SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for: DODO
Prepared by: Sherlock
Lead Security Expert: rogueredddwarf
Dates Audited: May 9 - May 12, 2023
Prepared on: July 18, 2023

Introduction

This contest enables margin trading enabled by Aave for the popular DODO trading protocol, powered by the Proactive Market Making (PMM) algorithm

Scope

Repository: DODOEX/dodo-margin-trading-contracts

Branch: main

Commit: f6279954cdfb48824c5186cbb86a200db2cddff5

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
1	1

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

VAD37
rvierdiiev
pengun
rogueredwarf
BAHOZ
sashik_eth
CRYP70
oot2k
Bauer
carrotsmuggler
nobody2018
shaka

0xHati
smiling_heretic
Tendency
curiousapple
shogoki
qbs
theOwl
Juntao
BowTiedOriole
Jiamin
Quantish
circlelooper
alexzoid

SHERLOCK

Issue H-1: MarginTrading.sol: Missing flash loan initiator check allows attacker to open trades, close trades and steal funds

Source: https://github.com/sherlock-audit/2023-05-dodo-judging/issues/34

Found by

0xHati, BAHOZ, Bauer, BowTiedOriole, CRYP70, Jiamin, Juntao, Quantish, Tendency, VAD37, alexzoid, carrotsmuggler, circlelooper, curiousapple, nobody2018, oot2k, pengun, qbs, rogue_reddwarf, rvierdiiev, sashik_eth, shaka, shogoki, smiling_heretic, theOwl

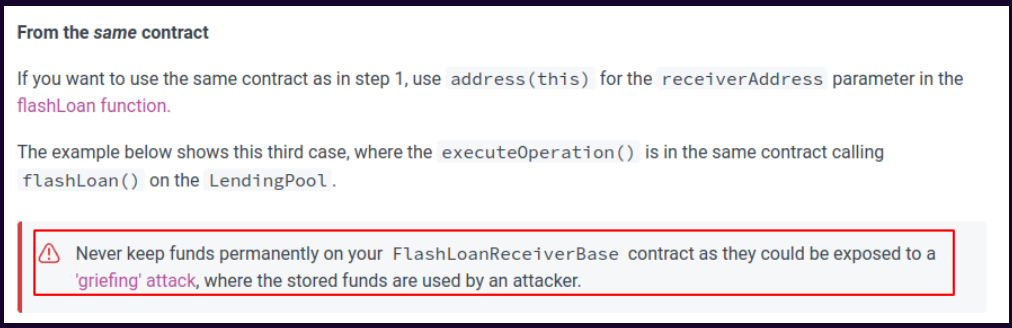
Summary

The `MarginTrading.executeOperation` function is called when a flash loan is made (and it can only be called by the `lendingPool`).

The wrong assumption by the protocol is that the flash loan can only be initiated by the `MarginTrading` contract itself.

However this is not true. A flash loan can be initiated for any `receiverAddress`.

This is actually a known mistake that devs make and the aave docs warn about this (although admittedly the warning is not very clear):
https://docs.aave.com/developers/v/2.0/guides/flash-loans



From the *same* contract

If you want to use the same contract as in step 1, use `address(this)` for the `receiverAddress` parameter in the flashLoan function.

The example below shows this third case, where the `executeOperation()` is in the same contract calling `flashLoan()` on the `LendingPool`.

⚠ Never keep funds permanently on your `FlashLoanReceiverBase` contract as they could be exposed to a 'griefing' attack, where the stored funds are used by an attacker.

So an attacker can execute a flash loan with the `MarginTrading` contract as `receiverAddress`. Also the funds that are needed to pay back the flash loan are pulled from the `receiverAddress` and NOT from the `initiator`:

https://github.com/aave/protocol-v2/blob/30a2a19f6d28b6fb8d26fc07568ca0f2918f4070/contracts/protocol/lendingpool/LendingPool.sol#L532-L536

This means the attacker can close a position or repay a position in the `MarginTrading` contract.

By crafting a malicious swap, the attacker can even steal funds.

Vulnerability Detail

Let's assume there is an ongoing trade in a `MarginTrading` contract:

```
daiAToken balance = 30000
wethDebtToken balance = 10

The price of WETH when the trade was opened was ~ 3000 DAI
```

In order to profit from this the attacker does the following (not considering fees for simplicity):

1. Take a flash loan of 30000 DAI with `MarginTrading` as `receiverAddress` with `mode=0` (flash loan is paid back in the same transaction)
2. Price of WETH has dropped to 2000 DAI. The attacker uses a malicious swap contract that pockets 10000 DAI for the attacker and swaps the remaining 20000 DAI to 10 WETH (the attacker can freely choose the swap contract in the `_params` of the flash loan).
3. The 10 WETH debt is repaid
4. Withdraw 30000 DAI from Aave to pay back the flash loan

Impact

The attacker can close trades, partially close trades and even steal funds.

(Note: It's not possible for the attacker to open trades because he cannot incur debt on behalf of the `MarginTrading` contract)

Code Snippet

https://github.com/sherlock-audit/2023-05-dodo/blob/main/dodo-margin-trading-contracts/contracts/marginTrading/MarginTrading.sol#L121-L166

https://github.com/aave/protocol-v2/blob/30a2a19f6d28b6fb8d26fc07568ca0f2918f4070/contracts/protocol/lendingpool/LendingPool.sol#L481-L562

Tool used

Manual Review

Recommendation

The fix is straightforward:

```
diff --git
 ↪  a/dodo-margin-trading-contracts/contracts/marginTrading/MarginTrading.sol
 ↪  b/dodo-margin-trading-contracts/contracts/marginTrading/MarginTrading.sol
index f68c1f3..5b4b485 100644
--- a/dodo-margin-trading-contracts/contracts/marginTrading/MarginTrading.sol
+++ b/dodo-margin-trading-contracts/contracts/marginTrading/MarginTrading.sol
@@ -125,6 +125,7 @@ contract MarginTrading is OwnableUpgradeable,
 ↪  IMarginTrading, IFlashLoanReceiver
         address _initiator,
         bytes calldata _params
     ) external override onlyLendingPool returns (bool) {
+        require(_initiator == address(this));
         //decode params exe swap and deposit
         {
```

This ensures that the flash loan has been initiated by the `MarginTrading.executeFlashLoans` function which is the intended initiator.

Discussion

Zack995

I agree with the severity and error of this issue

- `require(_initiator == address(this));`

We will adopt this recommendation to address the issue.

securitygrid

Escalate for 10 USDC I don't think the Vulnerability Detail of this report is correct. The key to this attack is the parameter. I wouldn't escalate it if it was a simple parameter. This is a complex parameter, especially when you want to steal the erc20 in MarginTrading or repay it and then steal the collateral. To repay for MarginTrading, the most important thing is: in the code of aave pool, msg.sender must be a borrower, that is, the MarginTrading contract itself. For detailed parameter encoding, see #69 and #74.

This escalatation is not intended to invalidate this issue. **The purposes are as follows**:

1. A best report usually comes from a big name. But in fact it is not the best. Should non-best reports appear in the final report?
2. I am thinking whether the report submitted in the future should be as simple as possible, which saves a lot of time. If a problem leads to an online attack

specific to the protocol, I always describe its steps as clearly as possible in the report. However, it seems like this doesn't make sense.

3. This is also a question that has been confusing everyone, and has not been answered. This is about multiple significant effects caused by the same code. Should separate reports be submitted or should different impacts be merged into one report? Some impacts cause DOS, some cause the core function logic to be destroyed, and the more serious one is the loss of funds.
4. In the judge contest, many reports may be classified just by looking at the title (this is my guess). Because the number of submissions is huge, it will make the judge too tired. Is the judge's bonus too small?
5. Are we doing security audit or functional audit? What is sherlock more concerned about?

sherlock-admin

Escalate for 10 USDC I don't think the Vulnerability Detail of this report is correct. The key to this attack is the parameter. I wouldn't escalate it if it was a simple parameter. This is a complex parameter, especially when you want to steal the erc20 in MarginTrading or repay it and then steal the collateral. To repay for MarginTrading, the most important thing is: in the code of aave pool, msg.sender must be a borrower, that is, the MarginTrading contract itself. For detailed parameter encoding, see #69 and #74.

This escalatation is not intended to invalidate this issue. **The purposes are as follows**:

1. A best report usually comes from a big name. But in fact it is not the best. Should non-best reports appear in the final report?
2. I am thinking whether the report submitted in the future should be as simple as possible, which saves a lot of time. If a problem leads to an online attack specific to the protocol, I always describe its steps as clearly as possible in the report. However, it seems like this doesn't make sense.
3. This is also a question that has been confusing everyone, and has not been answered. This is about multiple significant effects caused by the same code. Should separate reports be submitted or should different impacts be merged into one report? Some impacts cause DOS, some cause the core function logic to be destroyed, and the more serious one is the loss of funds.
4. In the judge contest, many reports may be classified just by looking at the title (this is my guess). Because the number of submissions is huge, it will make the judge too tired. Is the judge's bonus too small?

5. Are we doing security audit or functional audit? What is sherlock more concerned about?

You've created a valid escalation for 10 USDC!

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.

ctf-sec

The escalation aims to seek clarity, which is very important, recommend composing a thoughtful response on sherlock end and accept the escalation no matter what the response is!

hrishibhat

Result: High Has duplicates The questions raised on the escalations are not relevant here and will be considered and addressed separately

sherlock-admin

Escalations have been resolved successfully!

Escalation status:

- securitygrid: rejected

Attens1423

fix pr:https://github.com/DODOEX/dodo-margin-trading-contracts/pull/1

roguereddwarf

Mitigation Review: The root cause of this issue was a missing check that only a flash loan initiated by the `MarginTrading` contract can call the `MarginTrading.executeOperation` function.

The fix recommended by the Watson has been implemented and the issue is now fixed.

Issue M-1: MarginTrading.sol: The whole balance and not just the traded funds are deposited into Aave when a trade is opened

Source: https://github.com/sherlock-audit/2023-05-dodo-judging/issues/72

Found by

roguereddwarf

Summary

It's expected by the protocol that funds can be in the `MarginTrading` contract without being deposited into Aave as margin.

We can see this by looking at the `MarginTradingFactory.depositMarginTradingETH` and `MarginTradingFactory.depositMarginTradingERC20` functions.

If the user sets `margin=false` as the parameter, the funds are only sent to the `MarginTrading` contract but NOT deposited into Aave.

https://github.com/sherlock-audit/2023-05-dodo/blob/main/dodo-margin-trading-contracts/contracts/marginTrading/MarginTradingFactory.sol#L203-L211

https://github.com/sherlock-audit/2023-05-dodo/blob/main/dodo-margin-trading-contracts/contracts/marginTrading/MarginTradingFactory.sol#L259-L272

So clearly there is the expectation for funds to be in the `MarginTrading` contract that should not be deposited into Aave.

This becomes an issue when a trade is opened.

Vulnerability Detail

Let's look at the `MarginTrading._openTrade` function that is called when a trade is opened:

https://github.com/sherlock-audit/2023-05-dodo/blob/main/dodo-margin-trading-contracts/contracts/marginTrading/MarginTrading.sol#L257-L279

The whole balance of the token will be deposited into Aave:

```
_tradeAmounts[i] = IERC20(_tradeAssets[i]).balanceOf(address(this));
_lendingPoolDeposit(_tradeAssets[i], _tradeAmounts[i], 1);
```

Not just those funds that have been acquired by the swap. This means that funds that should stay in the `MarginTrading` contract might also be deposited as margin.

Impact

When opening a trade funds can be deposited into Aave unintentionally. Thereby the funds act as margin and the trade can incur a larger loss than expected.

Code Snippet

https://github.com/sherlock-audit/2023-05-dodo/blob/main/dodo-margin-trading-contracts/contracts/marginTrading/MarginTradingFactory.sol#L203-L211

https://github.com/sherlock-audit/2023-05-dodo/blob/main/dodo-margin-trading-contracts/contracts/marginTrading/MarginTradingFactory.sol#L259-L272

https://github.com/sherlock-audit/2023-05-dodo/blob/main/dodo-margin-trading-contracts/contracts/marginTrading/MarginTrading.sol#L257-L279

Tool used

Manual Review

Recommendation

It is necessary to differentiate the funds that are acquired by the swap and those funds that were there before and should stay in the contract:

```
diff --git
 ↪  a/dodo-margin-trading-contracts/contracts/marginTrading/MarginTrading.sol
 ↪  b/dodo-margin-trading-contracts/contracts/marginTrading/MarginTrading.sol
index f68c1f3..42f96cf 100644
--- a/dodo-margin-trading-contracts/contracts/marginTrading/MarginTrading.sol
+++ b/dodo-margin-trading-contracts/contracts/marginTrading/MarginTrading.sol
@@ -261,6 +261,10 @@ contract MarginTrading is OwnableUpgradeable,
 ↪  IMarginTrading, IFlashLoanReceiver
         bytes memory _swapParams,
         address[] memory _tradeAssets
     ) internal {
+        int256[] memory _amountsBefore = new uint256[](_tradeAssets.length);
+        for (uint256 i = 0; i < _tradeAssets.length; i++) {
+            _amountsBefore[i] =
 ↪  IERC20(_tradeAssets[i]).balanceOf(address(this));
+        }
         if (_swapParams.length > 0) {
             // approve to swap route
             for (uint256 i = 0; i < _swapApproveToken.length; i++) {
@@ -272,8 +276,10 @@ contract MarginTrading is OwnableUpgradeable,
 ↪  IMarginTrading, IFlashLoanReceiver
         }
         uint256[] memory _tradeAmounts = new uint256[](_tradeAssets.length);
```

```
         for (uint256 i = 0; i < _tradeAssets.length; i++) {
-            _tradeAmounts[i] = IERC20(_tradeAssets[i]).balanceOf(address(this));
-            _lendingPoolDeposit(_tradeAssets[i], _tradeAmounts[i], 1);
+            if (_amountsBefore[i] <
 ↪  IERC20(_tradeAssets[i]).balanceOf(address(this))) {
+                _tradeAmounts[i] =
 ↪  IERC20(_tradeAssets[i]).balanceOf(address(this)) - _amountsBefore[i];
+                _lendingPoolDeposit(_tradeAssets[i], _tradeAmounts[i], 1);
+            }
         }
         emit OpenPosition(_swapAddress, _swapApproveToken, _tradeAssets,
 ↪  _tradeAmounts);
     }
```

If funds that were in the contract prior to the swap should be deposited there is the separate `MarginTrading.lendingPoolDeposit` function to achieve this.

Discussion

Zack995

In terms of product design, users do not have a separate concept of balance. However, the contract is designed to be more flexible and allows for balances to be maintained. Users will not perceive or interact with balances in terms of user experience or operations.

roguereddwarf

Based on the smart contract logic there is clearly the notion of balance that is not intended to be used as collateral (but e.g. used to repay a loan). If this notion of a separate balance is not exposed on the front-end this is not a sufficient mitigation of the issue since the issue is clearly present in the smart contract.

securitygrid

Escalate for 10 USDC This is valid low/info as stated by the sponsor. No bad impact.

sherlock-admin

> Escalate for 10 USDC This is valid low/info as stated by the sponsor. No bad impact.

You've created a valid escalation for 10 USDC!

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.

ctf-sec

can consider #80 duplicate of this one

hrishibhat

Result: Medium Has duplicates Considering this issue as valid medium based on the above comments from smart contract perspective and enforcing in the front end is not a mitigation as mentioned above.

sherlock-admin

Escalations have been resolved successfully!

Escalation status:

- securitygrid: rejected

Attens1423

In fact we need tokens are deposited in aave. We suppose users use frontend to operate their assets and frontend will ensure all assets will be deposited in aave. If there are tokens left, users could deposit them in aave by themselves. So we think this design is not a problem.

MLON33

Interpreting sponsor's comment as "Won't Fix."

roguereddwarf

Mitigation Review: While Sherlock considers this a valid "Medium", the sponsor decided that there is no need to implement a fix.