SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for: DODO
Prepared by: Sherlock
Lead Security Expert: 0x52
Dates Audited: June 19 - July 1, 2023
Prepared on: September 13, 2023

Introduction

A leveraged market making solution to minimize IL and improve on liquidity management. The solution provides yield for retail LPs, higher profits for expert LPs, and better liquidity for traders.

Scope

Repository: DODOEX/new-dodo-v3

Branch: main

Commit: 75ba944a1c5126d73896b853b5baec29b3438311

For the detailed scope, see the contest details.

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
15	7

Security experts who found valid issues

dirk_y
kutugu
seeques
0x52
0xkaden
osmanozdemir1
bitsurfer
jprod15
qckhp
lemonmon
0x4db5362c
Proxy
skyge
Sulpiride
HALITUS
0xG0P1
Oxhunter526
Avci
MohammedRizwan
BugHunter101
0xDjango
0xdice91
tsvetanovv
amaechieth
shogoki
Kalyan-Singh
BugBusters

Vagner
PRAISE
Chandr

shtesesamoubiq
PNS
0xNoodleDon

shealtielanz
seerether
0xHati

Issue H-1: User can perform sandwich attack on withdrawReserves for profit

Source: https://github.com/sherlock-audit/2023-06-dodo-judging/issues/22

Found by

dirk_y, kutugu

Summary

A malicious user could listen to the mempool for calls to `withdrawReserves`, at which point they can perform a sandwich attack by calling `userDeposit` before the withdraw reserves transaction and then `userWithdraw` after the withdraw reserves transaction. They can accomplish this using a tool like flashbots and make an instantaneous profit due to changes in exchange rates.

Vulnerability Detail

When a user deposits or withdraws from the vault, the exchange rate of the token is calculated between the token itself and its dToken. As specified in an inline comment, the exchange rate is calculated like so:

```
// exchangeRate = (cash + totalBorrows -reserves) / dTokenSupply
```

where `reserves = info.totalReserves - info.withdrawnReserves`. When the owner of the vault calls `withdrawReserves` the withdrawnReserves value increases, so the numerator of the above formula increases, and thus the exchange rate increases. An increase in exchange rate means that the same number of dTokens is now worth more of the underlying ERC20.

Below is a diff to the existing test suite that demonstrates the sandwich attack in action:

```
diff --git a/new-dodo-v3/test/DODOV3MM/D3Vault/D3Vault.t.sol
↪  b/new-dodo-v3/test/DODOV3MM/D3Vault/D3Vault.t.sol
index a699162..337d1f5 100644
--- a/new-dodo-v3/test/DODOV3MM/D3Vault/D3Vault.t.sol
+++ b/new-dodo-v3/test/DODOV3MM/D3Vault/D3Vault.t.sol
@@ -233,6 +233,47 @@ contract D3VaultTest is TestContext {
         assertEq(d3Vault.getTotalDebtValue(address(d3MM)), 1300 ether);
     }

+    function testWithdrawReservesSandwichAttack() public {
+        // Get dToken
+        (address dToken2,,,,,,,,,) = d3Vault.getAssetInfo(address(token2));
```

```
+
+        // Approve tokens
+        vm.prank(user1);
+        token2.approve(address(dodoApprove), type(uint256).max);
+        vm.prank(user2);
+        token2.approve(address(dodoApprove), type(uint256).max);
+        vm.prank(user2);
+        D3Token(dToken2).approve(address(dodoApprove), type(uint256).max);
+
+        // Set user quotas and mint tokens
+        mockUserQuota.setUserQuota(user1, address(token2), 1000 ether);
+        mockUserQuota.setUserQuota(user2, address(token2), 1000 ether);
+        token2.mint(user1, 1000 ether);
+        token2.mint(user2, 1000 ether);
+
+        // User 1 deposits to allow pool to borrow
+        vm.prank(user1);
+        d3Proxy.userDeposit(user1, address(token2), 500 ether);
+        token2.mint(address(d3MM), 100 ether);
+        poolBorrow(address(d3MM), address(token2), 100 ether);
+
+        vm.warp(365 days + 1);
+
+        // Accrue interest from pool borrow
+        d3Vault.accrueInterest(address(token2));
+        uint256 reserves = d3Vault.getReservesInVault(address(token2));
+
+        // User 2 performs a sandwich attack on the withdrawReserves call to
↪  make a profit
+        vm.prank(user2);
+        d3Proxy.userDeposit(user2, address(token2), 100 ether);
+        vm.prank(vaultOwner);
+        d3Vault.withdrawReserves(address(token2), reserves);
+        uint256 dTokenBalance = D3Token(dToken2).balanceOf(user2);
+        vm.prank(user2);
+        d3Proxy.userWithdraw(user2, address(token2), dToken2, dTokenBalance);
+        assertGt(token2.balanceOf(user2), 1000 ether);
+    }
+
    function testWithdrawReserves() public {
        vm.prank(user1);
        token2.approve(address(dodoApprove), type(uint256).max);
```

Impact

An attacker can perform a sandwich attack on calls to `withdrawReserves` to make an instantaneous profit from the protocol. This effectively steals funds away from other legitimate users of the protocol.

Code Snippet

https://github.com/sherlock-audit/2023-06-dodo/blob/main/new-dodo-v3/contracts/DODOV3MM/D3Vault/D3VaultFunding.sol#L235

Tool used

Manual Review

Recommendation

There are a couple of ways this type of attack could be prevented:

1. User deposits could have a minimum lock time in the protocol to prevent an immediate withdraw. However the downside is the user will still profit in the same manner due to the fluctuation in exchange rates.
2. Increasing reserves whilst accruing interest could have an equal and opposite decrease in token balance accounting. Every time reserves increase you are effectively taking token value out of the vault and "reserving" it for the protocol. Given the borrow rate is higher than the reserve increase rate, the exchange rate will continue to increase. I think something like the following would work (please note I haven't tested this):

```
diff --git a/new-dodo-v3/contracts/DODOV3MM/D3Vault/D3VaultFunding.sol
↪  b/new-dodo-v3/contracts/DODOV3MM/D3Vault/D3VaultFunding.sol
index 2fb9364..9ad1702 100644
--- a/new-dodo-v3/contracts/DODOV3MM/D3Vault/D3VaultFunding.sol
+++ b/new-dodo-v3/contracts/DODOV3MM/D3Vault/D3VaultFunding.sol
@@ -157,6 +157,7 @@ contract D3VaultFunding is D3VaultStorage {
         uint256 compoundInterestRate =
↪  getCompoundInterestRate(borrowRatePerSecond, deltaTime);
         totalBorrowsNew = borrowsPrior.mul(compoundInterestRate);
         totalReservesNew = reservesPrior + (totalBorrowsNew -
↪  borrowsPrior).mul(info.reserveFactor);
+        info.balance = info.balance - (totalReservesNew - reservesPrior);
         borrowIndexNew = borrowIndexPrior.mul(compoundInterestRate);

         accrualTime = currentTime;
@@ -232,7 +233,7 @@ contract D3VaultFunding is D3VaultStorage {
         uint256 cash = getCash(token);
```

```
          uint256 dTokenSupply = IERC20(info.dToken).totalSupply();
          if (dTokenSupply == 0) { return 1e18; }
-         return (cash + info.totalBorrows - (info.totalReserves -
 ↪  info.withdrawnReserves)).div(dTokenSupply);
+         return (cash + info.totalBorrows).div(dTokenSupply);
      }

      /// @notice Make sure accrueInterests or accrueInterest(token) is called
 ↪  before
```

Discussion

hrishibhat

@traceurl Is this a valid issue?

traceurl

@hrishibhat This is a valid issue.

traceurl

fixed in https://github.com/DODOEX/new-dodo-v3/pull/45

IAm0x52

Fix looks good. info.balance is now factors in the amount removed. Exchange rate before and after withdrawal are the same

Issue H-2: Calls to liquidate don't write down totalBorrows which breaks exchange rate

Source: https://github.com/sherlock-audit/2023-06-dodo-judging/issues/46

Found by

dirk_y, seeques

Summary

When a pool is liquidated, the `totalBorrows` storage slot for the token in question should be decremented by `debtToCover` in order to keep the exchange rate of the corresponding `pToken` correct.

Vulnerability Detail

When users call `liquidate` to liquidate a pool, they specify the amount of debt they want to cover. In the end this is used to write down the borrow amount of the pool in question:

```
record.amount = borrows - debtToCover;
```

However, the `totalBorrows` of the token isn't written down as well (like it should be). The `finishLiquidation` method correctly writes down the `totalBorrows` state.

Impact

When a user calls `liquidate` to liquidate a pool, the exchange rate of the token (from its pToken) remains high (because the `totalBorrows` for the token isn't decremented). The result is that users that have deposited this ERC20 token are receiving a higher rate of interest than they should. Because this interest is not being covered by anyone the end result is that the last withdrawer from the vault will not be able to redeem their pTokens because there isn't enough of the underlying ERC20 token available. The longer the period over which interest accrues, the greater the incentive for LPs to withdraw early.

Code Snippet

https://github.com/sherlock-audit/2023-06-dodo/blob/main/new-dodo-v3/contracts/DODOV3MM/D3Vault/D3VaultLiquidation.sol#L57

Tool used

Manual Review

Recommendation

The `liquidate` method should include the following line to write down the total borrow amount of the debt token being liquidated:

```
info.totalBorrows = info.totalBorrows - debtToCover;
```

Discussion

djb15

Escalate

This is not a duplicate of #211. I believe this and #157 should be grouped together separately.

#211 is about the vault token balance not being updated during liquidations which allows users to claim more dTokens than they should with new deposits.

This issue (and #157) is about the exchange rate for the token being broken during liquidations due to a different variable not being updated. The solution for this issue is different to the solution to #211. The only similarity is that both issues occur during calls to `liquidate`.

sherlock-admin2

> Escalate
>
> This is not a duplicate of #211. I believe this and #157 should be grouped together separately.
>
> #211 is about the vault token balance not being updated during liquidations which allows users to claim more dTokens than they should with new deposits.
>
> This issue (and #157) is about the exchange rate for the token being broken during liquidations due to a different variable not being updated. The solution for this issue is different to the solution to #211. The only similarity is that both issues occur during calls to `liquidate`.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

maarcweiss

Agree with escalation. #157 and #211 do not describe the same issue as #46 and should be duped together

djb15

@maarcweiss Just to confirm, I'm suggesting #46 and #157 are duped together, and #211 is duped with #68, #122 and #156. I.e.

Issue A: #46, #157 Issue B: #211, #68, #122, #156

Just thought I'd check as the above message seems to suggest the opposite :)

hrishibhat

Result: High Has duplicates Agree with the above escalation and comments and two separate sets of issues that should be valid issues

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- djb15: accepted

traceurl

fixed in https://github.com/DODOEX/new-dodo-v3/pull/28

IAm0x52

Fix looks good. totalBorrows is now updated correctly on D3VaultLiquidation#liquidate

Issue H-3: Anyone can sell other users' tokens as `fromToken`, and get the `toToken`'s themselves due to `decodeData.payer` is never checked.

Source: https://github.com/sherlock-audit/2023-06-dodo-judging/issues/106

Found by

dirk_y, jprod15, osmanozdemir1, qckhp

Summary

Anyone can sell other users' tokens as `fromToken`, and get the `toToken`'s themselves due to `decodeData.payer` is never checked.

Vulnerability Detail

Let's examine the token-selling process and the transaction flow.

The user will initiate the transaction with the `sellTokens()` method in the `D3Proxy.sol` contract, and provide multiple inputs like `pool`, `fromToken`, `toToken`, `fromAmount`, `data` etc.

https://github.com/sherlock-audit/2023-06-dodo/blob/main/new-dodo-v3/contracts/DODOV3MM/periphery/D3Proxy.sol#L80-L101

```
// File: D3Proxy.sol
    function sellTokens(
        address pool,
        address to,
        address fromToken,
        address toToken,
        uint256 fromAmount,
        uint256 minReceiveAmount,
        bytes calldata data,
        uint256 deadLine
    ) public payable judgeExpired(deadLine) returns (uint256 receiveToAmount) {
        if (fromToken == _ETH_ADDRESS_) {
            require(msg.value == fromAmount, "D3PROXY_VALUE_INVALID");
            receiveToAmount = ID3MM(pool).sellToken(to, _WETH_, toToken,
↪   fromAmount, minReceiveAmount, data);
        } else if (toToken == _ETH_ADDRESS_) {
            receiveToAmount =
                ID3MM(pool).sellToken(address(this), fromToken, _WETH_,
↪   fromAmount, minReceiveAmount, data);
            _withdrawWETH(to, receiveToAmount);
```

```
            // multicall withdraw weth to user
        } else {
            receiveToAmount = ID3MM(pool).sellToken(to, fromToken, toToken,
↪   fromAmount, minReceiveAmount, data);
        }
    }
```

After some checks, this method in the `D3Proxy.sol` will make a call to the `sellToken()` function in the pool contract (inherits *D3Trading.sol*). After this call, things that will happen in the pool contract are:

1. Transferring the `toToken`'s to the "to" address (with `_transferOut`)
2. Making a callback to `D3Proxy` contract to deposit `fromToken`'s to the pool. (with `IDODOSwapCallback(msg.sender).d3MMSwapCallBack`)
3. Checking the pool balance and making sure that the `fromToken`'s are actually deposited to the pool. (with this line: `IERC20(fromToken).balanceOf(address(this)) - state.balances[fromToken] >= fromAmount`)

You can see the code here: https://github.com/sherlock-audit/2023-06-dodo/blob/main/new-dodo-v3/contracts/DODOV3MM/D3Pool/D3Trading.sol#L108-L118

```
// File: D3Trading.sol
// Method: sellToken()
108.-->  _transferOut(to, toToken, receiveToAmount);
109.
110.     // external call & swap callback
111.-->  IDODOSwapCallback(msg.sender).d3MMSwapCallBack(fromToken, fromAmount,
↪   data);
112.     // transfer mtFee to maintainer
113.     _transferOut(state._MAINTAINER_, toToken, mtFee);
114.
115.     require(
116.-->       IERC20(fromToken).balanceOf(address(this)) -
↪   state.balances[fromToken] >= fromAmount,
117.         Errors.FROMAMOUNT_NOT_ENOUGH
118.     );
```

The source of the vulnerability is the `d3MMSwapCallBack()` function in the `D3Proxy`. It is called by the pool contract with the `fromToken`, `fromAmount` and `data` inputs to make a `fromToken` deposit to the pool.

The issue is that the deposit is made from `decodeData.payer` and **it is never checked if that payer is actually the seller**. Here is the line that causes this vulnerability:

https://github.com/sherlock-audit/2023-06-dodo/blob/main/new-dodo-v3/contracts/DODOV3MM/periphery/D3Proxy.sol#L142

```
//File: D3Proxy.sol
    /// @notice This callback is used to deposit token into D3MM
    /// @param token The address of token
    /// @param value The amount of token need to deposit to D3MM
    /// @param _data Any data to be passed through to the callback
    function d3MMSwapCallBack(address token, uint256 value, bytes calldata
↪   _data) external override {
        require(ID3Vault(_D3_VAULT_).allPoolAddrMap(msg.sender),
↪   "D3PROXY_CALLBACK_INVALID");
        SwapCallbackData memory decodeData;
        decodeData = abi.decode(_data, (SwapCallbackData));
-->     _deposit(decodeData.payer, msg.sender, token, value);
    }
```

An attacker can create a `SwapCallbackData` struct with any regular user's address, encode it and pass it through the `sellTokens()` function, and get the `toToken`'s.

You can say that `_deposit()` will need the payer's approval but the attackers will know that too. A regular user might have already approved the pool & proxy for the max amount. Attackers can easily check any token's allowances and exploit already approved tokens. Or they can simply watch the mempool and front-run any normal seller right after they approve but before they call the `sellTokens()`.

Impact

An attacker can sell any user's tokens and steal their funds.

Code Snippet

https://github.com/sherlock-audit/2023-06-dodo/blob/main/new-dodo-v3/contracts/DODOV3MM/periphery/D3Proxy.sol#L80-L101

```
The `sellTokens()` function in the `D3Proxy.sol`:
// File: D3Proxy.sol
    function sellTokens(
        address pool,
        address to,
        address fromToken,
        address toToken,
        uint256 fromAmount,
        uint256 minReceiveAmount,
        bytes calldata data,
        uint256 deadLine
    ) public payable judgeExpired(deadLine) returns (uint256 receiveToAmount) {
```

```
        if (fromToken == _ETH_ADDRESS_) {
            require(msg.value == fromAmount, "D3PROXY_VALUE_INVALID");
            receiveToAmount = ID3MM(pool).sellToken(to, _WETH_, toToken,
↪   fromAmount, minReceiveAmount, data);
        } else if (toToken == _ETH_ADDRESS_) {
            receiveToAmount =
                ID3MM(pool).sellToken(address(this), fromToken, _WETH_,
↪   fromAmount, minReceiveAmount, data);
            _withdrawWETH(to, receiveToAmount);
            // multicall withdraw weth to user
        } else {
            receiveToAmount = ID3MM(pool).sellToken(to, fromToken, toToken,
↪   fromAmount, minReceiveAmount, data);
        }
    }
```

The `sellToken()` function in the `D3Trading.sol`: https://github.com/sherlock-audit/2023-06-dodo/blob/main/new-dodo-v3/contracts/DODOV3MM/D3Pool/D3Trading.sol#L90-L126

```
// File: D3Trading.sol
// Method: sellToken()
108.-->  _transferOut(to, toToken, receiveToAmount);
109.
110.     // external call & swap callback
111.-->  IDODOSwapCallback(msg.sender).d3MMSwapCallBack(fromToken, fromAmount,
↪   data);
112.     // transfer mtFee to maintainer
113.     _transferOut(state._MAINTAINER_, toToken, mtFee);
114.
115.     require(
116.-->       IERC20(fromToken).balanceOf(address(this)) -
↪   state.balances[fromToken] >= fromAmount,
117.         Errors.FROMAMOUNT_NOT_ENOUGH
118.     );
```

The `d3MMSwapCallBack()` function in the `D3Proxy.sol`: https://github.com/sherlock-audit/2023-06-dodo/blob/main/new-dodo-v3/contracts/DODOV3MM/periphery/D3Proxy.sol#L134-L143

```
//File: D3Proxy.sol
    /// @notice This callback is used to deposit token into D3MM
    /// @param token The address of token
    /// @param value The amount of token need to deposit to D3MM
    /// @param _data Any data to be passed through to the callback
    function d3MMSwapCallBack(address token, uint256 value, bytes calldata
↪   _data) external override {
```

```
        require(ID3Vault(_D3_VAULT_).allPoolAddrMap(msg.sender),
↪   "D3PROXY_CALLBACK_INVALID");
        SwapCallbackData memory decodeData;
        decodeData = abi.decode(_data, (SwapCallbackData));
-->     _deposit(decodeData.payer, msg.sender, token, value);
    }
```

Tool used

Manual Review

Recommendation

I would recommend to check if the `decodeData.payer == msg.sender` in the beginning of the `sellTokens()` function in `D3Proxy` contract. Because msg.sender will be the pool's address if you want to check it in the `d3MMSwapCallBack()` function, and this check will not be valid to see if the payer is actually the seller.

Another option might be creating a local variable called "seller" and saving the msg.sender value when they first started the transaction. After that make `decodeData.payer == seller` check in the `d3MMSwapCallBack()`.

Discussion

Attens1423

fix pr: https://github.com/DODOEX/new-dodo-v3/pull/41/commits/292141d1bb3be71cde6b154f7619c52d628ca18c

IAm0x52

Fix looks good. swapData.payer is now always msg.sender

Issue H-4: A user can get more dTokens than they should get via `D3VaultFunding.userDeposit()`, due to accounting issues in `D3VaultLiquidation.liquidate()`

Source: https://github.com/sherlock-audit/2023-06-dodo-judging/issues/211

Found by

0xkaden, dirk_y, lemonmon, seeques

Summary

The vault token balance (`assetInfo[debt].balance`) is not updated during liquidation (`D3VaultLiquidation.liquidate()`).

Thus, a user who calls `D3VaultFunding.userDeposit()` can get more dTokens than they should get.

Vulnerability Detail

When `D3VaultLiquidation.liquidate()` is called, the debt is transferred to the vault:

https://github.com/sherlock-audit/2023-06-dodo/blob/main/new-dodo-v3/contracts/DODOV3MM/D3Vault/D3VaultLiquidation.sol#L55

But `assetInfo[debt].balance` is not updated, even though the debt tokens were received.

This leads to the issue that if a user deposits this debt token right after the liquidation, they will receive more dTokens in return than they should, because `D3VaultFunding.userDeposit()` is using the wrongly tracked value of `assetInfo[debt].balance`:

https://github.com/sherlock-audit/2023-06-dodo/blob/main/new-dodo-v3/contracts/DODOV3MM/D3Vault/D3VaultFunding.sol#L32-L34

As a result, the protocol will mint more dTokens for the user than they should receive:

https://github.com/sherlock-audit/2023-06-dodo/blob/main/new-dodo-v3/contracts/DODOV3MM/D3Vault/D3VaultFunding.sol#L39-L41

Impact

A user can call `D3VaultFunding.userDeposit()` right after a token got liquidated by `D3VaultLiquidation.liquidate()`, resulting in that the user will receive more dToken

than they should receive, due to accounting issues in `D3VaultLiquidation.liquidate()`.

All LP holders will suffer from inflated dTokens.

Code Snippet

https://github.com/sherlock-audit/2023-06-dodo/blob/main/new-dodo-v3/contracts/DODOV3MM/D3Vault/D3VaultLiquidation.sol#L55

https://github.com/sherlock-audit/2023-06-dodo/blob/main/new-dodo-v3/contracts/DODOV3MM/D3Vault/D3VaultFunding.sol#L32-L34

https://github.com/sherlock-audit/2023-06-dodo/blob/main/new-dodo-v3/contracts/DODOV3MM/D3Vault/D3VaultFunding.sol#L39-L41

Tool used

Manual Review

Recommendation

After `D3VaultLiquidation.liquidate()` is transferring the debt tokens to the vault, update the `assetInfo[debt].balance` of the vault.

If the repaid debt in `D3VaultLiquidation.liquidate()` was meant to be sent to the pool, like in the function `D3VaultLiquidation.liquidateByDODO()`, the `ID3MM(pool).updateReserveByVault(debt)` should be called at the end of `D3VaultLiquidation.liquidate()`. Otherwise a very similar problem can occur since the `state.balances[debtToken]` is not being updated. `state.balances[debtToken]` is used in a similar way in the D3Trading.sol contract to determine the actual balance received.

Discussion

traceurl

fixed in https://github.com/DODOEX/new-dodo-v3/pull/28

IAm0x52

Fix looks good. Balance is now properly updated when a pool is liquidated

Issue H-5: When a D3MM pool repays all of the borrowed funds to vault using `D3Funding.sol repayAll`, an attacker can steal double the amount of those funds from vault

Source: https://github.com/sherlock-audit/2023-06-dodo-judging/issues/217

Found by

0x4db5362c, 0xG0P1, 0xkaden, HALITUS, Proxy, Sulpiride, dirk_y, osmanozdemir1, seeques, skyge

Summary

When a D3MM pool repays all of the borrowed funds to vault using D3Funding.sol repayAll, an attacker can steal double the amount of those funds from vault. This is because the balance of vault is not updated correctly in D3VaultFunding.sol _poolRepayAll.

`amount` should be added in `info.balance` instead of being subtracted.

```
function _poolRepayAll(address pool, address token) internal {
    .
    .
    info.totalBorrows = info.totalBorrows - amount;
    info.balance = info.balance - amount; // amount should be added here
    .
    .
}
```

Vulnerability Detail

A `D3MM pool` can repay all of the borrowed funds from vault using the function D3Funding.sol repayAll which further calls D3VaultFunding.sol poolRepayAll and eventually D3VaultFunding.sol _poolRepayAll.

```
function repayAll(address token) external onlyOwner nonReentrant poolOngoing {
    ID3Vault(state._D3_VAULT_).poolRepayAll(token);
    _updateReserve(token);
    require(checkSafe(), Errors.NOT_SAFE);
}
```

The vault keeps a record of borrowed funds and its current token balance.

`_poolRepayAll()` is supposed to:

1. Decrease the borrowed funds by the repaid amount
2. Increase the token balance by the same amount #vulnerability
3. Transfer the borrowed funds from pool to vault

However, `_poolRepayAll()` is decreasing the token balance instead.

```
function _poolRepayAll(address pool, address token) internal {
    .
    .
    .
    .

    info.totalBorrows = info.totalBorrows - amount;
    info.balance = info.balance - amount; // amount should be added here

    IERC20(token).safeTransferFrom(pool, address(this), amount);

    emit PoolRepay(pool, token, amount, interests);
}
```

Let's say a vault has 100,000 USDC A pool borrows 20,000 USDC from vault

When the pool calls `poolRepayAll()`, the asset info in vault will change as follows:

1. `totalBorrows => 20,000 - 20,000 => 0` // info.totalBorrows - amount
2. `balance => 100,000 - 20,000 => 80,000` // info.balance - amount
3. `tokens owned by vault => 100,000 + 20,000 => 120,000 USDC` // 20,000 USDC is transferred from pool to vault (repayment)
4. The difference of recorded balance (80,000) and actual balance (120,000) is `40,000 USDC`

An attacker waits for the `poolRepayAll()` function call by a pool.

When `poolRepayAll()` is executed, the attacker calls D3VaultFunding.sol userDeposit(), which deposits 40,000 USDC in vault on behalf of the attacker.

After this, the attacker withdraws the deposited amount using D3VaultFunding.sol userWithdraw() and thus gains 40,000 USDC.

```
function userDeposit(address user, address token) external nonReentrant
↪   allowedToken(token) {
    .
    .
    .
    AssetInfo storage info = assetInfo[token];
    uint256 realBalance = IERC20(token).balanceOf(address(this)); // check
↪   tokens owned by vault
```

```
    uint256 amount = realBalance - info.balance; // amount = 120000-80000
    .
    .
    .
    IDToken(info.dToken).mint(user, dTokenAmount);
    info.balance = realBalance;

    emit UserDeposit(user, token, amount);
}
```

Impact

Loss of funds from vault. The loss will be equal to 2x amount of borrowed tokens that a D3MM pool repays using D3VaultFunding.sol poolRepayAll

Code Snippet

D3VaultFunding.sol _poolRepayAll()

```
function _poolRepayAll(address pool, address token) internal {
    .
    .
    info.totalBorrows = info.totalBorrows - amount;
    info.balance = info.balance - amount; // vulnerability: amount should be
↪   added here

    IERC20(token).safeTransferFrom(pool, address(this), amount);

    emit PoolRepay(pool, token, amount, interests);
}
```

Tool used

Manual Review

Recommendation

In D3VaultFunding.sol _poolRepayAll, do the following changes:

Current code: `info.balance = info.balance - amount;`

New (replace '-' with '+'): `info.balance = info.balance + amount;`

Discussion

traceurl

Fixed in this PR: https://github.com/DODOEX/new-dodo-v3/pull/26

IAm0x52

Fix looks good. amount is now correctly added to info.balance instead of subtracted

Issue H-6: vault balance not updated in withdrawReserves()

Source: https://github.com/sherlock-audit/2023-06-dodo-judging/issues/247

Found by

Protocol Team

Vulnerability Detail

https://github.com/sherlock-audit/2023-06-dodo/blob/a8d30e611acc9762029f8756d6a5b81825faf348/new-dodo-v3/contracts/DODOV3MM/D3Vault/D3Vault.sol#L178-L186

Impact

This is a critical bug, will cause wrong exchange rate calculation, which will impact almost all the funding related functions.

Tool Used

Manual Review

Recommandation

update vault balance in withdrawReserves()

Discussion

hrishibhat

Please note: This issue is not part of the contest submissions and is not eligible for contest rewards.

traceurl

fixed in https://github.com/DODOEX/new-dodo-v3/pull/45

IAm0x52

Fix looks good. Same issue and fix as #22

Issue H-7: traceurl - miss updating pool's token balance after finishLiquidation()

Source: https://github.com/sherlock-audit/2023-06-dodo-judging/issues/249

Found by

Protocol Team tracerul

high

Summary

In D3Liquidation's finishLiquidation() function, update pool's token balance is missing.

Vulnerability Detail

https://github.com/sherlock-audit/2023-06-dodo/blob/a8d30e611acc9762029f8756d6a5b81825faf348/new-dodo-v3/contracts/DODOV3MM/D3Vault/D3VaultLiquidation.sol#L145

Impact

After finishLiquidation, there will be unequally of pool's token balance record and IERC20(token).balanceOf(pool), which will causing serious problems.

Tool Used

Manual Review

Recommandation

Call updateReserveByVault after pool transfer token to vault

Discussion

traceurl

fixed in https://github.com/DODOEX/new-dodo-v3/pull/56

IAm0x52

Fix looks good. Debt token is now properly updated after it has been transferred to keep internal balances correct

Issue M-1: possible precision loss in D3VaultLiquidation.finishLiquidation() function when calculating realDebt because of division before multiplication

Source: https://github.com/sherlock-audit/2023-06-dodo-judging/issues/45

Found by

0xdice91, BugBusters, BugHunter101, Kalyan-Singh, MohammedRizwan, Oxhunter526, PRAISE, amaechieth, kutugu

Summary

finishLiquidation() divides before multiplying when calculating realDebt.

Vulnerability Detail

```
uint256 realDebt = borrows.div(record.interestIndex == 0 ? 1e18 :
↪   record.interestIndex).mul(info.borrowIndex);
```

There will be precision loss when calculating the realDebt because solidity truncates values when dividing and dividing before multiplying causes precision loss.

Values that suffered from precision loss will be updated here

```
info.totalBorrows = info.totalBorrows - realDebt;
```

Impact

Values that suffered from precision loss will be updated here

```
info.totalBorrows = info.totalBorrows - realDebt;
```

Code Snippet

https://github.com/sherlock-audit/2023-06-dodo/blob/main/new-dodo-v3/contracts/DODOV3MM/D3Vault/D3VaultLiquidation.sol#L144

https://github.com/sherlock-audit/2023-06-dodo/blob/main/new-dodo-v3/contracts/DODOV3MM/D3Vault/D3VaultLiquidation.sol#L147

Tool used

Manual Review

Recommendation

don't divide before multiplying

Discussion

traceurl

fixed in https://github.com/DODOEX/new-dodo-v3/pull/31

IAm0x52

Doesn't address this issue. Though precision loss is very minimal given the decimal math utilized. Optional change here

Attens1423

There is another fix for this issue. We realised the last one does not work: https://github.com/DODOEX/new-dodo-v3/pull/54/files

IAm0x52

Fix looks good. Creates the _borrowAmount function which fixes the order of operations

Issue M-2: D3Oracle.getPrice() and D3Oracle.getOriginalPrice() doesn't check If Arbitrum sequencer is down for Chainlink feeds

Source: https://github.com/sherlock-audit/2023-06-dodo-judging/issues/62

Found by

0xHati, 0xNoodleDon, 0xdice91, Avci, MohammedRizwan, PNS, PRAISE, bitsurfer, jprod15, kutugu, qckhp, seeques, shogoki, shtesesamoubiq, skyge, tsvetanovv

Summary

When utilizing Chainlink in L2 chains like Arbitrum, it's important to ensure that the prices provided are not falsely perceived as fresh, even when the sequencer is down. This vulnerability could potentially be exploited by malicious actors to gain an unfair advantage.

Vulnerability Detail

There is no check in D3Oracle.getPrice()

```
function getPrice(address token) public view override returns (uint256) {
       require(priceSources[token].isWhitelisted, "INVALID_TOKEN");
       AggregatorV3Interface priceFeed =
↪   AggregatorV3Interface(priceSources[token].oracle);
       (uint80 roundID, int256 price,, uint256 updatedAt, uint80
↪   answeredInRound) = priceFeed.latestRoundData();
       require(price > 0, "Chainlink: Incorrect Price");
       require(block.timestamp - updatedAt < priceSources[token].heartBeat,
↪   "Chainlink: Stale Price");
       require(answeredInRound >= roundID, "Chainlink: Stale Price");
       return uint256(price) * 10 ** (36 - priceSources[token].priceDecimal -
↪   priceSources[token].tokenDecimal);
   }
```

no check in D3Oracle.getOriginalPrice() too

```
function getOriginalPrice(address token) public view override returns (uint256,
↪   uint8) {
       require(priceSources[token].isWhitelisted, "INVALID_TOKEN");
       AggregatorV3Interface priceFeed =
↪   AggregatorV3Interface(priceSources[token].oracle);
       (uint80 roundID, int256 price,, uint256 updatedAt, uint80
↪   answeredInRound) = priceFeed.latestRoundData();
```

```
        require(price > 0, "Chainlink: Incorrect Price");
        require(block.timestamp - updatedAt < priceSources[token].heartBeat,
↪   "Chainlink: Stale Price");
        require(answeredInRound >= roundID, "Chainlink: Stale Price");
        uint8 priceDecimal = priceSources[token].priceDecimal;
        return (uint256(price), priceDecimal);
    }
```

Impact

could potentially be exploited by malicious actors to gain an unfair advantage.

Code Snippet

https://github.com/sherlock-audit/2023-06-dodo/blob/main/new-dodo-v3/contracts/DODOV3MM/periphery/D3Oracle.sol#L48

https://github.com/sherlock-audit/2023-06-dodo/blob/main/new-dodo-v3/contracts/DODOV3MM/periphery/D3Oracle.sol#L58

Tool used

Manual Review

Recommendation

code example of Chainlink:
https://docs.chain.link/data-feeds/l2-sequencer-feeds#example-code

Discussion

traceurl

fixed in https://github.com/DODOEX/new-dodo-v3/pull/44

IAm0x52

Fix looks good. Oracle now checks for sequencer uptime if a sequencer has been set allowing it to function correctly on L2s

Issue M-3: `D3VaultFunding.userWithdraw()` doen not have mindTokenAmount

Source: https://github.com/sherlock-audit/2023-06-dodo-judging/issues/85

Found by

0xDjango, Avci, BugHunter101, Oxhunter526, dirk_y

Summary

`D3VaultFunding.userWithdraw()` doen not have mindTokenAmount, and use `_getExchangeRate` directly.This is vulnerable to a sandwich attack.

Vulnerability Detail

As we can see, `D3VaultFunding.userWithdraw()` doen not have mindTokenAmount, and use `_getExchangeRate` directly.

```
function userWithdraw(address to, address user, address token, uint256
↪  dTokenAmount) external nonReentrant allowedToken(token) returns(uint256
↪  amount) {
        accrueInterest(token);
        AssetInfo storage info = assetInfo[token];
        require(dTokenAmount <= IDToken(info.dToken).balanceOf(msg.sender),
↪  Errors.DTOKEN_BALANCE_NOT_ENOUGH);

        amount = dTokenAmount.mul(_getExchangeRate(token));//@audit does not
↪  check amount value
        IDToken(info.dToken).burn(msg.sender, dTokenAmount);
        IERC20(token).safeTransfer(to, amount);
        info.balance = info.balance - amount;

        // used for calculate user withdraw amount
        // this function could be called from d3Proxy, so we need "user" param
        // In the meantime, some users may hope to use this function directly,
        // to prevent these users fill "user" param with wrong addresses,
        // we use "msg.sender" param to check.
        emit UserWithdraw(msg.sender, user, token, amount);
    }
```

And the `_getExchangeRate()` result is about `cash` , `info.totalBorrows`, `info.totalReserves`,`info.withdrawnReserves`,`dTokenSupply`,This is vulnerable to a sandwich attack leading to huge slippage

```
function _getExchangeRate(address token) internal view returns (uint256) {
        AssetInfo storage info = assetInfo[token];
        uint256 cash = getCash(token);
        uint256 dTokenSupply = IERC20(info.dToken).totalSupply();
        if (dTokenSupply == 0) { return 1e18; }
        return (cash + info.totalBorrows - (info.totalReserves -
↪   info.withdrawnReserves)).div(dTokenSupply);
    }
```

Impact

This is vulnerable to a sandwich attack.

Code Snippet

https://github.com/sherlock-audit/2023-06-dodo/blob/main/new-dodo-v3/contracts/DODOV3MM/D3Vault/D3VaultFunding.sol#L56

Tool used

Manual Review

Recommendation

Add `mindTokenAmount` parameter for `userWithdraw()` function and check if `amount < mindTokenAmount`

Discussion

Attens1423

We will add slippage protection in D3Proxy

traceurl

fixed in https://github.com/DODOEX/new-dodo-v3/pull/43

IAm0x52

Fix looks good. Both userDeposit and userWithdraw now allow the user to specify a minimum amount received

Issue M-4: Possible loss of Funds

Source: https://github.com/sherlock-audit/2023-06-dodo-judging/issues/101

Found by

Avci, BugHunter101, Chandr, HALITUS, MohammedRizwan, PRAISE, amaechieth, kutugu, seerether, shealtielanz, shogoki, skyge, tsvetanovv

Summary

Transfers at Liquidation may silently fail, causing collateral be paid out withoud debt being paid, or liquidator not getting collateral.

Vulnerability Detail

in D3VaultLiquidation.sol:liquidate the caller pays the outstanding debt, and received the collateral tokens with a discount in exchange. However, the transfer of the tokens to pay the debt, as well as the transfer of the collateral Tokens are using the `transferFrom` function of the ERC20 interface instead of `safeTransferFrom` (which is used in other functions). Moreover, the return value of this function is not checked. As not all ERC20 tokens revert on a failed transfer, this could lead to a silent failure of a transfer. As the function will go on in this case this could lead to the liquidation to finish with either:

- The debt not being paid, but the collateral still paid to the caller --> Loss of protocol funds!
- The debt being repaid by the caller, but the collateral is not transferred --> Loss of user funds!

Impact

Possible loss of user or protocol funds.

Code Snippet

https://github.com/sherlock-audit/2023-06-dodo/blob/main/new-dodo-v3/contracts/DODOV3MM/D3Vault/D3VaultLiquidation.sol#L55

https://github.com/sherlock-audit/2023-06-dodo/blob/main/new-dodo-v3/contracts/DODOV3MM/D3Vault/D3VaultLiquidation.sol#L59

Tool used

Manual Review

Recommendation

Usage of `safeTransferFrom` as in other functions of the same contract is recommended.

Discussion

Shogoki

Escalate for 10USDC This is not a duplicate of #203, which talks about USDC and Approval Race Condition protected tokens. Furthermore this is a separate issue together with duplicates like: #5 #42 #64 & #214

sherlock-admin2

> Escalate for 10USDC This is not a duplicate of #203, which talks about USDC and Approval Race Condition protected tokens. Furthermore this is a separate issue together with duplicates like: #5 #42 #64 & #214

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

0xffff11

Agree with escalation, it was wrongly duped.

jesusrod15

I agree that all the issues that came about race approvals were wrongly duped. Likewise, the rest of the issues that reported incorrect use of transferfrom instead of safetrasnferfrom were always duplicated in the same way in other contest regardless that whats happens. So I think this is a duplicate x example of #11 #34 #5 #42 #88 and others issue as this, this is not duplicate of #2 #35 and others issue as this that are duplicate of #203

hrishibhat

Result: Medium Has duplicates The duplication has been changed accordingly

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- Shogoki: accepted

H4LITUS

This is a duplicate of #214. So why was the status of 214 changed from `reward` to `non-reward`?

Proxy1967

#151 is a dup of this issue yet the status was changed from reward to non-reward and to low/info. Why ? @hrishibhat

hrishibhat

@KirinFilip I don't think #151 clearly describes any issue. Please add a little more detail in the future from the context of the code in the submissions.

Proxy1967

@hrishibhat understandable, thanks for the explanation.

Attens1423

fix pr: https://github.com/DODOEX/new-dodo-v3/pull/30

IAm0x52

Fix looks good.

Issue M-5: D3Oracle will return the wrong price if the Chainlink aggregator returns price outside min/max range

Source: https://github.com/sherlock-audit/2023-06-dodo-judging/issues/129

Found by

0xdice91, BugHunter101, MohammedRizwan, PRAISE, Proxy, dirk_y, kutugu

Summary

Chainlink oracles have a min and max price that they return. If the price goes below the minimum price the oracle will not return the correct price but only the min price. Same goes for the other extremity.

Vulnerability Detail

Both getPrice() and getOriginalPrice() only check `price > 0` not are they within the correct range

```
(uint80 roundID, int256 price,, uint256 updatedAt, uint80 answeredInRound) =
↪  priceFeed.latestRoundData();
require(price > 0, "Chainlink: Incorrect Price");
require(block.timestamp - updatedAt < priceSources[token].heartBeat, "Chainlink:
↪  Stale Price");
require(answeredInRound >= roundID, "Chainlink: Stale Price");
```

Impact

The wrong price may be returned in the event of a market crash. The functions with the issue are used in D3VaultFunding.sol, D3VaultLiquidation.sol and D3UserQuota.sol

Code Snippet

- D3Oracle.sol functions:
  - getPrice()
  - getOriginalPrice()

Tool used

Manual Review

Recommendation

Check the latest answer against reasonable limits and/or revert in case you get a bad price

```
require(price >= minAnswer && price <= maxAnswer, "invalid price");
```

Discussion

Attens1423

How can we get minPrice and maxPrice from oracle contract? Could you give us a more detailed procession?

0xffff11

https://docs.chain.link/data-feeds#check-the-latest-answer-against-reasonable-limits @Attens1423

Attens1423

We understand this doc. If you could offer a code example, including how to get minPrice and maxPrice from code, we would appreciate it

Issue M-6: parseAllPrice not support the tokens whose decimal is greater than 18

Source: https://github.com/sherlock-audit/2023-06-dodo-judging/issues/154

Found by

kutugu

Summary

`parseAllPrice` not support the token decimal is greater than 18, such as NEAR with 24 decimal. Since `buyToken / sellToken` is dependent on `parseAllPrice`, so users can't trade tokens larger than 18 decimal, but DODOv3 is intended to be compatible with all standard ERC20, which is not expected.

Vulnerability Detail

```
// fix price decimal
if (tokenDecimal != 18) {
    uint256 fixDecimal = 18 - tokenDecimal;
    bidDownPrice = bidDownPrice / (10 ** fixDecimal);
    bidUpPrice = bidUpPrice / (10 ** fixDecimal);
    askDownPrice = askDownPrice * (10 ** fixDecimal);
    askUpPrice = askUpPrice * (10 ** fixDecimal);
}
```

If `tokenDecimal > 18`, `18 - tokenDecimal` will revert

Impact

DODOv3 is not compatible the tokens whose decimal is greater than 18, users can't trade them.

Code Snippet

https://github.com/sherlock-audit/2023-06-dodo/blob/a8d30e611acc9762029f8756d6a5b81825faf348/new-dodo-v3/contracts/DODOV3MM/lib/MakerTypes.sol#L99-L106

Tool used

Manual Review

Recommendation

Fix decimal to 36 instead of 18

Discussion

traceurl

fixed in https://github.com/DODOEX/new-dodo-v3/pull/32 now use parseRealAmount() in Types.sol to deal with token whose decimals is not 18

KuTuGu

Escalate There are two questions:

1. This issue does not seem to be the repeat of the other two issues, they refer to overflows where the sum of two oracle token decimals is greater than 36, while this issue targets overflows where a single token decimal is greater than 18
2. Sponsor indicates that it is mainly for chainlink tokens, and there are tokens greater than 18 decimals on the main network: NEAR: 24 decimals, address: https://etherscan.io/token/0x85f17cf997934a597031b2e18a9ab6ebd4b9f6a4, oracle: https://etherscan.io/address/0xC12A6d1D827e23318266Ef16Ba6F397F2F91dA9b

sherlock-admin2

> Escalate There are two questions:
>
> 1. This issue does not seem to be the repeat of the other two issues, they refer to overflows where the sum of two oracle token decimals is greater than 36, while this issue targets overflows where a single token decimal is greater than 18
> 2. Sponsor indicates that it is mainly for chainlink tokens, and there are tokens greater than 18 decimals on the main network: NEAR: 24 decimals, address: https://etherscan.io/token/0x85f17cf997934a597031b2e18a9ab6ebd4b9f6a4, oracle: https://etherscan.io/address/0xC12A6d1D827e23318266Ef16Ba6F397F2F91dA9b

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

maarcweiss

I agree with escalation. As > 18 decimals tokens are more than just one specific non standard erc20, and the likelihood is higher, I would agree with a medium

hrishibhat

Result: Medium Unique Considering this a valid medium

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- kutugu: accepted

traceurl

We will not support tokens whose decimals are greater than 18, like NEAR. We use D3Oracle to whitelist tokens. Even if a token has chainlink price feed, if it's decimals are greater than 18, we will not add it to D3Oracle.

Issue M-7: Wrong assignment of `cumulativeBid` for Range-Order state in getRangeOrderState function

Source: https://github.com/sherlock-audit/2023-06-dodo-judging/issues/178

Found by

bitsurfer

Summary

Wrong assignment of `cumulativeBid` for RangeOrder state

Vulnerability Detail

In `D3Trading`, the `getRangeOrderState` function is returning RangeOrder (get swap status for internal swap) which is assinging wrong toTokenMMInfo.cumulativeBid which suppose to be `cumulativeBid` not `cumulativeAsk`

The error lies in the assignment of `roState.toTokenMMInfo.cumulativeBid`. Instead of assigning `tokenCumMap[toToken].cumulativeAsk`, it should be assigning `tokenCumMap[toToken].cumulativeBid`.

```
File: D3Trading.sol
86:         roState.toTokenMMInfo.cumulativeBid =
87:             allFlag >> (toTokenIndex) & 1 == 0 ? 0 :
 ↪  tokenCumMap[toToken].cumulativeAsk;
```

This wrong assignment value definitely will mess up accounting balance, resulting unknown state will occure, which is not expected by the protocol

For one case, this `getRangeOrderState` is being used in `querySellTokens` & `queryBuyTokens` which may later called from `sellToken` and `buyToken`. The issue is when calling `_contructTokenState` which can be reverted from `PMMRangeOrder` when buy or sell token

```
File: PMMRangeOrder.sol
100:         // B
101:         tokenState.B = askOrNot ? tokenState.B0 - tokenMMInfo.cumulativeAsk
 ↪  : tokenState.B0 - tokenMMInfo.cumulativeBid;
```

When the `tokenMMInfo.cumulativeBid` (which was wrongly assign from `cumulativeAsk`) is bigger than `tokenState.B0`, this will revert

Impact

This wrong assignment value definitely will mess up accounting balance, resulting unknown state will occure, which is not expected by the protocol. For example reverting state showing a case above.

Code Snippet

https://github.com/sherlock-audit/2023-06-dodo/blob/main/new-dodo-v3/contracts/DODOV3MM/D3Pool/D3Trading.sol#L86-L87

Tool used

Manual Review

Recommendation

Fix the error to

```
File: D3Trading.sol
86:         roState.toTokenMMInfo.cumulativeBid =
--:             allFlag >> (toTokenIndex) & 1 == 0 ? 0 :
 ↪  tokenCumMap[toToken].cumulativeAsk;
++:             allFlag >> (toTokenIndex) & 1 == 0 ? 0 :
 ↪  tokenCumMap[toToken].cumulativeBid;
```

Discussion

Attens1423

fix pr:https://github.com/DODOEX/new-dodo-v3/pull/40

IAm0x52

Fix looks good. getRangeOrderState now correctly updates cumulativeBid instead of cumulativeAsk twice

Issue M-8: D3VaultFunding#checkBadDebtAfterAccrue is inaccurate and can lead to further damage to both LP's and MM

Source: https://github.com/sherlock-audit/2023-06-dodo-judging/issues/192

Found by

0x52

Summary

D3VaultFunding#checkBadDebtAfterAccrue makes the incorrect assumption that a collateral ratio of less than 1e18 means that the pool has bad debt. Due to how collateral and debt weight affect the collateral ratio calculation a pool can have a collateral ratio less than 1e18 will still maintaining debt that is profitable to liquidate. The result of this is that the after this threshold has been passed, a pool can no longer be liquidate by anyone which can lead to continued losses that harm both the LPs and the MM being liquidated.

Vulnerability Detail

D3VaultFunding.sol#L382-L386

```
if (balance >= borrows) {
    collateral += min(balance - borrows,
    ↪  info.maxCollateralAmount).mul(info.collateralWeight).mul(price);
} else {
    debt += (borrows - balance).mul(info.debtWeight).mul(price);
}
```

When calculating the collateral and debt values, the value of the collateral is adjusted by the collateralWeight and debtWeight respectively. This can lead to a position in which the collateral ratio is less than 1e18, which incorrectly signals the pool has bad debt via the checkBadDebtAfterAccrue check.

Example:

```
Assume a pool has the following balances and debts:

Token A - 100 borrows 125 balance
Token B - 100 borrows 80 balance

Price A = 1
collateralWeightA = 0.8
```

```
Price B = 1
debtWeightB = 1.2

collateral = 25 * 1 * 0.8 = 20
debt = 20 * 1 * 1.2 = 24

collateralRatio = 20/24 = 0.83
```

The problem here is that there is no bad debt at all and it is still profitable to liquidate this pool, even with a discount:

```
ExcessCollateral = 125 - 100 = 25

25 * 1 * 0.95 [DISCOUNT] = 23.75

ExcessDebt = 100 - 80 = 20

20 * 1 = 20
```

The issue with this is that once this check has been triggered, no other market participants besides DODO can liquidate this position. This creates a significant inefficiency in the market that can easily to real bad debt being created for the pool. This bad debt is harmful to both the pool MM, who could have been liquidated with remaining collateral, and also the vault LPs who directly pay for the bad debt.

Impact

Unnecessary loss of funds to LPs and MMs

Code Snippet

D3VaultFunding.sol#L308-L310

Tool used

Manual Review

Recommendation

The methodology of the bad debt check should be changed to remove collateral and debt weights to accurately indicate the presence of bad debt.

Discussion

Attens1423

The market maker actually controls two contracts with two separate accounts. The owner account of D3Maker is responsible for price feeding, while the owner account of D3MM is responsible for depositing and withdrawing funds. The use of modifiers here meets the design requirements https://github.com/sherlock-audit/2023-06-dodo/blob/a8d30e611acc9762029f8756d6a5b81825faf348/new-dodo-v3/contracts/DODOV3MM/D3Vault/D3VaultLiquidation.sol#L142C1-L148C31

```
uint256 realDebt = borrows.div(record.interestIndex == 0 ? 1e18 :
↪  record.interestIndex).mul(info.borrowIndex);
// if balance > realDebt, transferFrom realDebt instead of debt
IERC20(token).transferFrom(pool, address(this), realDebt);
```

hrishibhat

@IAm0x52

Attens1423

We have discovered some hidden issues in the dodo liquidation process, and we agree to modify the check of bad debts.

traceurl

fixed in https://github.com/DODOEX/new-dodo-v3/pull/52

IAm0x52

Fix looks good. Weight have been eliminated as suggested leading to more accurate calculations

Issue M-9: D3UserQuote#getUserQuote queries incorrect token for exchangeRate leading to inaccurate quota calculations

Source: https://github.com/sherlock-audit/2023-06-dodo-judging/issues/193

Found by

0x4db5362c, 0x52, dirk_y

Summary

A small typo in the valuation loop of D3UserQuote#getUserQuote uses the wrong variable leading to and incorrect quota being returned. The purpose of a quota is to mitigate risk of positions being too large. This incorrect assumption can dramatically underestimate the quota leading to oversized (and overrisk) positions.

Vulnerability Detail

D3UserQuota.sol#L75-L84

```
for (uint256 i = 0; i < tokenList.length; i++) {
    address _token = tokenList[i];
    (address assetDToken,,,,,,,,,) = d3Vault.getAssetInfo(_token);
    uint256 tokenBalance = IERC20(assetDToken).balanceOf(user);
    if (tokenBalance > 0) {
        tokenBalance = tokenBalance.mul(d3Vault.getExchangeRate(token)); <-
        ↪  @audit-issue queries token instead of _token
        (uint256 tokenPrice, uint8 priceDecimal) =
        ↪  ID3Oracle(d3Vault._ORACLE_()).getOriginalPrice(_token);
        usedQuota = usedQuota + tokenBalance * tokenPrice / 10 **
        ↪  (priceDecimal+tokenDecimals);
    }
}
```

D3UserQuota.sol#L80 incorrectly uses token rather than _token as it should. This returns the wrong exchange rate which can dramatically alter the perceived token balance as well as the calculated quota.

Impact

Quota is calculated incorrectly leading to overly risky positions, which in turn can cause loss to the system

Code Snippet

D3UserQuota.sol#L69-L97

Tool used

Manual Review

Recommendation

Change variable from token to _token:

```
-           tokenBalance = tokenBalance.mul(d3Vault.getExchangeRate(token));
+           tokenBalance = tokenBalance.mul(d3Vault.getExchangeRate(_token));
```

Discussion

traceurl

We redesigned D3UserQuota.

In the old version:

1. used quota is calculated based on the USD value of the deposited token
2. global quota is shared by all tokens

In this new version:

1. used quota is the amount of the deposited token, so price change won't affect quota
2. each token has its own global quota

traceurl

fixed in https://github.com/DODOEX/new-dodo-v3/pull/36

IAm0x52

Fix looks good. Changes have been made to quota so that it is no longer valued in USD but instead the nominal amount of token. Small nitpick. Make sure to adjust token quotas to have the correct number of decimals. In test cases I still see values like 100 and 1,000 which are much too small

Issue M-10: Protocol is completely incompatible with USDT due to lack of 0 approval

Source: https://github.com/sherlock-audit/2023-06-dodo-judging/issues/203

Found by

0x4db5362c, 0x52, MohammedRizwan, PRAISE, Sulpiride, Vagner, jprod15, kutugu, shealtielanz, shogoki, tsvetanovv

Summary

USDT will revert if the current allowance is greater than 0 and an non-zero approval is made. There are multiple instances throughout the contracts where this causes issues. In some places this can create scenarios where it becomes impossible to liquidate and/or borrow it.

Vulnerability Detail

See summary.

Impact

USDT may become impossible to liquidate or borrow

Code Snippet

D3Funding.sol#L20-L23

D3Funding.sol#L50-L53

D3Funding.sol#L64-L67

D3MMLiquidationRouter.sol#L24

Tool used

Manual Review

Recommendation

Utilize the OZ safeERC20 library and safeApprove

Discussion

osmanozdemir1

Escalate for 10 USDC

I agree with "approve to 0 first" is one of the most common bugs in the space but I think it is invalid for this protocol.

Approval calls to token contracts are made from pool contracts in this protocol, like borrow() and updateReserveByVault() functions.

Before calling the approve function, the allowance of the vault is checked and the call is made if `allowance < type(uint256).max`. Then the approve is called and the allowance is set to type(uint256).max.

- If a pool tries to borrow for the first time, allowance is set to `type(uint256).max`.
- Allowance will not decrease after that ever even if the vault uses that allowance. The reason for that is according to ERC20, allowances will not decrease after spending if the allowance was `type(uint256).max`.
- If the pool tries to borrow again, there won't be a problem as allowance is already `type(uint256).max`, and this call will not be made.

I believe the issue is invalid for this protocol as the approval is not expected to be used by EOAs. A pool can not directly approve to a non-zero value because it doesn't have a function to make that call. That's why I don't see a scenario where the allowance is not zero, and not `type(uint256).max` at the same time. It has to be either 0 or `type(uint256).max`, and this won't cause function to revert.

sherlock-admin2

> Escalate for 10 USDC
>
> I agree with "approve to 0 first" is one of the most common bugs in the space but I think it is invalid for this protocol.
>
> Approval calls to token contracts are made from pool contracts in this protocol, like borrow() and updateReserveByVault() functions.
>
> Before calling the approve function, the allowance of the vault is checked and the call is made if `allowance < type(uint256).max`. Then the approve is called and the allowance is set to type(uint256).max.
>
> - If a pool tries to borrow for the first time, allowance is set to `type(uint256).max`.
> - Allowance will not decrease after that ever even if the vault uses that allowance. The reason for that is according to ERC20, allowances will not decrease after spending if the allowance was `type(uint256).max`.

- If the pool tries to borrow again, there won't be a problem as allowance is already `type(uint256).max`, and this call will not be made.

> I believe the issue is invalid for this protocol as the approval is not expected to be used by EOAs. A pool can not directly approve to a non-zero value because it doesn't have a function to make that call. That's why I don't see a scenario where the allowance is not zero, and not `type(uint256).max` at the same time. It has to be either 0 or `type(uint256).max`, and this won't cause function to revert.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

marie-fourier

@osmanozdemir1 isn't the allowance check done wrong in the line 50? It should be `allowance(address(this), state._D3_VAULT_)`, otherwise it will call approve on each call of `updateReserveByVault`, so I think the issue is valid.

osmanozdemir1

> @osmanozdemir1 isn't the allowance check done wrong in the line 50? It should be `allowance(address(this), state._D3_VAULT_)`, otherwise it will call approve on each call of `updateReserveByVault`, so I think the issue is valid.

Oh, I see. You're definitely right about that, and this makes it problematic due to the wrong implementation in line 50. Root cause is wrong allowance check and I totally missed it. Thanks for the comment.

hrishibhat

Result: Medium Has duplicates

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- osmanozdemir1: rejected

traceurl

fixed in https://github.com/DODOEX/new-dodo-v3/pull/30

IAm0x52

Fix looks good

Issue M-11: Calculation B0 meets devision 0 error when a token has small decimal and high price with a small kBid

Source: https://github.com/sherlock-audit/2023-06-dodo-judging/issues/241

Found by

Protocol Team When a token has small decimal and high price, like wbtc, it will generate a low-precision bid price. In _calB0WithPriceLimit, https://github.com/sherlock-audit/2023-06-dodo/blob/a8d30e611acc9762029f8756d6a5b81825faf348/new-dodo-v3/contracts/DODOV3MM/lib/PMMRangeOrder.sol#L141C1-L148C10

```
if (k == 0) {
    baseTarget = amount;
} else {
    uint256 temp1 = priceUp + DecimalMath.mul(i, k) - i;
    uint256 temp2 = DecimalMath.mul(i, k);
    uint256 temp3 = DecimalMath.div(temp1, temp2);
    uint256 temp5 = DecimalMath.sqrt(temp3) - ONE;

    baseTarget = amount + DecimalMath.div(amount, temp5);
```

small bid price and kBid will cause temp2 =0, so that temp3 couldn't be calculated rightly.

Vulnerability Detail

Here is poc

```
function testQueryFail() public {
     token1ChainLinkOracle.feedData(30647 * 1e18);
     token2ChainLinkOracle.feedData(1 * 1e18);
     vm.startPrank(maker);
     uint32[] memory tokenKs = new uint32[](2);
     tokenKs[0] = 0;
     tokenKs[1] = (1<< 16) +1;
     address[] memory tokens = new address[](2);
     tokens[0] = address(token2);
     tokens[1] = address(token1);
     address[] memory slotIndex = new address[](2);
     slotIndex[0] = address(token1);
     slotIndex[1] = address(token2);
     uint80[] memory priceSlot = new uint80[](2);
     priceSlot[0] = 2191925019632266903652;
     priceSlot[1] = 720435765840878108682;
```

```
    uint64[] memory amountslot = new uint64[](2);
    amountslot[0] = stickAmount(10,8, 400000, 18);
    amountslot[1] = stickAmount(400000, 18, 400000, 18);
    d3MakerWithPool.setTokensKs(tokens, tokenKs);
    d3MakerWithPool.setTokensPrice(slotIndex, priceSlot);
    d3MakerWithPool.setTokensAmounts(slotIndex, amountslot);
    vm.stopPrank();

    (uint256 askDownPrice, uint256 askUpPrice, uint256 bidDownPrice, uint256
↪   bidUpPrice, uint256 swapFee) =
        d3MM.getTokenMMPriceInfoForRead(address(token1));
    assertEq(askDownPrice, 3045550280000000000000000000000000);
    assertEq(askUpPrice, 3072319000000000000000000000000000);
    assertEq(bidDownPrice, 3291);
    assertEq(bidUpPrice, 3320);
    assertEq(swapFee, 1200000000000000);

    //console.log(askDownPrice);
    //console.log(askUpPrice);
    //console.log(bidDownPrice);
    //console.log(bidUpPrice);
    //console.log(swapFee);

    (,,uint kask, uint kbid,,) =
↪   d3MM.getTokenMMOtherInfoForRead(address(token1));
    assertEq(kask, 1e14);
    assertEq(kbid, 1e14);

    (askDownPrice, askUpPrice, bidDownPrice, bidUpPrice, swapFee) =
        d3MM.getTokenMMPriceInfoForRead(address(token2));
    assertEq(askDownPrice, 999999960000000000);
    assertEq(askUpPrice, 1000799800000000000);
    assertEq(bidDownPrice, 1000400120032008002);
    assertEq(bidUpPrice, 1001201241249250852);
    assertEq(swapFee, 200000000000000);

    (,,kask, kbid,,) = d3MM.getTokenMMOtherInfoForRead(address(token2));
    assertEq(kask, 0);
    assertEq(kbid, 0);

    //console.log(askDownPrice);
    //console.log(askUpPrice);
    //console.log(bidDownPrice);
    //console.log(bidUpPrice);
    //console.log(swapFee);
    //console.log(kask);
    //console.log(kbid);
```


```
        SwapCallbackData memory swapData;
        swapData.data = "";
        swapData.payer = user1;

        //uint256 gasleft1 = gasleft();
        uint256 receiveToToken = d3Proxy.sellTokens(
            address(d3MM),
            user1,
            address(token1),
            address(token2),
            1000000,
            0,
            abi.encode(swapData),
            block.timestamp + 1000
        );
```

It will revert. In this example, wbtc price is 30445, and k = 0.0001, suppose maker contains rules, but model is invalid.

Impact

Maker sets right parameters but traders can't swap. It will make swap model invalid.

Tool Used

Manual Review

Recommendation

1. Fix formula for this corner case, like making temp2 = 1
2. Improve calculation accuracy by consistently using precision 18 for calculations and converting to real decimal when processing amounts.

Discussion

Attens1423

fix pr: https://github.com/DODOEX/new-dodo-v3/pull/32

hrishibhat

Please note: This issue is not part of the contest submissions and is not eligible for contest rewards.

IAm0x52

Fix looks good. PMMRangeOrder#querySellTokens now utilizes 18 dp normalized amounts for calculations. Oracle has an additional function to specifically accommodate token amounts being normalized.

Issue M-12: It's dangerous for makers to set token decimal by manual

Source: https://github.com/sherlock-audit/2023-06-dodo-judging/issues/242

Found by

Protocol Team

Vulnerability Detail

https://github.com/sherlock-audit/2023-06-dodo/blob/a8d30e611acc9762029f8756d6a5b81825faf348/new-dodo-v3/contracts/DODOV3MM/D3Pool/D3Maker.sol#L158C1-L178C57

Contract does not check decimal but record the variable directly.

Impact

If maker entered wrong decimal, swap will transfer more or less amount than expected.

Tool Used

Manual Review

Recommandation

In setNewToken function, taking token decimal through token's interface rather than entering by maker.

```
state.tokenMMInfoMap[token].decimal = IERC20(token).decimal();
```

Discussion

Attens1423

in new calculation model, we don't need token decimal anymore. fix pr:https://github.com/DODOEX/new-dodo-v3/pull/32

hrishibhat

Please note: This issue is not part of the contest submissions and is not eligible for contest rewards.

IAm0x52

Fix looks good. Token decimals are now queried directly rather than relying a manual input

Issue M-13: When swapping 18-decimal token to 8-decimal token , user could buy decimal-18-token with 0 amount of decimal-8-token

Source: https://github.com/sherlock-audit/2023-06-dodo-judging/issues/245

Found by

Protocol Team

Vulnerability Detail

Here is the poc:

```
uint256 payFromToken = d3Proxy.buyTokens(
            address(d3MM),
            user1,
            address(token1),
            address(token2),
            10000000,
            0,
            abi.encode(swapData),
            block.timestamp + 1000
        );
assertEq(payFromToken, 0);
```

Impact

It may cause unexpected loss

Tool Used

Manual Review

Recommendation

In buyToken() of D3Trading.sol, add this rule:

```
if(payFromAmount == 0) { // value too small
            payFromAmount = 1;
        }
```

Discussion

Attens1423

https://github.com/DODOEX/new-dodo-v3/pull/37

hrishibhat

Please note: This issue is not part of the contest submissions and is not eligible for contest rewards.

IAm0x52

Fix looks good. Previously if swapping a lower precision token, the precision loss could be abused to by small amounts of higher DP tokens, though the amount gained would almost always be too small for any abuse profitably. 0 amount from token swaps are now always rounded up to 1.

Issue M-14: liquidationTarget is not set when removing pool.

Source: https://github.com/sherlock-audit/2023-06-dodo-judging/issues/246

Found by

Protocol Team Remove pool has 3 steps:

1. call removeD3Pool(address) and force the pool into liquidation state. However, the vault call ID3MM(pool).startLiquidation() directly and skip setting liquidationTarget
2. DODO team uses liquidationByDODO() to balance all tokens.
3. call finishPoolRemove() and finish removing.

The bugs could happen in step 2. During liquidationByDODO, liquidator only rebalance tokens in pool. It can't update vault borrow recording. If dodo team want to call finishLiquidation(), they will find liquidationTarget wasn't set and they could not balance vault borrow recording.

Impact

It may not remove pool successfully.

Tool Used

Manual Review

Recommandation

1. Calculation liquidationTarget in removeD3Pool()
2. call finishLiquidation() to finish pool balance and then call finishPoolRemove() Thinking of balance > debt situation, it could transfer realDebt into vault: https://github.com/sherlock-audit/2023-06-dodo/blob/a8d30e611acc9762029f8756d6a5b81825faf348/new-dodo-v3/contracts/DODOV3MM/D3Vault/D3VaultLiquidation.sol#L141C1-L145C67

```
// note: During liquidation process, the pool's debt will slightly increase due
 ↪  to the generated interests.
// The liquidation process will not repay the interests. Thus all dToken holders
 ↪  will share the loss equally.
uint256 realDebt = borrows.div(record.interestIndex == 0 ? 1e18 :
 ↪  record.interestIndex).mul(info.borrowIndex);
IERC20(token).transferFrom(pool, address(this), debt);
```

change into:

```
IERC20(token).transferFrom(pool, address(this), realDebt);
```

Discussion

hrishibhat

Please note: This issue is not part of the contest submissions and is not eligible for contest rewards.

traceurl

@Attens1423 We don't need liquidationTarget when removing pool.

Issue M-15: borrow amount recorded in AssetInfo and BorrowRecord unmatched due to the precision loss

Source: https://github.com/sherlock-audit/2023-06-dodo-judging/issues/248

Found by

Protocol Team

If the token has small decimals, like WBTC, which has 8 decimals, since borrowIndex is larger than 1e18, during accrueInterest, assetInfo[token].totalBorrows will loss more precision than assetInfo[token].borrowIndex. Eventually the assetInfo[token].totalBorrows will be smaller than the total sum of BorrowRecord.amount.

Vulnerability Detail

POC

```
/*

    Copyright 2023 DODO ZOO.
    SPDX-License-Identifier: Apache-2.0

*/

pragma solidity 0.8.16;

import "forge-std/Test.sol";
import "../../TestContext.t.sol";

contract POCTest is TestContext {
    D3UserQuota public d3UserQutoa;

    MockERC20 public wbtc;
    address public wbtcAddr;
    address public dodoAddr;

    function setUp() public {
        contextBasic();

        wbtc = token1;
        dodo = token3;
        wbtcAddr = address(token1);
        dodoAddr = address(token3);

        wbtc.mint(user1, 1000e8);
```

```
        vm.prank(user1);
        wbtc.approve(address(dodoApprove), type(uint256).max);

        wbtc.mint(poolCreator, 1000e8);
        vm.prank(poolCreator);
        wbtc.approve(address(dodoApprove), type(uint256).max);

        dodo.mint(poolCreator, 1000000 ether);
        vm.prank(poolCreator);
        dodo.approve(address(dodoApprove), type(uint256).max);

        d3UserQutoa = new D3UserQuota(address(token4), address(d3Vault));
        vm.prank(vaultOwner);
        d3Vault.setNewD3UserQuota(address(d3UserQutoa));

        vm.prank(vaultOwner);
        d3Vault.addLiquidator(address(this));
        vm.prank(vaultOwner);
        d3Vault.addLiquidator(liquidator);
    }

    function testPOC() public {
        vm.prank(user1);
        d3Proxy.userDeposit(user1, wbtcAddr, 100e8, 90e8);

        // make dodo price high
        token3ChainLinkOracle.feedData(1e9 * 1e18);

        // pool1 deposit high price dodo, and borrow wbtc
        vm.prank(poolCreator);
        d3Proxy.makerDeposit(address(d3MM), dodoAddr, 1000 ether);
        vm.prank(poolCreator);
        d3MM.borrow(wbtcAddr, 10e8);

        // make dodo price low
        token3ChainLinkOracle.feedData(1 * 1e18);

        vm.warp(3600 * 2);
        d3Vault.accrueInterests(); // This is the key step. The test will pass
↪   if this line is comment out.

        vm.warp(3600 * 3);

        wbtc.burn(address(d3MM), 9e8);
        d3MM.updateReserve(wbtcAddr);

        d3Vault.startLiquidation(address(d3MM));
        liquidateSwap(address(d3MM), dodoAddr, wbtcAddr, 1000 ether);
```


```
        d3Vault.finishLiquidation(address(d3MM));
    }

    function testPOC2() public {
        vm.prank(user1);
        d3Proxy.userDeposit(user1, wbtcAddr, 100e8, 90e8);

        // make dodo price high
        token3ChainLinkOracle.feedData(1e9 * 1e18);

        // pool1 deposit high price dodo, and borrow wbtc
        vm.prank(poolCreator);
        d3Proxy.makerDeposit(address(d3MM), dodoAddr, 1000 ether);
        vm.prank(poolCreator);
        d3MM.borrow(wbtcAddr, 10e8);

        uint256 i = 1;
        while (i < 365 * 2) {
            vm.warp(3600 * 12 * i);
            console.log("d3Vault.accrueInterests()...");
            d3Vault.accrueInterest(wbtcAddr);
            i++;
        }

        vm.prank(poolCreator);
        d3Proxy.makerDeposit(address(d3MM), wbtcAddr, 1e8);
        vm.prank(poolCreator);
        d3MM.repayAll(wbtcAddr);
    }
}
```

Impact

The last pool's repay() / finishLiquidation() will be blocked, since the record.amount will be larger than the info.borrowAmount, which will cause a underflow in `info.totalBorrows = info.totalBorrows - realDebt;`

Tool Used

Manual Review

Recommandation

Check if info.totalBorrows is smaller than the debt. If info.totalBorrows is smaller than the debt, then make info.totalBorrows to 0.

Discussion

traceurl

fixed in https://github.com/DODOEX/new-dodo-v3/pull/55

IAm0x52

Fixes look good. D3VaultFunding and D3VaultLiquidation have both been updated to fix this by setting to 0 instead of underflowing.