracts/pull/51 the fix restricts the nb of offer/provision per loan to 1 eliminating the vulnerability

Issue M-5: If auction price goes to 0, NFT might become unclaimable/ stuck forever

Source: https://github.com/sherlock-audit/2023-02-kairos-judging/issues/17

Found by

GimelSec, Koolex, Vagner, chaduke, deadrxsezzz

Summary

There are certain ERC20 tokens which revert on zero value transfers (e.g. LEND). If an NFT's price drops down to 0, nobody will be able to claim it as the transaction will always revert.

Vulnerability Detail

The time of `loan.auction.duration` passes. The NFT's price is 0. Alice tries to purchase/ claim it, however the ERC20 token on which the auctions is going reverts on 0 value transfers. The NFT becomes stuck forever and no one can take the rights of it.

Consider the following scenarios: 1.

> The ERC20 used in the auction is pausable Throughout the auction, the token gets paused. Now, the auction is still going, the price is dropping and no one is able to claim it. The ERC20 doesn't get unpaused up until the auction ends. Since no one was able to purchase the NFT during the auction, its price now is 0. Since the token reverts on zero value transfers, the NFT is stuck forever.

2.

> Alice is looking at NFT auction which is coming near its end. Alice is looking to purchase the NFT for as little as possible and starts monitoring the mempool, so in case someone tries to buy it, she can front-run the transaction and get the NFT herself. At this point Alice getting the NFT should be guaranteed However, there aren't many other active users/ they aren't paying attention to said NFT. Little time goes by, auction ends and price is set to 0. Alice is happy she can claim the NFT for free. However, the token reverts on 0 value transfers and now no one can claim it and the NFT is lost forever.

Impact

NFT might be lost forever

Code Snippet

https://github.com/sherlock-audit/2023-02-kairos/blob/main/kairos-contracts/src/AuctionFacet.sol#L43-#L45

Tool used

Manual Review

Recommendation

Address ERC20 tokens which revert on 0 value transfers. Auctions which are run with such tokens should have a minimal price of 1 wei (instead of 0)

Discussion

npasquie

fix: https://github.com/kairos-loan/kairos-contracts/pull/49